

UNIVERSITY OF VISVESVARAYA COLLEGE OF ENGINEERING

(A State Autonomous Public University on IIT Model)

K.R. Circle, Bengaluru 560001

DBMS Mini Project

Report On

Pharmacy Management System

Submitted By

Poovarasan S

B.Tech ISE 3rd Year, 5th Sem

UUCMS No. U25UV22T064042

Under the guidance of

Dr. Arunalatha J S

Professor

Department of Computer Science and Engineering

University of Visvesvaraya College of Engineering

K.R. Circle, Bengaluru - 560001

March-2025

UNIVERSITY OF VISVESVARAYA COLLEGE OF ENGINEERING

(A State Autonomous Public University on IIT Model)

K.R. Circle, Bengaluru 560001

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that **Poovarasana S**, bearing UUCMS Number **U25UV22T064042**, has successfully completed the Database Management System (DBMS) Mini-Project titled “**Pharmacy Management System**.” This project was carried out in partial fulfillment of the requirements for the Database Management System Laboratory, as part of the 5th semester of the Bachelor of Technology (B.Tech) program in Information Science and Engineering during the academic year **2024–2025**.

Dr. Arunalatha J S

Professor,
Department of CSE,
UVCE.

Examiner 1

Dr. Thriveni J

Professor and Chairperson,
Department of CSE,
UVCE.

Examiner 2

ACKNOWLEDGEMENT

I take this opportunity to express my sincere and heartfelt gratitude to all those who supported and guided me throughout the course of this project.

First and foremost, I would like to extend my deep appreciation to the **University of Visvesvaraya College of Engineering (UVCE), Bangalore**, and to **Prof. Subhasish Tripathy**, Director, UVCE, for providing the necessary infrastructure and a conducive environment to carry out this project. Their vision and commitment to academic excellence have played a pivotal role in shaping my learning journey.

I am profoundly grateful to **Dr. Thriveni J**, Chairperson, Department of Computer Science and Engineering, UVCE, for her continuous support and encouragement throughout my academic tenure. Her guidance has been instrumental in motivating me to pursue excellence.

I would also like to express my special thanks to **Dr. Arunalatha J S**, Professor, Department of Computer Science and Engineering, UVCE, for her invaluable mentorship, insightful suggestions, and constructive feedback during the course of this project. Her expertise and dedication have significantly enriched my understanding and execution of the project.

Furthermore, I am deeply thankful to my parents and well-wishers for their unwavering moral support, encouragement, and belief in my capabilities. Their constant motivation has been the foundation of my perseverance and success.

Lastly, I extend my sincere thanks to everyone who, directly or indirectly, contributed to the successful completion of this project.

Poovarasan S

ABSTRACT

This DBMS mini project, titled **Pharmacy Management System**, is a web-based application aimed at automating and simplifying pharmacy operations. It is built using **MySQL** for backend database management and **EJS** for dynamic frontend rendering. The system handles core functionalities such as inventory control, medicine records, sales tracking, and user authentication.

The application provides separate dashboards for both customers and administrators. Customers can register to receive a unique customer card and access their dashboard to manage addresses, provide feedback, make purchases, complete payments, and download invoices. The admin dashboard allows management of customers, medicines, suppliers, stock, purchases, and payment records.

This project is highly useful for reducing manual errors, improving data consistency, and increasing overall efficiency in a pharmacy setting. It showcases the practical use of database concepts in solving real-world problems and contributes to the digital transformation of healthcare services.

TABLE OF CONTENT

Sl.No.	Chapters	Page No.
1	INTRODUCTION 1.1 Introduction 1.2 Objective 1.3 Roles 1.4 System Views 1.5 Functionality 1.6 Database Management System 1.7 MySQL	01-06
2	LITERATURE REVIEW 2.1 Survey of Existing System 2.2 Developed System 2.3 Software Requirements 2.4 Frontend Technologies 2.5 Backend Technologies 2.6 Entity Relationship Model 2.7 Relational Schema Design 2.8 Normalization 2.9 Advantages of the Developed System	07-15
3	PROPOSED WORK 3.1 Entity-Relationship (ER) Model 3.2 Relational Model 3.3 Normalization	16-24
4	RESULT 4.1 Screenshots 4.1.1 Home Page 4.1.2 Customer Registration Page 4.1.3 Login Page 4.1.3 Contact Page 4.1.4 Customer Dashboard Page 4.1.5 Admin Dashboard Page 4.1.6 Payment Page	25-29
5	CONCLUSION AND FUTURE ENHANCEMENTS	30
6	BIBLIOGRAPHY	31

