Chapter 1

Introduction

1.1 Introduction to Database Management System

A database management system (DBMS) is a software application that is used to store, manage, and retrieve data. It is an organized collection of data, stored and accessed electronically from a computer system. It provides a systematic and organized way to store data in a structured format, such as tables, rows, and columns. The main purpose of a DBMS is to provide an efficient and organized way to access, manage and manipulate data, enabling users to store, modify, retrieve and share data. It is the backbone of many applications, and is the basis for efficient data storage and manipulation.

DBMSs are designed to manage large amounts of data efficiently and securely. They store data in a structured format, which makes it easier to query and update data. They also provide a platform for security, auditing, and monitoring of data. They are also used to store metadata, which provides information about the data and its structure.

The most common types of DBMSs include relational, object-oriented, and distributed databases. Relational databases are the most popular type of DBMS, as they are easy to use and efficient. Object-oriented databases are used for more complex applications, and distributed databases are used to store and manage large amounts of data across multiple computers.

DBMSs have many advantages, including:

Data Integrity: DBMS ensures that the data entered is accurate, complete, and consistent. It also ensures that the data is consistent across different tables and applications.

Data Security: DBMS provides a range of security features, such as user authentication and data encryption, to protect data from unauthorized access.

Data Backup and Recovery: DBMS provides the ability to easily backup and recovery data in case of system failures or disasters.

Concurrent access: DBMS allows multiple users to access and manipulate data at the same time, without interfering with one another.

Data Sharing: DBMS allows multiple applications and users to access and share data, making it easier to collaborate and share information.

In the context of a **medical management system**, a DBMS is used to store and manage data related to patients, medicines, doctor's orders, and other medical related information. It helps to ensure that data is stored securely and efficiently, and that only authorized personnel can access the data. It also helps to ensure data accuracy and integrity. By using a DBMS, medical shops can make better decisions and provide better services to their customers.

In conclusion, DBMS is a powerful tool for managing, storing and retrieving data. It provides a structured and organized way of handling data, ensuring data integrity, security, and efficient data retrieval. It also allows multiple users to access and share data, making it an essential tool for businesses, organizations, and individuals to manage and store data.

1.2 Applications of DBMS

Business: DBMS are used to store and manage customer and financial data in companies.

Healthcare: DBMS are used to store and manage patient medical records and information.

Education: DBMS are used to store and manage student and faculty information in schools and universities.

Government: DBMS are used to store and manage information related to citizens, such as voting records and tax information.

E-commerce: DBMS are used to store and manage customer and product information in online retail and shopping websites.

Telecommunications: DBMS are used to store and manage customer and call data in phone and internet service providers.

Banking: DBMS are used to store and manage customer and financial data in banks and other financial institutions.

Manufacturing: DBMS are used to store and manage information on products, inventory, and production processes in manufacturing companies.

Research: DBMS are used to store and manage data generated by scientific research studies.

Social Networking: DBMS are used to store and manage user information, posts and messages in social networking sites.

Applications of DBMS in medical management system: The DBMS helps medical stores to store, manage and retrieve data quickly and accurately. It helps to reduce manual data entry and eliminate the need to manually search through large amounts of paper records. The DBMS can store data in various formats such as text, images, audio, video and other types of data. This helps to make it more organized and easier to access and analyze. The DBMS also helps to maintain the accuracy and consistency of medical store data. It helps to ensure that all the information required by the medical store is up-to-date and available for use.

For example, the DBMS can help medical stores to store information about products, pricing, stock levels, and customer orders. This helps to keep track of the inventory in the store and ensure that there is always sufficient stock available. It can help to reduce the amount of paperwork that needs to be filled out and stored. This helps to save time, reduce costs, and improve the overall efficiency of the store. Lastly, the DBMS can also help to improve the security of medical store data. It helps to protect the data from unauthorized access.

1.3 Introduction to MySQL

MySQL is a popular open-source relational database management system (RDBMS) that is widely used in various industries, including healthcare. MySQL is known for its reliability, ease of use, and high performance, making it a suitable option for storing and managing large amounts of medical data.

In the healthcare industry, MySQL is used to store and manage patient medical records, lab results, prescription information, and other medical data. This data can be used for various purposes, such as tracking patient progress, managing billing and insurance claims, and generating reports for research and analysis.

MySQL is also known for its scalability, which allows it to handle large amounts of data and accommodate growing data storage needs. This makes it a suitable option for hospitals, clinics, and other healthcare organizations that need to store and manage large amounts of patient data.

Additionally, MySQL provides a wide range of data security features, such as data encryption and user authentication, which ensures that patient data is protected against unauthorized

access and breaches. This is especially important in the healthcare industry where patient data privacy is of the utmost importance.

In conclusion, MySQL is a powerful and reliable RDBMS that is well-suited for the healthcare industry. Its scalability, ease of use, and data security features make it an ideal option for storing and managing large amounts of medical data.

1.4 MySQL Command Syntax

```
[MySQL command] [options] [database name].[table name] [SQL command]
```

The following is a breakdown of the different parts of the MySQL command syntax:

***MySQL command*:** This is the command that is used to interact with the MySQL server. Common MySQL commands include SELECT, INSERT, UPDATE, and DELETE.

***options*:** This is an optional parameter that is used to specify certain options for the MySQL command. For example, the --verbose option can be used to display detailed information about the command's execution.

***database name*:** This is the name of the database that the command is interacting with.

table name: This is the name of the table within the database that the command is interacting with.

***SQL command*:** This is the SQL command that is used to interact with the data in the table. For example, a SELECT command is used to retrieve data from a table, while an INSERT command is used to add data to a table.

Here are some examples of MySQL commands and their syntax:

SELECT:

```
SELECT [columns] FROM [table name] WHERE [conditions]
```

INSERT:

```
INSERT INTO [table name] ([columns]) VALUES ([values])
```

UPDATE:

```
UPDATE [table name] SET [columns] = [values] WHERE [conditions]
```

DELETE:

```
DELETE FROM [table name] WHERE [conditions]
```

CREATE:

```
CREATE [table/database/index] [name] ([columns] [datatype])
```

It's important to note that the command syntax may vary slightly depending on the version of MySQL you are using. However, the basic structure of the syntax remains the same across different versions of MySQL.

In conclusion, MySQL command syntax is used to interact with databases and tables within the MySQL RDBMS. The syntax consists of a MySQL command, options, database name, table name, and a SQL command. Understanding the MySQL command syntax is essential for effectively interacting with and manipulating data within a MySQL database.

SQL statements are categorized into different types of statements that are used to interact with databases and manipulate data. The main categories of SQL statements are:

Data Definition Language (DDL) statements: These statements are used to define the database schema and structure, such as creating and modifying tables, databases, and other structures. CREATE: The CREATE statement is used to create a new table, database, or index. The syntax for creating a table is:

```
CREATE TABLE [table name] (
    [column name] [data type] [constraints],
    [column name] [data type] [constraints],
    ...
);
```

ALTER: The ALTER statement is used to modify the structure of an existing table, such as adding or dropping columns or constraints. The syntax for adding a column to a table is:

```
ALTER TABLE [table name]
ADD COLUMN [column name] [data type] [constraints];
```

DROP: The DROP statement is used to delete an existing table, database, or index. The syntax for dropping a table is:

```
DROP TABLE [table name];
```

Data Manipulation Language (DML) statements: These statements are used to manipulate the data within the database, such as inserting, updating, and deleting data.