Chapter-1 : Introduction

1.1 Introduction

The **Pharmacy Management System** is a web-based mini project developed to streamline and automate the core operations of a pharmacy. Built using **EJS** for the frontend and **MySQL** for the backend, it enables efficient management of inventory, sales, purchases, customer records, and suppliers.

The system supports pharmacists in managing medicines, tracking stock levels, generating bills, and handling customer data with ease. Customers can register and access a personalized dashboard to update their profile, add addresses, provide feedback, make purchases, process payments, and download invoices.

An integrated **admin dashboard** allows for centralized control over customers, medicines, suppliers, stock, purchases, and payments. This project not only showcases the use of DBMS concepts but also highlights how dynamic web technologies can address real-world challenges in pharmacy management.

1.2 Objective

The main objective of this project is to develop a **database-driven application** that simplifies and automates pharmacy operations. It aims to:

- Reduce manual tasks through automation of stock, billing, and data management.
- Ensure secure and structured data storage using MySQL.
- Enable customers to interact via a dedicated dashboard for purchases and feedback.
- Provide admin tools for managing all pharmacy-related activities.
- Enhance efficiency and support digital transformation in pharmacy workflows.

This project demonstrates the practical application of **DBMS principles** in creating a scalable, user-friendly solution for the healthcare sector.

1.3 Roles

I. Customer Role

The **Customer** interacts with the system primarily for purchasing medicines and managing their personal profile. Key responsibilities and features include:

- **Registration & Login:** Sign up and log in securely to the system.
- **Customer Card Generation:** A unique customer card is generated upon registration.
- **Profile Management:** Add or update personal details such as name, contact, and address.
- **Dashboard Access:** Use a dedicated dashboard to navigate available features.
- **Medicine Purchase:** Browse available medicines, add to cart, and place orders.
- **Payments:** Make online payments for purchases.
- **Invoice Download:** Download invoices for completed transactions.
- **Feedback:** Provide feedback or reviews on services or medicines.

II. Admin Role

The **Admin** has full control over the system's data and operations. Key responsibilities include:

- **Authentication:** Secure login to access the admin dashboard.
 - **Customer Management:** View, update, or delete customer records.
 - **Medicine Management:** Add, update, or remove medicine details from the system.
 - **Stock Management:** Monitor and manage stock levels to prevent shortages.
 - **Supplier Management:** Maintain supplier records and manage supply entries.
 - **Purchase Records:** Track purchases made by customers.
 - **Payment Tracking:** Oversee and manage payment history and statuses.
 - **Feedback Monitoring:** View customer feedback and address concerns if needed.
-

1.4 System Views

The Pharmacy Management System includes multiple user-friendly and functional views to enhance user experience and simplify navigation. Below are the key pages (views) available in the system:

- **Homepage Dashboard:** Displays an overview including total number of customers, suppliers, medicines, positive customer feedback, top-selling medicines, customer growth analytics, and FAQs for general queries.
 - **Contact Page:** Allows users to get in touch with the pharmacy for inquiries or support.
 - **Error Handling Page:** Displays user-friendly messages for broken links, unauthorized access, or invalid actions.
 - **Customer Registration Page:** Enables new users to register and create an account. A unique customer card is generated upon successful registration.
 - **Login Pages:** Separate login pages for **Customer** and **Admin** to access their respective dashboards securely.
 - **All Medicines Page:** Displays a complete list of available medicines with details such as name, price, stock status, and description.
 - **Admin Dashboard:** Gives the admin full control to manage medicines, customers, suppliers, stock, purchases, payments, and view customer feedback.
 - **Customer Dashboard:** Allows customers to update their profile, manage addresses, give feedback, view purchase history, and download invoices.
 - **Payment Page:** Facilitates secure payments for customer purchases with billing summary.
 - **Reset Password Page:** Allows users who have forgotten their password to reset it securely via email verification or other recovery options.
-

1.5 Functionality of the System

The Pharmacy Management System offers a range of features and functionalities designed to automate, manage, and simplify the daily operations of a pharmacy. Below is a breakdown of its core functionalities:

1. User Authentication

- Secure login system for **admin** and **customers**.
 - **Registration** system for new customers.
-

- **Forgot password** functionality with a secure reset process.