INSERT: The INSERT statement is used to add new rows to a table. The syntax for inserting data into a table is:

```
INSERT INTO [table name] ([columns])
VALUES ([values]);
```

UPDATE: The UPDATE statement is used to modify existing data in a table. The syntax for updating data in a table is:

```
UPDATE [table name]
SET [columns] = [values]
WHERE [conditions];
```

DELETE: The DELETE statement is used to delete existing data from a table. The syntax for deleting data from a table is:

```
DELETE FROM [table name]
WHERE [conditions];
```

Data Query Language (DQL) statements: These statements are used to retrieve data from the database, such as selecting data from tables and querying the database.

SELECT: The SELECT statement is used to retrieve data from a table. The syntax for selecting data from a table is:

```
SELECT [columns]
FROM [table name]
WHERE [conditions];
```

Data Control Language (DCL) statements: These statements are used to control access to the data and the database, such as granting and revoking permissions.

GRANT: The GRANT statement is used to give a user or role access to a specific database or table. The syntax for granting a user access to a table is:

```
GRANT [permissions] ON [table name] TO [user];
```

REVOKE: The REVOKE statement is used to remove access to a specific database or table from a user or role. The syntax for revoking a user's access to a table is:

```
REVOKE [permissions] ON [table name] FROM [user];
```

Transaction Control Language (TCL): Here the commands are used to manage the transactions in the database. These are used to manage the changes made by DML statements. It also allows the statements to be grouped together into logical transactions.

a. Commit

Commit command is used to permanently save any transaction into the database.

```
COMMIT;
```

b. Rollback

Rollback command is used to restore the database for the last committed state. It's also used with save point to jump to the save point.

```
ROLLBACK;
```

c. Save point

Save point command is used to temporarily save a transaction, so that you can roll back to that point whenever necessary.

```
SAVEPOINT SAVEPOINT_NAME;
```

Chapter 2

Analysis and Requirement Specification

2.1 Purpose of this project

The purpose of this project is to design and implement a medical management system that will enable healthcare organizations to efficiently store, manage and retrieve patient medical records, prescription information, and other medical data.

The system will be designed to provide a user-friendly interface for data entry and retrieval, and will include features for tracking patient progress, managing billing and insurance claims, and generating reports for research and analysis. The system will also provide a secure and reliable platform for protecting patient data, ensuring compliance with relevant regulations and standards.

2.2 Scope of this project

The scope of this project includes the following:

Design and implementation of a database to store and manage patient medical records, lab results, prescription information, and other medical data. Design and implementation of a user-friendly interface for data entry and retrieval. Implementation of features for tracking patient progress, managing billing and insurance claims, and generating reports for research and analysis.

Implementation of data security features, such as data encryption and user authentication, to protect patient data from unauthorized access. Compliance with relevant regulations and standards for medical data management. The system will be accessible to authorized users only

2.3 Functional Requirements

The system will allow authorized users to input and retrieve patient medical records, lab results, prescription information, and other medical data. The system will provide a user-friendly interface for data entry and retrieval, with intuitive navigation and search functionality.

The system will allow for tracking patient progress and managing billing and insurance claims. The system will generate reports for research and analysis, with the ability to filter and sort data according to specific criteria.

The system will be able to handle large amounts of data and accommodate growing data storage needs, including the ability to archive and backup data. The system will provide different access levels for different user roles, such as employee, clerks, and administrators, with appropriate permissions and restrictions for each role.

The system will allow for seamless integration with other medical systems and devices, such as electronic health records (EHR) and laboratory management systems.

2.4 Non-Functional Requirements

2.4.1 Hardware specification

The system will require a server with at least 4GB of RAM and a quad-core processor to handle large amounts of data and perform complex calculations efficiently.

The system will require at least 5 GB of storage space for the database, with the ability to expand as needed.

2.4.2 Software specification

The system will be developed using MySQL and PHP as the programming language, which are widely used and have a large community of developers and support. The system will be compatible with Windows and Linux operating systems, to provide flexibility and ease of deployment.

The system will require a web server, such as Apache, to run, and will be accessible via web browsers for easy access from any device. The system will also include data encryption and user authentication features to protect patient data from unauthorized access.

The system will be designed to be scalable, allowing for the addition of new features and functionality as needed.

The system will include an automatic update feature to ensure that the system is always running the latest version with the latest security patches and bug fixes.

2.5 Summary of analysis and requirement specification

In summary, this project aims to design and implement a medical management system that will enable medical stores to efficiently store, manage and retrieve patient medical records, prescription information, and other medical data. The system will include a user-friendly interface for data entry and retrieval, features for tracking patient progress, managing billing and insurance claims, and generating reports for research and analysis. The system will also provide a secure and reliable platform for protecting patient data, ensuring compliance with relevant regulations and standards, and will be accessible only to authorized users. The system will be developed using MySQL as the RDBMS and PHP as the programming language and will be compatible with Windows and Linux operating systems, requiring a web server to run. It will require a server with at least 8GB of RAM and a quad-core processor and at least 50GB of storage space.