2. Dashboard Overview

- **Homepage dashboard** showing key statistics: number of customers, medicines, suppliers, top-selling medicines, positive feedback, and customer growth.
- Real-time insights and updates for better decision-making.

3. Medicine Management

- Add, update, or delete medicine records (admin only).
- View all available medicines with search and filter options.
- Track medicine stock levels and availability.

4. Customer Management

- Admin can view and manage all customer details.
- Customers can manage their profiles, addresses, and view purchase history.

5. Supplier Management

- Admin can add, update, and delete supplier details.
- Monitor medicine supply sources and track supplier information.

6. Stock and Inventory Control

- Monitor current stock levels.
- Update stock automatically based on purchases.
- Alerts or indications for low-stock medicines (optional).

7. Purchase and Billing

- Customers can add medicines to cart and make purchases.
- Auto-generated **bills/invoices** available for download.
- Purchase records are stored and accessible for both admin and customer.

8. Payments

- Secure **payment processing** page for customer purchases.
- Admin can view and manage all payment transactions.

9. Feedback System

- Customers can submit feedback or reviews.
- Admin can view and manage feedback for service improvement.

10. Error Handling

- Friendly error pages for invalid actions, unauthorized access, or unavailable resources.

11. Contact and Support

- A dedicated **Contact Us** page for customer inquiries and pharmacy support.

1.6 Database Management System

A **Database Management System (DBMS)** is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to **create, retrieve, update and manage data**. The DBMS essentially serves as an interface between the database and end users application programs, ensuring that data is consistently organized and remains easily accessible.

The DBMS manages three important things: the data, the database engine that allows data to be accessed, locked and modified, and the database schema, which defines the database's logical structure. These three foundational elements help to provide concurrency, security, data integrity and uniform administration procedures. Typical database administration tasks supported by the DBMS include change management, performance monitoring/tuning and backup and recovery. Many database management systems are also responsible for automated rollbacks, restarts and recovery as well as the logging and auditing of active.

Characteristics of Database Management System:

- Self-describing nature.
- Keeps a tight control on data redundancy.
- Enforces user defined rules to ensure that integrity of table data
- Provides insulation between Programs and data, Data abstraction.
- Supports multiple views of the data.
- Helps sharing of data and Multi-user transaction processing.

Advantages of DBMS:

- Controlling the redundancy.
- Restricting unauthorized access.
- Providing persistent storage for program objects.
- Providing storage structures for efficient query processing.
- Providing multiple users interfaces
- Representing complex relationships among data.
- Enforcing integrity constraints.

1.5 MySQL

1.5.1 Overview

MySQL is an open-source **Relational Database Management System (RDBMS)** developed and maintained by Oracle Corporation. It uses **Structured Query Language (SQL)** for managing and manipulating data. MySQL is widely used for web-based applications due to its speed, reliability, and ease of integration. It supports cross-platform functionality and is commonly used in applications requiring efficient and secure data handling—making it ideal for systems like Pharmacy Management.

1.5.2 Why MySQL for Pharmacy Management System?

MySQL has been chosen as the backend database for the Pharmacy Management System due to the following reasons:

- **Reliability & ACID Compliance:** Ensures accurate and consistent transaction handling, which is crucial for managing medicine stock, purchases, and payments.
- **Scalability:** Can efficiently handle growing volumes of medicines, customers, suppliers, and sales data.
- **High Performance:** Optimized for fast read/write operations, making it ideal for real-time tasks like stock updates and billing.