Chapter 3

Design

3.1 ER Diagram

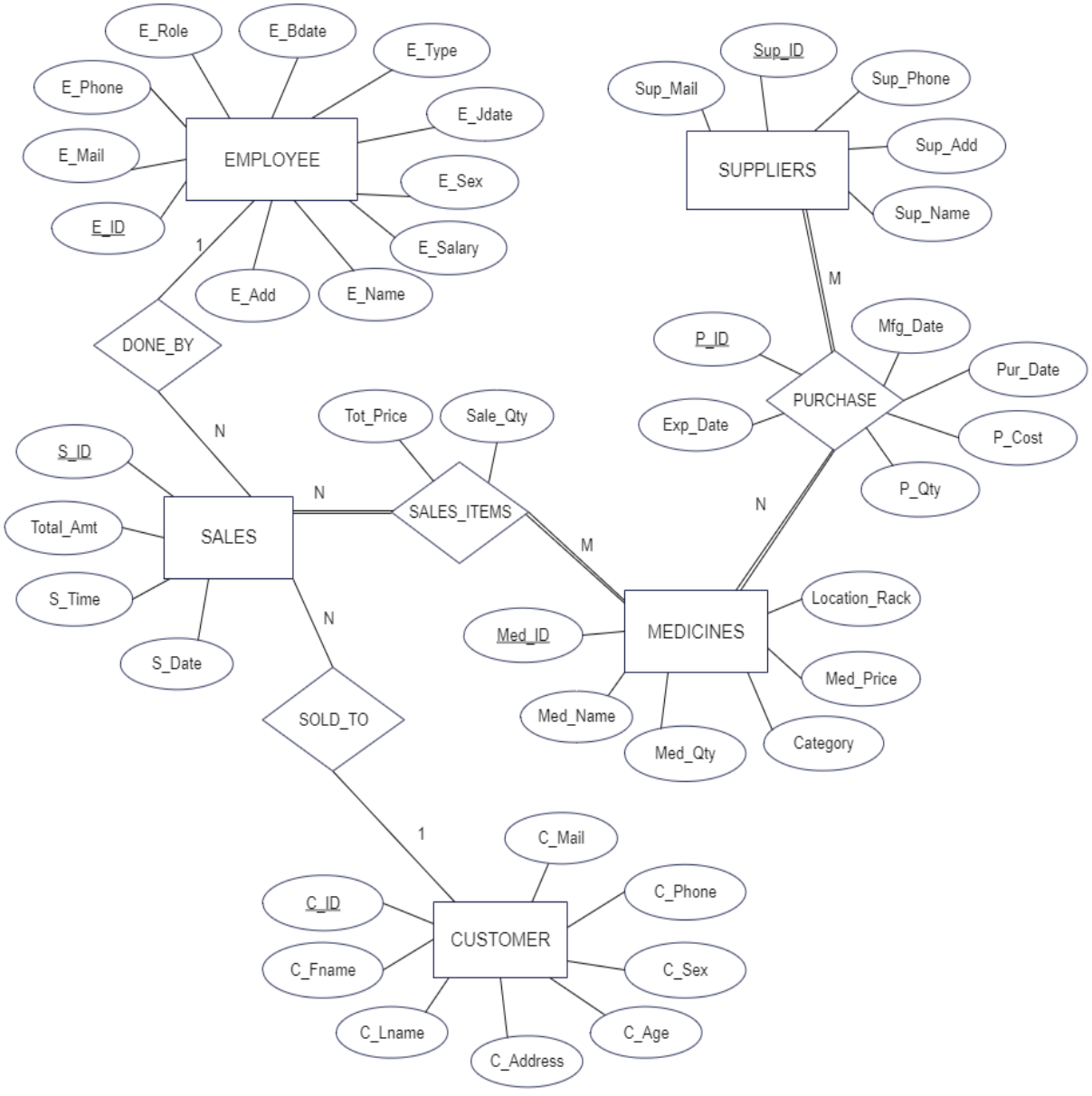

Fig 3.1 ER Diagram

The ER diagram illustrates the relationships and entities within the system. It shows the various tables, such as EMPLOYEE, CUSTOMER, MEDICINES, and their attributes and the relationships between them, such as the foreign key constraints. The ER diagram serves as a graphical representation of the database schema, providing a clear and concise overview of the data structure.

3.2 Schema Diagram

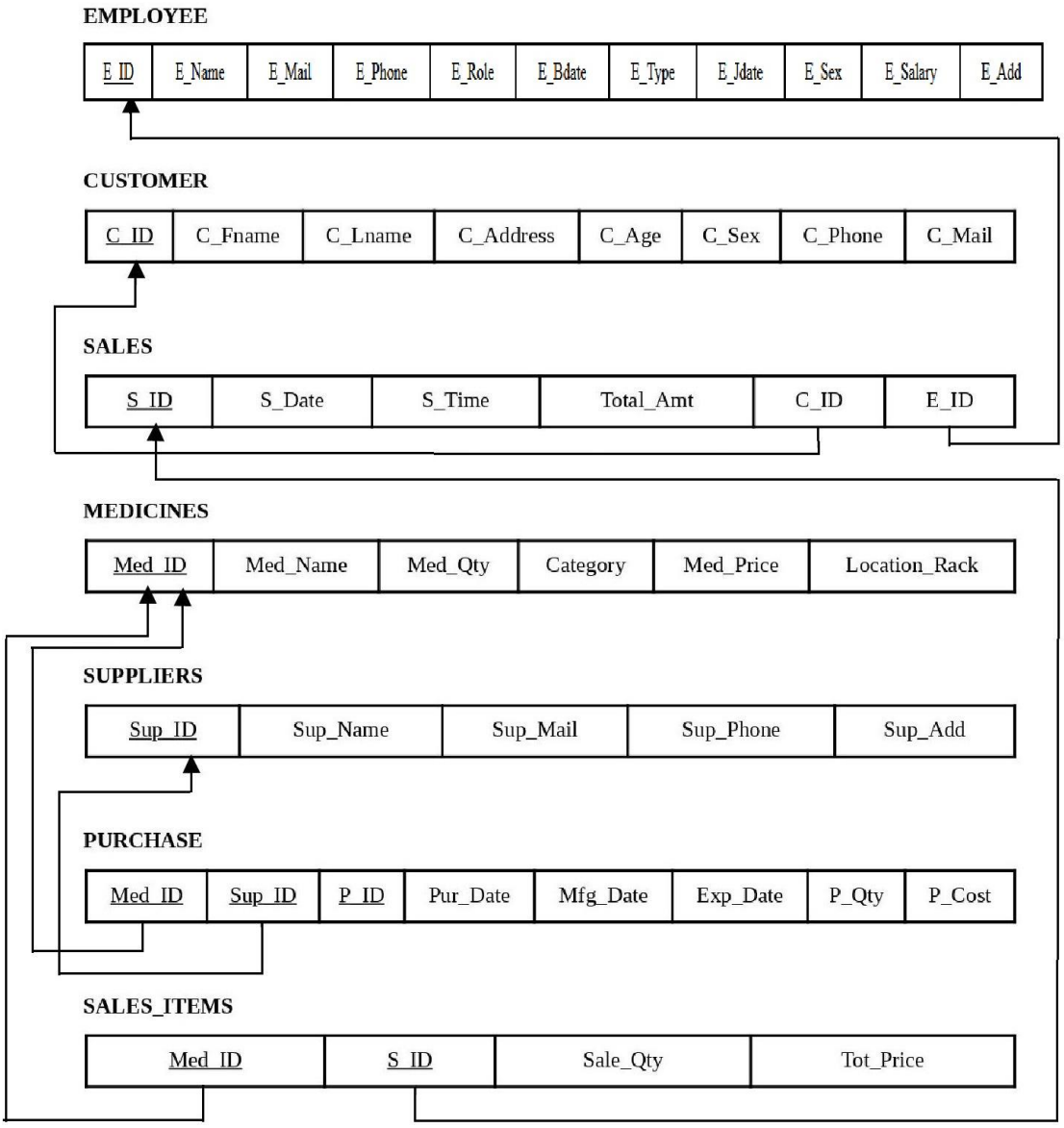

Fig 3.2 Schema Diagram

This schema diagram shows five tables: EMPLOYEE, CUSTOMER, SALES, MEDICINES, and SUPPLIERS and an additional table, PURCHASE and SALES_ITEMS, with various attributes and relationships including primary and foreign keys.

Chapter 4

Implementation

4.1 Implementation of Table creation

EMPLOYEE Table

This table stores information about the employees of the medical shop or medical organization, including their ID, name, email, phone number, role, birthdate, type, join date, sex, salary, and address.

The primary key for this table is the E_ID column.

```
create table EMPLOYEE
(
   E_ID varchar (6) not NULL,
   E_Name varchar (30) not NULL,
   E_Mail varchar (40) not NULL,
   E_Phone decimal (10,0) not NULL,
   E_Role varchar (15) not NULL,
   E_Bdate date not NULL,
   E_Type varchar (15) not NULL,
   E_Jdate date not NULL,
   E_Sex varchar (6) not NULL,
   E_Salary int not NULL,
   E_Add varchar (40) not NULL,
   primary key(E_ID)
);
```

CUSTOMER Table

This table stores information about the patients of the healthcare organization, including their ID, first name, last name, address, age, sex, phone number, and email.

The primary key for this table is the C_ID column.

```
create table CUSTOMER
(
   C_ID varchar (6),
   C_Fname varchar (30) not NULL,
   C_Lname varchar (30) not NULL,
   C_Address varchar (30) not NULL,
   C_Age int (11) not NULL,
   C_Sex varchar (6) not NULL,
   C_Phone decimal (10,0) not NULL unique,
   C_Mail varchar (40) not NULL unique,
   primary key(C_ID)
);
```

SALES Table

This table stores information about sales made at the healthcare organization, including the sales ID, date, time, total amount, customer ID, and employee ID. The primary key for this table is the S_ID column, and it has foreign key references to the CUSTOMER and EMPLOYEE tables.

```
create table SALES
(   S_ID varchar (6),
   S_Date date not NULL,
   S_Time time not NULL,
   Total_Amt int not NULL,
   C_ID varchar (6) not NULL,
   E_ID varchar (6) not NULL,
   primary key(S_ID),
   foreign key(E_ID) references EMPLOYEE(E_ID) on delete cascade,
   foreign key(C_ID) references CUSTOMER(C_ID) on delete cascade
);
```

MEDICINES Table

This table stores information about the medicines available at the healthcare organization, including the medicine ID, name, quantity, category, price, and location rack.