- **Security Features:** Supports user privileges, role-based access control, and secure authentication to protect sensitive medical and customer information.
 - **Backup & Replication:** Allows database redundancy, minimizing data loss risks and downtime during failures or crashes.
-

1.5.3 SQL Commands in Pharmacy Management System

Below are examples of commonly used SQL operations within the system:

1. Creating Tables (Defining Structure):

Tables are created to store information about medicines, customers, suppliers, and transactions.

```
CREATE TABLE MEDICINES (  
    MEDICINE_ID INT AUTO_INCREMENT PRIMARY KEY,  
    MEDICINE_NAME VARCHAR(50) NOT NULL,  
    MEDICINE_COMPOSITION VARCHAR(100) NOT NULL,  
    MEDICINE_PRICE DECIMAL(10, 2) NOT NULL,  
    MEDICINE_TYPE ENUM('TABLET', 'SYRUP', 'CAPSULE', 'INJECTION', 'OINTMENT'),  
    MEDICINE_EXPIRY_DATE DATE NOT NULL,  
    MEDICINE_IMG MEDIUMBLOB,  
    CONSTRAINT UNIQUE_MEDICINE UNIQUE (MEDICINE_NAME, MEDICINE_COMPOSITION)  
);
```

2. Inserting Data (Adding Records):

 Used to add new medicine entries to the database.

```
INSERT INTO MEDICINES (MEDICINE_NAME, MEDICINE_COMPOSITION,  
    MEDICINE_PRICE, MEDICINE_EXPIRY_DATE, MEDICINE_TYPE) VALUES  
('PARACETAMOL', 'ACETAMINOPHEN 500MG', 10, '2025-12-31', 'TABLET');
```

3. Retrieving Data (Fetching Records):

 Used to fetch data such as available medicines.

```
SELECT * FROM MEDICINES;
```

4. Updating Data (Modifying Records):

 Used to update stock or medicine pricing.

```
UPDATE MEDICINES SET MEDICINE_PRICE = 50 WHERE MEDICINE_ID = 1;
```

5. Deleting Data (Removing Records):

 Used to delete outdated or expired medicine records.

```
DELETE FROM MEDICINES WHERE MEDICINE_IMG IS NULL;
```

Aggregate Functions Used

1.	COUNT()	Counts the number of medicines or transactions.
2.	SUM()	Calculates the total sales amount.
3.	MAX()	Returns the highest price or latest expiry date.
4.	MIN()	Finds the lowest stock level or minimum price
5.	AVG()	Returns average sales amount or average medicine price.

SQL Constraints Used

To ensure data integrity, the following SQL constraints are implemented:

1. **NOT NULL** – Ensures essential fields like medicine name or price are always filled.
2. **UNIQUE** – Ensures no two medicines or customers share the same unique identifier.
3. **PRIMARY KEY** – Uniquely identifies records, e.g., medicine_id, customer_id.
4. **FOREIGN KEY** – Maintains relationships between tables, e.g., linking purchases to customers.
5. **CHECK** – Validates conditions like positive stock quantity or price.
6. **DEFAULT** – Assigns default values, such as default status to 'Pending' in orders.
7. **INDEX** – Speeds up searches and queries on key columns.
8. **AUTO_INCREMENT** – Automatically generates unique IDs for new records.

Chapter-2 : Literature Review

2.1 Survey of Existing Systems

Traditionally, pharmacy operations have been managed using manual methods such as handwritten records, physical registers, and basic spreadsheet software. While these methods were once considered adequate, they present several limitations in modern pharmacy environments:

- **Speed:** Manual billing, stock updates, and customer handling significantly slow down pharmacy operations.
- **Accuracy:** High chances of human errors in data entry, dosage details, billing, and stock updates can lead to serious consequences.
- **Scalability:** Managing large volumes of medicine stock, supplier records, and customer transactions becomes increasingly difficult.
- **Security:** Physical records are vulnerable to loss, theft, or unauthorized access, risking sensitive customer and medical information.
- **Efficiency:** Redundant and repetitive data entry can lead to inconsistencies and operational delays.
- **Reporting:** Generating real-time reports for sales, stock levels, and supplier activity is time-consuming and often inaccurate.