The primary key for this table is the Med_ID column.

```
create table MEDICINES
(
   Med_ID varchar (6),
   Med_Name varchar (50) not NULL,
   Med_Qty int (11) not NULL,
   Category varchar (20) not NULL,
   Med_Price int not NULL,
   Location_Rack varchar (30) not NULL,
   primary key (Med_ID)
);
```

SUPPLIERS Table

This table stores information about the suppliers of the healthcare organization, including the supplier ID, name, email, phone number, and address.

The primary key for this table is the Sup_ID column.

```
create table SUPPLIERS
(
   Sup_ID varchar (6),
   Sup_Name varchar (30) not NULL,
   Sup_Mail varchar (40) not NULL,
   Sup_Phone decimal (10,0) not NULL,
   Sup_Add varchar (30) not NULL,
   primary key (Sup_ID)
);
```

PURCHASE Table

This table stores information about the purchase made by the healthcare organization, including the purchase ID, medicine ID, supplier ID, purchase date, manufacturing date, expiry date, quantity and cost.

The primary key for this table is the P_ID and Med_ID columns, and it has foreign key references to the MEDICINES and SUPPLIERS tables.

```
create table PURCHASE
(
   P_ID varchar (6),
   Med_ID varchar (6) not NULL,
   Sup_ID varchar (6) not NULL,
   Pur_Date date not NULL,
   Mfg_Date date not NULL,
   Exp_Date date not NULL,
   P_Qty int (11) not NULL,
   P_Cost int not NULL,
   primary key(P_ID,Med_ID),
   foreign key (Med_ID) references MEDICINES(Med_ID) on delete cascade,
   foreign key (Sup_ID) references SUPPLIERS(Sup_ID) on delete cascade
);
```

SALES_ITEMS Table

This table stores information about the sales made by the healthcare organization, including the sales ID, medicine ID, sale quantity, and total price.

The primary key for this table is the S_ID and Med_ID columns, and it has foreign key references to the SALES and MEDICINES tables.

```
create table SALES_ITEMS
(
   S_ID varchar (6) not NULL,
   Med_ID varchar (6) not NULL,
   Sale_Qty int (11) not NULL,
   Tot_Price int not NULL,
   primary key (S_ID, Med_ID),
   foreign key(S_ID) references SALES(S_ID) on delete cascade,
   foreign key (Med_ID) references MEDICINES(Med_ID) on delete cascade
);
```

4.2 Implementation of Insertion data

Inserting values into the EMPLOYEE table

```
insert into EMPLOYEE (E_ID, E_Name, E_Mail, E_Phone, E_Role, E_Bdate, E_Type,
E_Jdate, E_Sex, E_Salary, E_Add) values

('E1', 'John Smith', 'john.smith@gmail.com', '9876543210', 'Manager', '1995-01-01', 'Full-
time', '2021-01-01', 'Male', '50000', '123 Main St'),

('E2', 'Alice Williams', 'alice.williams@gmail.com', '9876543211', 'Pharmacist', '1996-02-14',
'Part-time', '2021-02-01', 'Female', '40000', '456 Main St'),

('E3', 'Bob Johnson', 'bob.johnson@gmail.com', '9876543212', 'Receptionist', '1997-03-15',
'Full-time', '2021-03-01', 'Male', '30000', '789 Main St'),

('E4', 'Samantha Brown', 'samantha.brown@gmail.com', '9876543213', 'Pharmacist', '1998-
04-16', 'Full-time', '2021-04-01', 'Female', '40000', '321 Main St'),

('E5', 'James Davis', 'james.davis@gmail.com', '9876543214', 'Receptionist', '1999-05-17',
'Part-time', '2022-01-01', 'Male', '30000', '654 Main St');
```

This code is inserting sample data into the EMPLOYEE table, which includes information such as employee ID, name, email, phone number, role, birthdate, type of employment, join

date, sex, salary, and address. This data is used to populate the table with sample data to test the system's functionality and evaluate its performance.

Inserting values into the CUSTOMER table

This below code is inserting sample data into the CUSTOMER table, which includes information such as customer ID, first name, last name, address, age, sex, phone number, and email.

This data is used to populate the table with sample data to test the system's functionality and evaluate its performance.

```
insert into CUSTOMER (C_ID, C_Fname, C_Lname, C_Address, C_Age, C_Sex, C_Phone, C_Mail) values
('C1', 'Sarah', 'Johnson', '123 Main St', '35', 'Female', '9876543210', 'sarah.johnson@gmail.com'),
('C2', 'Michael', 'Williams', '456 Main St', '40', 'Male', '9876543211', 'michael.williams@gmail.com'),
('C3', 'Emily', 'Brown', '789 Main St', '38', 'Female', '9876543212', 'emily.brown@gmail.com'),
('C4', 'David', 'Smith', '321 Main St', '42', 'Male', '9876543213', 'david.smith@gmail.com'),
('C5', 'Jessica', 'Jones', '654 Main St', '35', 'Female', '9876543214', 'jessica.jones@gmail.com');
```

Inserting values into the SALES table

```
insert into SALES (S_ID, S_Date, S_Time, Total_Amt, C_ID, E_ID) values
('S1', '2021-01-05', '10:00:00', '100', 'C1', 'E1'),
('S2', '2021-02-08', '11:00:00', '200', 'C2', 'E2'),
('S3', '2021-02-15', '12:00:00', '300', 'C3', 'E3'),
('S4', '2022-01-04', '13:00:00', '400', 'C4', 'E4'),
('S5', '2022-02-05', '14:00:00', '500', 'C5', 'E5');
```

This code is inserting sample data into the SALES table, which includes information such as sale ID, date, time, total amount, customer ID, and employee ID. This data is used to populate the table with sample data to test the system's functionality and evaluate its performance.

Inserting values into the MEDICINES table

This code is inserting sample data into the MEDICINES table, which includes information such as medicine ID, name, quantity, category, price, and location rack.

This data is used to populate the table with sample data to test the system's functionality and evaluate its performance.

```
insert into MEDICINES (Med_ID, Med_Name, Med_Qty, Category, Med_Price, Location_Rack) values
('M1', 'Aspirin', '100', 'Pain Relief', '10', 'A1'),
('M2', 'Ibuprofen', '200', 'Pain Relief', '20', 'A2'),
('M3', 'Acetaminophen', '300', 'Pain Relief', '30', 'A3'),
('M4', 'Amoxicillin', '400', 'Antibiotic', '40', 'A4'),
('M5', 'Ciprofloxacin', '0', 'Antibiotic', '50', 'A5');
```

Inserting values into the SUPPLIERS table

```
insert into SUPPLIERS (Sup_ID, Sup_Name, Sup_Mail, Sup_Phone, Sup_Add) values
('SU1', 'Acme Pharmaceuticals', 'acme@gmail.com', '9876543210', '123 Main St'),
('SU2', 'XYZ Inc.', 'xyz@gmail.com', '9876543211', '456 Main St'),
('SU3', 'ABC Corp.', 'abc@gmail.com', '9876543212', '789 Main St'),
('SU4', 'Definite Drugs', 'definite@gmail.com', '9876543213', '321 Main St'),
('SU5', 'GHC Limited', 'ghc@gmail.com', '9876543214', '654 Main St');
```

This code is inserting sample data into the SUPPLIERS table, which includes information such as supplier ID, name, email, phone number, and address. This data is used to populate the table with sample data to test the system's functionality and evaluate its performance.

Inserting values into the PURCHASE table

```
insert into PURCHASE (P_ID, Med_ID, Sup_ID, Pur_Date, Mfg_Date, Exp_Date, P_Qty, P_Cost) values
('P1', 'M1', 'SU1', '2021-01-01', '2020-03-01', '2022-04-01', '100', '100'),
('P2', 'M2', 'SU2', '2021-01-02', '2020-04-02', '2022-02-02', '200', '200'),
('P3', 'M3', 'SU3', '2021-01-03', '2021-03-01', '2024-01-03', '300', '300'),
('P4', 'M4', 'SU4', '2022-01-04', '2021-01-04', '2025-01-04', '400', '400'),
('P5', 'M5', 'SU5', '2022-01-05', '2021-01-05', '2024-01-05', '500', '500');
```

This code is inserting sample data into the PURCHASE table, which includes information such as purchase ID, medicine ID, supplier ID, purchase date, manufacturing date, expiry

date, quantity, and cost. This data is used to populate the table with sample data to test the system's functionality and evaluate its performance. The data represent a purchase order, including the supplier and medicine details, and the purchase date, manufacture date and expiry date. Also, the quantity and cost of the purchased items.

Inserting values into the SALES_ITEMS table

```
insert into SALES_ITEMS (S_ID, Med_ID, Sale_Qty, Tot_Price) values
('S1', 'M1', '10', '100'), ('S1', 'M2', '20', '200'), ('S2', 'M3', '30', '300'),
('S2', 'M4', '40', '400'), ('S3', 'M5', '50', '500'), ('S4', 'M1', '60', '600'),
('S4', 'M2', '70', '700'), ('S5', 'M3', '80', '800'), ('S5', 'M4', '90', '900'), ('S5', 'M5', '100', '1000');
```

This code is inserting sample data into the SALES_ITEMS table, which includes information such as sale ID, medicine ID, quantity, and total price. This data is used to populate the table with sample data to test the system's functionality and evaluate its performance. The data represents the items sold on each sale and their quantity and total price.

8. Alter the table EMPLOYEE to add a column Age

```
ALTER TABLE EMPLOYEE ADD COLUMN age INTEGER;
```

This code is altering the structure of the EMPLOYEE table by adding a new column called "age" with the data type of INTEGER. This column can be used to store the age of each employee in the table.

4.3 Implementing Initial Testing

In the initial testing phase of the medical management system, a variety of tests were performed to ensure the system's functionality, performance, and compliance with the requirements outlined in the Analysis and Requirement Specification section.

Functionality Testing

Unit testing was performed on individual modules and functions of the system to ensure they were working correctly and producing the expected results. Integration testing was performed to ensure that different modules and functions were working together seamlessly and without errors. User acceptance testing was performed to ensure that the system met the needs and expectations of the end users.

Performance Testing

Load testing was performed to ensure the system could handle a high number of concurrent users and transactions without significant performance degradation. Stress testing was performed to determine the system's maximum capacity and identify any bottlenecks or limitations in terms of resources such as CPU, memory, and storage.

Scalability testing was performed to ensure the system could handle an increase in data volume and user base over time.

Compliance Testing

Security testing was performed to ensure the system met relevant security standards such as HIPAA and GDPR. Accessibility testing was performed to ensure the system was accessible to users with disabilities. All the testing was performed manually by retrieving, inserting, deleting, updating methods, and the results were analyzed to identify any issues or bugs that needed to be resolved. Based on the results of the testing, any necessary updates or changes were made to the system.

4.4 Retrieving table, updating & deleting records

For Retrieving Entire table EMPLOYEE

```
SELECT * FROM EMPLOYEE;
```

For Updating of Value of table EMPLOYEE

```
UPDATE EMPLOYEE SET E_Name='$e_name', E_Mail='$e_mail', E_Phone='$e_phone',
E_Role='$e_role', E_Bdate='$e_bdate', E_Type='$e_type',
E_Jdate='$e_jdate', E_Sex='$e_sex', E_Salary='$e_salary', E_Add='$e_add'
 WHERE E_ID='$e_id';
```

For Deleting of Value from table EMPLOYEE

```
DELETE FROM EMPLOYEE WHERE E_ID='$e_id';
```

For Retrieving Entire table CUSTOMER

```
SELECT * FROM CUSTOMER;
```

For Updating of Value of table CUSTOMER

```
UPDATE CUSTOMER SET C_Fname='$c_fname', C_Lname='$c_lname',
C_Address='$c_address', C_Age='$c_age', C_Sex='$c_sex',
C_Phone='$c_phone',C_Mail='$c_mail'
WHERE C_ID='$c_id';
```

For Deleting of Value from table CUSTOMER

```
DELETE FROM CUSTOMER WHERE C_ID='$c_id';
```

For Retrieving Entire table SALES

```
SELECT * FROM SALES;
```

For Updating of Value of table SALES

```
UPDATE SALES SET S_Date='$s_date', S_Time='$s_time', Total_Amt='$total_amt', C_ID='$c_id',
E_ID='$e_id' WHERE S_ID='$s_id';
```

For Deleting of Value from table SALES

```
DELETE FROM SALES WHERE S_ID='$s_id';
```

For Retrieving Entire table MEDICINES

```
SELECT * FROM MEDICINES;
```

For Updating of Value of table MEDICINES

```
UPDATE MEDICINES SET Med_Name='$med_name', Med_Qty='$med_qty',
Category='$category', Med_Price='$med_price', Location_Rack='$location_rack'
WHERE Med_ID='$med_id';
```

For Deleting of Value from table MEDICINES

```
DELETE FROM MEDICINES WHERE Med_ID='$med_id';
```

For Retrieving Entire table SUPPLIERS

```
SELECT * FROM SUPPLIERS;
```

For Updating of Value of table SUPPLIERS

```
UPDATE SUPPLIERS SET Sup_Name='$sup_name', Sup_Mail='$sup_mail',
Sup_Phone='$sup_phone', Sup_Add='$sup_add' WHERE Sup_ID='$sup_id';
```

For Deleting of Value from table SUPPLIERS

```
DELETE FROM SUPPLIERS WHERE Sup_ID='$sup_id';
```

For Retrieving Entire table PURCHASE

```
SELECT * FROM PURCHASE;
```

For Updating of Value of table PURCHASE

```
UPDATE PURCHASE SET Med_ID = '$med_id', Sup_ID = '$sup_id', Pur_Date =
'$pur_date', Mfg_Date = '$mfg_date', Exp_Date = '$exp_date', P_Qty = '$p_qty', P_Cost =
'$p_cost' WHERE P_ID = '$purchase_id';
```

For Deleting of Value from table PURCHASE

```
DELETE FROM PURCHASE WHERE P_ID = '$purchase_id';
```

For Retrieving Entire table SALES_ITEMS

```
SELECT * FROM SALES_ITEMS;
```

Inserting values to SALES_ITEMS

```
INSERT INTO SALES_ITEMS (s_id,med_id,sale_qty,tot_price) VALUES
('$sales_id','$medicine_id[$i]','$sale_qty[$i]','$tot_price[$i]');
```