Despite the growth of digital solutions in healthcare, many small and independent pharmacies still lack an integrated and database-driven system tailored to their daily operations. This leads to fragmented workflows, poor data management, delayed service, and a lack of transparency and traceability in transactions.

The proposed Pharmacy Management System addresses these challenges by offering an automated, centralized, and scalable solution that improves accuracy, security, and operational efficiency in modern pharmacy environments.

2.2 Developed System

In this project, the Pharmacy Management System is developed as a web-based application using **HTML, CSS, and JavaScript** for the front-end interface. The application utilizes **EJS (Embedded JavaScript Templates)** for dynamic page rendering, and follows a **Node.js** and **Express.js** architecture for backend development. The backend logic is structured using a modular approach with **routes and controllers** for better code organization, maintainability, and scalability. The system uses **MySQL** as the backend relational database to store and manage structured data securely.

The developed system efficiently handles core pharmacy functions by centralizing operations such as medicine inventory management, sales and purchase transactions, user authentication, and administrative controls. It supports **role-based access control**, where customers and administrators can access features relevant to their roles—for instance, customers can manage profiles, make purchases, and provide feedback, while admins manage stock, suppliers, customers, and payments.

By replacing traditional manual processes with an integrated, real-time digital system, the Pharmacy Management System significantly improves **data accuracy, operational efficiency, and security**, while providing a user-friendly and responsive interface for both customers and administrators.

2.3 Software Requirements

- **Operating System:** Windows / Linux / macOS
 - **Frontend Technologies:** HTML, CSS, JavaScript, EJS (Embedded JavaScript Templates)
 - **Backend Technologies:** Node.js with Express.js (JavaScript)
 - **Database Management System:** MySQL
 - **Templating Engine:** EJS for rendering dynamic content
 - **Web Server:** Express.js development server (Node-based)
 - **Development Tools:** Visual Studio Code (or any preferred code editor), MySQL Workbench, Node Package Manager (npm)
-

2.4 Frontend Technologies

The frontend serves as the primary user interface for interacting with the Pharmacy Management System. It is designed to provide a responsive, user-friendly, and visually appealing experience across all devices. The system's frontend is built using a combination of modern web technologies that ensure efficient user interaction and seamless data flow to the backend.

1. EJS (Embedded JavaScript Templates):

EJS is used as the templating engine to dynamically generate HTML content on the server-side. It allows embedding JavaScript code directly within HTML pages, enabling real-time data rendering, conditional content display, and modular template reuse for components like headers, footers, and navigation bars.

2. Tailwind CSS:

Tailwind CSS is a utility-first CSS framework used to style the application. It enables rapid UI development with pre-defined classes for layout, spacing, typography, and responsive design, resulting in a clean and consistent visual interface across all pages.

3. JavaScript:

JavaScript adds interactivity to the web pages, such as input validation, toggling views, handling dynamic UI elements, and managing user interactions based on roles (e.g., admin vs. customer). It also supports real-time form validation before data submission.

4. Forms for Data Entry:

HTML forms are used throughout the system to collect user inputs—such as customer registration, login, medicine details, purchases, and feedback. These inputs are validated and submitted to the backend via routes, where they are processed and stored in the MySQL database.

2.5 Backend Technologies

The backend of the Pharmacy Management System handles the core functionality of the application, including processing user requests, managing authentication, interacting with the database, and delivering responses to the frontend. It ensures secure, reliable, and efficient execution of all pharmacy-related operations.

1. Node.js with Express Framework

Node.js is a fast, event-driven JavaScript runtime built on Chrome's V8 engine, used for executing JavaScript on the server-side. Express.js is a lightweight, unopinionated web framework built on top of Node.js that simplifies backend development using a modular and scalable structure. It supports the MVC (Model-View-Controller) architecture, routing, and middleware functionalities.

In this project, Node.js with Express is used to:

- Handle API requests and responses
- Manage routes and controllers for better code organization
- Integrate with MySQL for performing CRUD operations on data
- Handle authentication, session management, and role-based access for admin and customer users
- Provide secure form handling and input validation

2. MySQL (Relational Database Management System)

MySQL is an open-source RDBMS used to store, retrieve, and manage structured data such as medicine inventory, customer details, supplier records, transactions, and feedback. It ensures data consistency, integrity, and supports complex queries for fast and accurate data retrieval.