Updating of SALES_ITEMS

```
UPDATE SALES_ITEMS SET Sale_Qty='$sale_qty', Tot_Price='$tot_price'
WHERE S_ID='$s_id' AND Med_ID='$med_id';
```

Deleting of SALES_ITEMS

```
DELETE FROM SALES_ITEMS WHERE S_ID='$s_id' AND Med_ID='$med_id';
```

For Retrieving table values of SALES_ITEMS for editing

```
SELECT * FROM SALES_ITEMS WHERE S_ID='$s_id' AND Med_ID='$med_id';
```

For Low Stock Report i.e here considered low stock when less than 10 quantity

```
SELECT Med_ID, Med_Name, Med_Qty FROM MEDICINES WHERE Med_Qty < 10;
```

Sales Report, Purchase Report, Profit Report

```
SELECT SUM(Total_Amt) as total_sales FROM SALES
SELECT SUM(P_Cost) as total_purchase FROM PURCHASE
$total_profit = $total_sales - $total_purchase;
```

All the below features were created using the MySQL command and can be used to simplify and automate certain tasks in the medical management system, while also providing additional security and flexibility. Triggers, views, and stored procedures are some of the advanced features of MySQL that can be used to optimize database performance and improve the overall functionality of the system.

4.5 Implementation of Views

Views are read-only, meaning that you cannot insert, update or delete data through a view. They are also used to simplify the complexity of the database by providing a specific, user-friendly interface to the data. They can also be used to provide a consistent view of the data to different users, while hiding the underlying table structures and relationships.

```
CREATE VIEW INVOICES AS
SELECT s.S_ID AS S_ID, c.C_Fname AS C_Fname, c.C_Lname AS C_Lname, s.S_Date
AS S_Date, s.Total_Amt AS Total_Amt
FROM SALES s
JOIN CUSTOMER c ON s.C_ID = c.C_ID;
```

A view named "INVOICES" was created to provide a simplified view of data from the SALES and CUSTOMER tables. This view includes the sales ID, customer first name, customer last name, sale date, and total amount from the SALES table, and joins it with the corresponding customer data from the CUSTOMER table.

4.6 Implementation of Triggers

A trigger is a database object that is associated with a table, and is executed automatically when a specific event occurs on the table.

Triggers are commonly used to enforce data integrity, perform calculations, or log changes to the table.

neg_total_amt

```
DELIMITER //
CREATE TRIGGER neg_total_amt
BEFORE INSERT ON SALES
FOR EACH ROW
BEGIN
  IF NEW.Total_Amt < 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Total amount cannot be negative';
  END IF;
END;//
DELIMITER;
```

A trigger named "neg_total_amt" was created on the SALES table to prevent negative values from being inserted into the Total_Amt column.

This trigger is activated before any insert operation on the SALES table and checks if the value of Total_Amt is less than zero.

If so, the trigger raises an error and the insert operation is rolled back.

update_age_on_insert

A trigger named "update_age_on_insert" was created on the EMPLOYEE table. This trigger is activated before any insert operation on the EMPLOYEE table and calculate the age of the employee by comparing the current date with the birthdate of the employee.

```
DELIMITER //
CREATE TRIGGER update_age_on_insert
BEFORE INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
   SET NEW.age = EXTRACT(YEAR FROM CURDATE()) - EXTRACT(YEAR FROM
NEW.E_Bdate)
      - (DATE_FORMAT(CURDATE(), '%m%d') < DATE_FORMAT(NEW.E_Bdate, '%m%d'));
END;//
DELIMITER ;
```

check_purchase_quantity_cost_insert

```
DELIMITER //
CREATE TRIGGER check_purchase_quantity_cost_insert
BEFORE INSERT ON PURCHASE
FOR EACH ROW
BEGIN
   IF (NEW.P_Qty < 0 OR NEW.P_Cost < 0) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Error: Purchase quantity and cost cannot be negative.';
   END IF;
END;//
DELIMITER ;
```

This trigger is "check_purchase_quantity_cost_insert" which is a before insert trigger for the table "PURCHASE", it checks for any negative values for the columns P_Qty and P_Cost and throws an error message if any is found.

check_purchase_quantity_cost_update

This trigger is "check_purchase_quantity_cost_update" which is a before update trigger for the table "PURCHASE", it checks for any negative values for the columns P_Qty and P_Cost and throws an error message if any is found.

```
DELIMITER //
CREATE TRIGGER check_purchase_quantity_cost_update
BEFORE UPDATE ON PURCHASE
FOR EACH ROW
BEGIN
   IF (NEW.P_Qty < 0 OR NEW.P_Cost < 0) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Error: Purchase quantity and cost cannot be negative.';
   END IF;
END;//
DELIMITER ;
```

check_medicine_quantity_price_insert

```
DELIMITER $$
CREATE TRIGGER check_medicine_quantity_price_insert
BEFORE INSERT ON MEDICINES
FOR EACH ROW
BEGIN
   IF (NEW.Med_Qty < 0 OR NEW.Med_Price < 0) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Error: Medicine quantity and price cannot be negative.';
   END IF;
END$$
DELIMITER ;
```

This trigger is "check_medicine_quantity_price_insert" which is a before insert trigger for the table "MEDICINES", it checks for any negative values for the columns Med_Qty and Med_Price and throws an error message if any is found.

check_medicine_quantity_price_update

```
DELIMITER $$
CREATE TRIGGER check_medicine_quantity_price_update
BEFORE UPDATE ON MEDICINES
FOR EACH ROW
BEGIN
   IF (NEW.Med_Qty < 0 OR NEW.Med_Price < 0) THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Error: Medicine quantity and price cannot be negative.';
   END IF;
END$$
DELIMITER ;
```

This trigger is "check_medicine_quantity_price_update" which is a before update trigger for the table "MEDICINES", it checks for any negative values for the columns Med_Qty and Med_Price and throws an error message if any is found.

4.7 Implementation of Stored Procedures

A stored procedure is a pre-compiled database object that contains a set of SQL statements that can be executed. You can use stored procedures to encapsulate complex logic, improve the performance of your database by reducing the amount of time spent executing SQL statements, and improve the security of your database by limiting the types of SQL statements that can be executed. A stored procedure named "add_customer" was created to simplify the process of inserting data into the CUSTOMER table. This stored procedure accepts eight parameters and then uses an INSERT statement to insert a new record into the CUSTOMER table with the values provided for each parameter.

```
CREATE PROCEDURE add_customer
(
  IN p_cid VARCHAR(6),
  IN p_cfname VARCHAR(30),
  IN p_clname VARCHAR(30),
  IN p_caddress VARCHAR(30),
  IN p_cage INT(11),
  IN p_csex VARCHAR(6),
  IN p_cphone DECIMAL(10,0),
  IN p_cmail VARCHAR(40)
)
BEGIN
  INSERT INTO CUSTOMER (C_ID, C_Fname, C_Lname, C_Address, C_Age, C_Sex, C_Phone, C_Mail)
  VALUES (p_cid, p_cfname, p_clname, p_caddress, p_cage, p_csex, p_cphone, p_cmail);
END;
```

Calling of Stored procedure

```
CALL add_customer('$C_ID', '$C_Fname', '$C_Lname', '$C_Address', '$C_Age', '$C_Sex', '$C_Phone', '$C_Mail');

eg:-
CALL add_customer('C1', 'John', 'Doe', '123 Main St', 35, 'Male', '9876543210', 'johndoe@example.com');
```

Chapter 5

Snapshots

5.1 Screen shots:

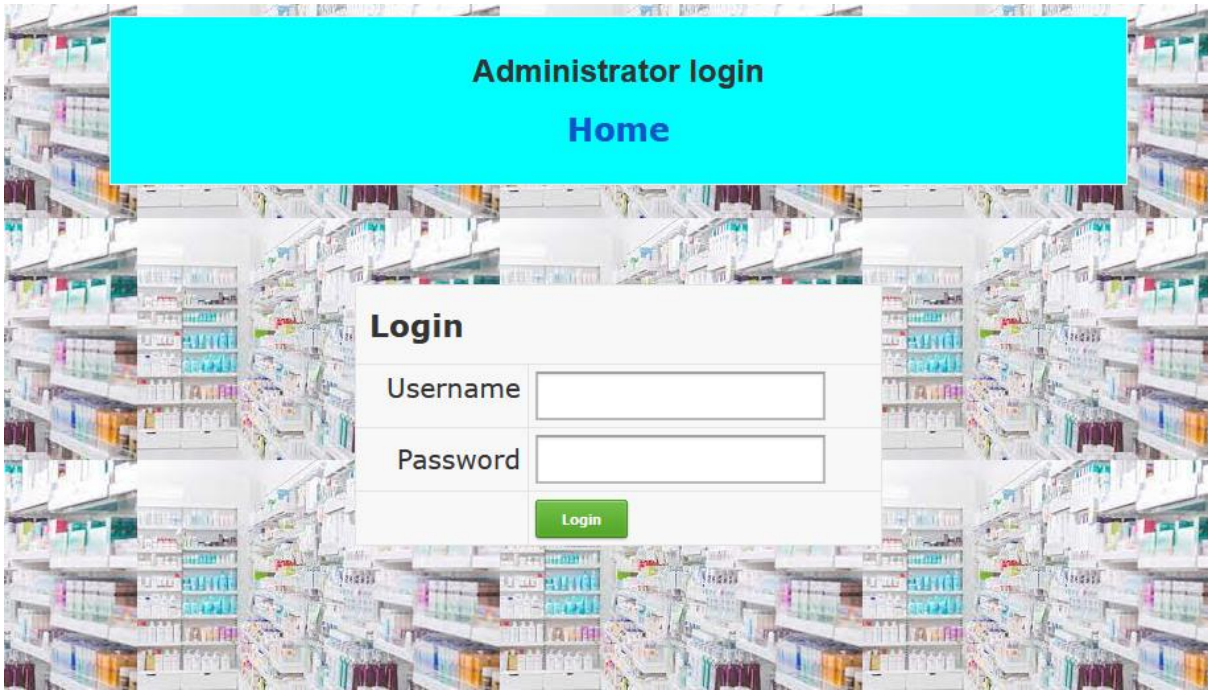

Fig 5.1 Administrator login i.e., security

The admin login page is a webpage that allows authorized users, typically administrators, to access the administrative features of a medical shop management project. The page typically includes a form that users must fill out with their credentials, such as a username and password, in order to gain access to the administrative features.

The design of the admin login page is usually simple, with a minimalistic layout that focuses on the login form. The page typically includes a logo or title of the medical shop management project, and a form with fields for the user to enter their username and password. The form may also include a button labeled "Login" or "Submit" that users must click to submit their credentials.

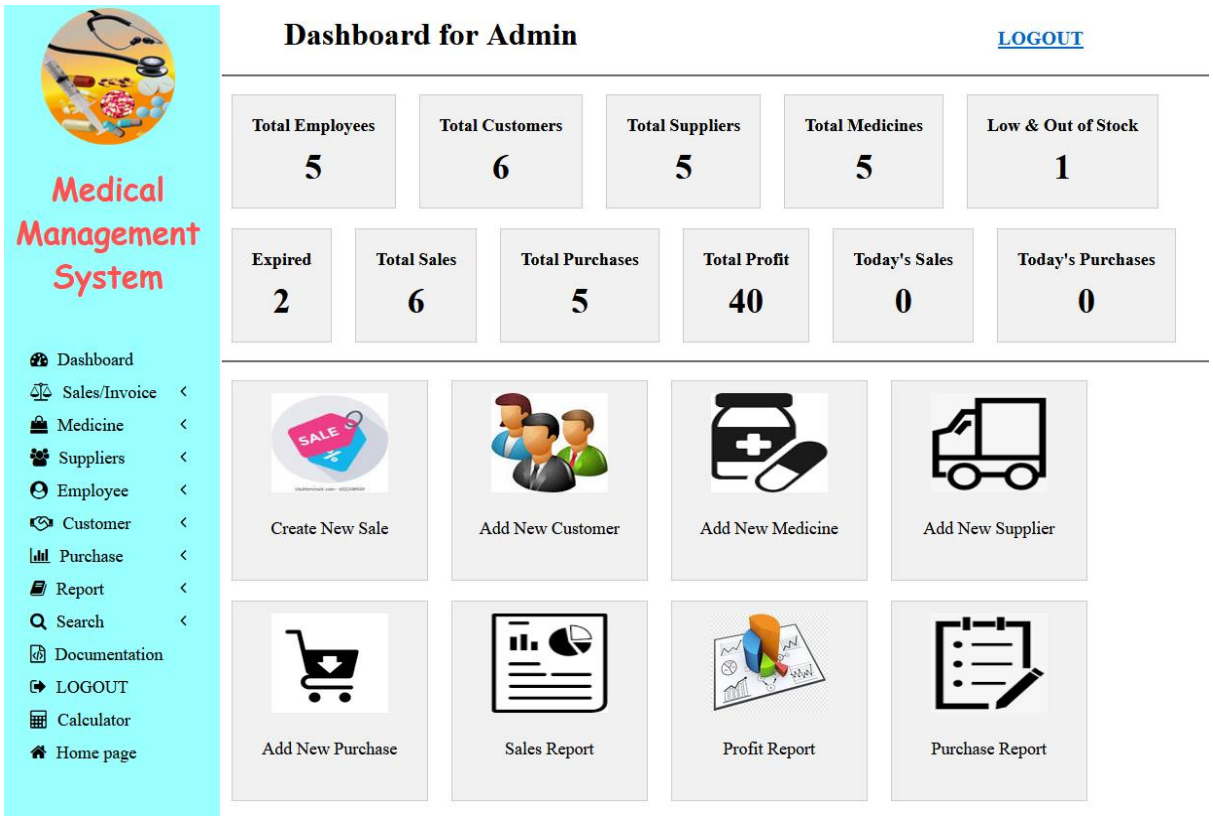

Fig 5.2 Administrator dashboard

The image in the report is a screenshot of the administrator page in a medical shop management project. The page is likely a web-based interface that allows the administrator to manage various aspects of the medical shop, such as inventory, sales, and customer information.

The layout of the page is likely divided into several sections, each with its own specific function. The top of the page may have a navigation bar that allows the administrator to access different sections of the application. There may also be a menu bar on the left side of the page that allows the administrator to access specific features such as adding new products, managing customer information, and viewing sales reports.

The main section of the page is likely devoted to displaying information relevant to the administrator's current task. This may include a list of products in the inventory, customer information, or sales data. The administrator may also be able to perform actions such as adding new products, editing customer information, and generating reports from this section of the page.

There may also be a section of the page dedicated to notifications, alerts or message, which displays important information about the medical shop such as low stock levels, upcoming expiry dates, or customer complaints.

Overall, the administrator page in the medical shop management project is a powerful tool that allows the administrator to effectively manage the various aspects of the medical shop, including inventory, sales, and customer information.