In this Pharmacy Management System, MySQL is utilized for:

- Creating and managing normalized database schemas with relationships using primary and foreign keys
- Ensuring referential integrity and transactional safety
- Executing stored procedures for encapsulating complex operations (e.g., generating invoices or purchase summaries)
- Implementing triggers to automate tasks like updating stock levels when a purchase is made
- Supporting concurrent operations while maintaining high performance and data accuracy

Together, Node.js and MySQL form a robust backend that supports the seamless functioning of the pharmacy system with scalability, maintainability, and data security.

2.6 Entity-Relationship (ER) Model

The Entity-Relationship (ER) model is a high-level conceptual data model that serves as a blueprint for designing a structured and efficient relational database. It defines the key data elements, their attributes, and the relationships between them. In the context of a Pharmacy Management System, the ER model is crucial for organizing and understanding how different components of the system interact with each other.

Purpose of the ER Model

The main goals of the ER model in this system are to:

- Visually represent the data structure in a way that is easy to understand for both developers and stakeholders.
- Identify and define the key entities involved in pharmacy operations.

- Establish the relationships between these entities (e.g., between customers and purchases, payment and invoice).
- Define attributes for each entity and relationship.
- Enforce integrity constraints such as primary keys, foreign keys, and cardinalities.

Core Components of the ER Model

1. **Entities:** Entities are real-world objects or concepts that store data in the system.

In the Pharmacy Management System, major entities include:

- **Admin:** Represents system administrators who manage the application.
- **Customers:** Represents registered users who can place orders and make payments.
- **Medicines:** Represents the products available for purchase.
- **Suppliers:** Represents vendors or suppliers who provide stock of medicines.
- **Stocks:** Represents current stock quantity of each medicine from each supplier.
- **Purchase:** Represents medicine purchases made by customers.
- **Purchase Sessions:** Groups together purchases made by a customer at the same time.
- **Payment:** Represents payment transactions made by customers.
- **Invoice:** Represents billing details for a purchase session.
- **Strong Entity:** Has a primary key (e.g., Medicine, Customer).
- **Weak Entity:** Depends on another entity for identification (e.g., Stocks depend on Suppliers and Medicines)

2. **Attributes:** Attributes are the characteristics used to describe entities within the database.

Below are examples and types based on my schema:

- **Customers Attributes:**
customer_id, customer_name, customer_email, customer_ph_no, customer_password, customer_balance_amt, customer_created_at, customer_photo.
- **Address (Separate entity):**
Stored in customer_addresses, with attributes like address_type, street, city, state, zip_code.
- **Medicines Attributes:**
medicine_id, medicine_name, medicine_composition, medicine_price, medicine_type, medicine_expiry_date, medicine_img.
- **Suppliers Attributes:**
supplier_id, supplier_name, supplier_email, supplier_ph_no.
- **Address:**
Managed in a separate entity supplier_addresses.
- **Purchase Attributes:**
purchase_id, purchase_time, purchased_quantity, total_amt, customer_id, medicine_id, supplier_id.

Types of Attributes:

- **Simple (Atomic):**
medicine_id, medicine_price, customer_email, supplier_ph_no — indivisible values.
- **Composite (Logically):**
customer_name or supplier_name could be treated as composite (first and last name), although stored as a single field here.

- **Derived:**
 - **net_total** in the invoice table is a **derived attribute**, calculated from **total_amt_to_pay - discount**.
 - **total_amt** in purchases is also derived using a stored procedure based on price × quantity.
- **Multivalued:**
 - **Customer Addresses:** A customer may have multiple addresses (customer_addresses table).
 - **Feedbacks:** A customer may give multiple feedback entries.

3. Relationships Between Entities: Relationships define how entities are interact with each other

- **Customer and Purchase**
 - **Type:** One-to-Many (1:N)
One customer can make many purchases.
- **Purchase and Medicine**
 - **Type:** Many-to-One (N:1)
Each purchase references a single medicine. However, over multiple purchases, a medicine can be referenced multiple times → Many-to-Many in the larger context of sessions or orders.
- **Supplier and Medicines**
 - **Type:** Many-to-Many (M:N)
Implemented via the stocks table — a supplier can supply multiple medicines, and a medicine can be supplied by multiple suppliers.
- **Customer and Address**
 - **Type:** One-to-Many (1:N)
One customer can have multiple addresses.
- **Customer and Feedback**
 - **Type:** One-to-Many (1:N)
One customer can provide multiple feedbacks.
- **Customer and Payment**
 - **Type:** One-to-Many (1:N)
A customer can make multiple payments.
- **Purchase and Purchase Session**
 - **Type:** Many-to-One (N:1)
Multiple purchases can belong to one session.
- **Purchase Session and Invoice**
 - **Type:** One-to-One (1:1)
Each session can have one corresponding invoice.

- **Invoice and Admin**

- **Type:** Many-to-One (N:1)

Invoices can be processed by an admin.

Constraints in the Schema

- **Primary Keys:** Uniquely identify each row (e.g., customer_id, medicine_id, purchase_id, payment_id).
- **Foreign Keys:** Maintain referential integrity across entities:
 - customer_id in purchases, feedbacks, customer_addresses, payments, etc.
 - medicine_id and supplier_id in stocks, purchases.
- **Unique Constraints:** Ensure values like customer_email, customer_ph_no, supplier_ph_no, and the composite key (medicine_name, medicine_composition) are not duplicated.

Benefits of Using ER Model in Pharmacy Systems

- Provides a clear and structured visualization of system data.
- Helps in planning efficient database schemas with normalized tables.
- Ensures data consistency, accuracy, and integrity.
- Simplifies communication between developers and stakeholders.
- Acts as a reliable reference during database development, testing, and maintenance.

2.7 Relational Schema Design

- The relational schema represents the logical structure of the Pharmacy Management System database, designed using relational database principles. This schema ensures data consistency, integrity, and normalization. Each table corresponds to a real-world entity or relationship from the domain and is constructed with appropriate primary keys, foreign keys, and constraints.
- Below is the 7-step ER-to-Relational Schema conversion applied to our project:

Step 1: Mapping Regular (Strong) Entity

- Sets each strong entity in the ER model becomes a separate relation (table).
 - All simple and composite attributes are included as columns.
 - A primary key is chosen to uniquely identify each tuple.
 - Composite attributes are flattened into individual attributes.
 - Example:
 - Entity: medicines (medicine_id, medicine_name , medicine_composition, medicine_price, medicine_type , medicine_expiry_date, medicine_img)
 - Relational Schema: medicines (medicine_id PRIMARY KEY, medicine_name , medicine_composition, medicine_price, medicine_type , medicine_expiry_date, medicine_img)

Step 2: Mapping Weak Entity Sets

- Weak entities do not have a primary key of their own and depend on a strong entity for identification.
 - Create a relation for the weak entity.
 - Include the partial key and foreign key referencing the strong entity's primary key.
 - The combination of the foreign key and partial key becomes the composite primary key.
 - Example:
 - Entity: stocks dependent on medicines and suppliers
 - Relational Schema: stocks (medicine_id, supplier_id, stock_quantity, PRIMARY KEY (medicine_id, supplier_id), FOREIGN KEY (medicine_id) REFERENCES medicines(medicine_id), FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id))

Step 3: Mapping Binary 1:1 Relationships

- For one-to-one relationships, the foreign key can be placed in either of the participating entities' tables, preferably the one with total participation.
 - Ensure unique constraint on the foreign key.
 - If participation is total on one side, foreign key goes to that side.
 - Example: Purchase Session and Invoice where each session can have one corresponding invoice

Step 4: Mapping Binary 1:N Relationships

- For one-to-many relationships, the foreign key is placed in the relation on the 'many' side of the relationship.
 - The foreign key references the primary key of the 'one' side.
 - Ensures referential integrity.
 - Example: One **Customer** can give multiple **Feedbacks**.