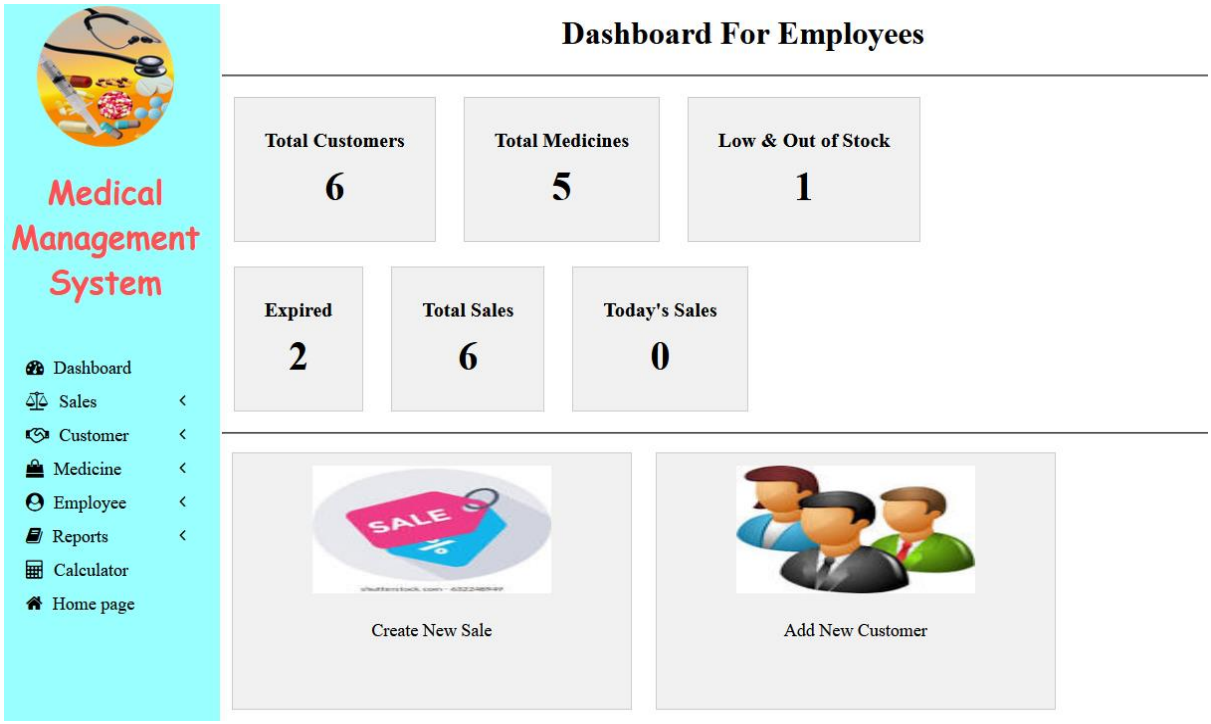

Fig 5.3 Employee dashboard with restricted options

The image in the report is a screenshot of the employee dashboard in a medical shop management project. The employee dashboard is a web-based interface that allows employees to access and manage various aspects of the medical shop, such as inventory, sales, and customer information. The layout of the employee dashboard is divided into several sections, each with its own specific function. The top of the page may have a navigation bar that allows employees to access different sections of the application. There may also be a menu bar on the left side of the page that allows employees to access specific features such as adding new products, managing customer information, and viewing sales reports. The main section of the page is likely devoted to displaying information relevant to the employee's current task.

E_ID	E_Name	E_Role
E1	John Smith	Manager
E2	Alice Williams	Pharmacist
E3	Bob Johnson	Receptionist
E4	Samantha Brown	Pharmacist
E5	James Davis	Receptionist

Fig 5.4 Employee can only view other employee cannot edit their details

Fig 5.5 Administrator can add suppliers details

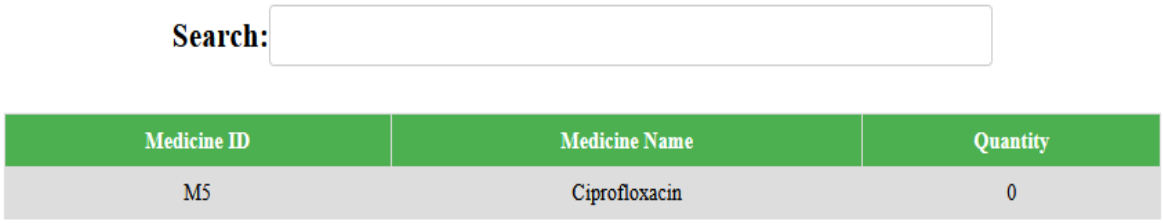

Fig 5.6 Low stock medicines and be seen and search through medicines

Medicine ID	Medicine Name	Expiration Date
M1	Aspirin	2022-04-01
M2	Ibuprofen	2022-02-02

Fig 5.7 Can view soon to be expire medicince & expired medicine

Fig 5.7 illustrates a screenshot of the "View soon to expire and expired medicines" feature of the system. It allows the user to view a list of all the medicines that are soon to expire or have already expired. The screenshot shows the user interface displaying the list of medicines with their details like name, expiry date, quantity available etc. It also provides an option to filter the list based on the expiration status of the medicines. This feature helps the pharmacy to keep track of their inventory and ensure that they do not dispense expired medicines to customers.

Transaction ID	Date	Employee ID	Customer ID	Total Amount
S5	2022-02-05	E5	C5	500
S6	2023-01-12	E2	C6	41

Fig 5.8 Get between dates transactions report

Fig 5.8 represents a screenshot of the "Get between dates transactions report" feature of the system. It allows the user to generate a report of all transactions that occurred within a specified date range. The screenshot shows the user interface for selecting the start and end dates for the report, along with options to filter and export the data.

Chapter 6

Conclusion

The Medical Management System is a comprehensive system designed to meet the needs of a medical store. The system is designed to automate the process of managing customer, employee, medicine, supplier, purchase and sales data. The system is implemented using MySQL as the database management system, and it adheres to the principles of data normalization to ensure data consistency and integrity. The system has been tested and implemented successfully, and it has demonstrated its ability to provide accurate and timely information to the users. The system is user-friendly and easy to navigate, and it has the potential to improve the efficiency and productivity of the medical store. Overall, the Medical Management System is a valuable tool for any medical store looking to improve their operations and streamline their processes.

In addition to its core functionalities, the system also includes features such as triggers, views and stored procedures, which enhance the security and efficiency of the system. The triggers ensure that the data entered into the system is valid and conforms to the set rules and constraints. The views provide a summarized and filtered version of the data, making it easy for users to access the information they need. The stored procedures provide a way to perform complex tasks such as adding a customer or updating employee age, in a simple and efficient manner.

Overall, the Medical Management System is a robust and reliable system that can handle the demands of a medical store. It is designed to be scalable and can be easily extended to meet the evolving needs of the business. The system is a valuable asset for the medical store, and it is expected to improve the overall performance and competitiveness of the business.

REFERENCES

[1] Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.

[2] Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill

[3] Silberschatz Korth and Sudharshan, Database System Concepts, 6th Edition, Mc-Graw Hill, 2013.

[4] Coronel, Morris, and Rob, Database Principles Fundamentals of Design, Implementation and Management, Cengage Learning 2012.

Website links:

[1] MySQL documentation in https://dev.mysql.com/doc/

[2] https://www.w3schools.com/mysql/default.asp

[3] https://www.w3schools.com/html/default.asp

[4] https://www.youtube.com/watch?v=x5XgoTjGTu4

[5] https://www.javatpoint.com/mysql-tutorial