Step 5: Mapping Binary M:N Relationships

- Many-to-many relationships require a separate relation.
 - Create a new relation to represent the relationship.
 - Include foreign keys referencing the primary keys of both participating entities.
 - The combination of both foreign keys becomes the primary key of the new relation.
 - Include any attributes of the relationship as additional columns.
 - Example: Supplier and Medicines (M:N) Implemented via the stocks table — a supplier can supply multiple medicines, and a medicine can be supplied by multiple suppliers.

Step 6: Mapping Multivalued Attributes

- Multivalued attributes are represented by creating a separate relation.
 - The new relation includes the multivalued attribute and the primary key of the entity it belongs to.
 - The schema separates address-related data into their own tables:
 - customer_addresses for customers and supplier_addresses for suppliers
- Each allows multiple address entries linked to a single customer or supplier using foreign keys.

Step 7: Mapping Ternary (and Higher) Relationships

- For relationships involving three or more entities, create a new relation.
 - Include the primary keys of all participating entities as foreign keys.
 - These foreign keys together form the composite primary key.
 - Add any attributes of the relationship to the relation.

This 7-step method ensures a systematic and consistent transformation of the ER model into a fully functional relational schema. It preserves: Data integrity through keys and constraints, logical consistency by maintaining relationships, and efficiency through normalization and redundancy elimination. The resulting relational schema becomes the foundation for physical database implementation in systems such as MySQL or PostgreSQL

2.8 Normalization

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update, and Deletion Anomalies. Normalization rules divide larger tables into smaller tables and link them using relationships. Normalization of data can be looked upon as a process of analyzing the given relation schemas based on their Functional Dependencies and primary keys to achieve the desirable properties of:

- Minimizing redundancy.
- Minimizing the insertion, deletion, and update anomalies.

Functional Dependency: The Functional Dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that for any two tuples t1 and t2 in r that have $t1[X] = t2[X]$, the values of the Y component of a tuple in r depend on or are determined by the values of the X components. Alternatively, the values of the X component of a tuple uniquely determine the values of the Y component.

Here is a list of Normal Forms in SQL:

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)

a) First Normal Form (1NF): It states that the domain of an attribute must include only atomic values and that the value of any attribute in a tuple must be a single value from the domain of the attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.

b) Second Normal Form (2NF): This normal form is based on full functional dependency. A functional dependency $X \rightarrow Y$ is fully functional if the removal of any attribute A from X means that the dependency does not hold anymore. A relation schema R is in 2NF if every non-prime attribute A in R is fully functionally dependent on the primary key of R.

c) Third Normal Form (3NF): Third Normal Form (3NF) is based on the concept of transitive dependency. A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there is a set of attributes Z that is neither a candidate key nor a subset of any key of R , and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. A relation schema R is in 3NF if it satisfies 2NF and no non-prime attribute of R is transitively dependent on the primary key.

d) Boyce-Codd Normal Form (BCNF): A relation schema R is in BCNF if, for every one of its functional dependencies $X \rightarrow Y$, X is a super key. BCNF is a stricter version of 3NF where there should not be any partial or transitive dependencies.

e) Fourth Normal Form (4NF): A relation is in 4NF if it is in BCNF and does not have multi-valued dependencies. Multi-valued dependencies occur when one attribute in a table uniquely determines another attribute, independent of other attributes.

f) Fifth Normal Form (5NF): A relation is in 5NF if it is in 4NF and does not have join dependencies that are not implied by the candidate keys. This form deals with eliminating redundancy caused by overcomplicated relationships within the data.

2.9 Advantages of the Developed System

The developed **Pharmacy Management System** provides numerous advantages over traditional, manual record-keeping and inventory management methods. By integrating modern relational database concepts and implementing a structured MySQL-based backend, the system ensures higher efficiency, security, and ease of use. Below are the key benefits of the developed system:

- **Automation of Core Operations:**
The system automates critical pharmacy functions such as **medicine stock management, invoice generation, supplier tracking, and purchase recording**, significantly reducing manual workload and the risk of human error.
- **Improved Accuracy and Efficiency:**
By relying on a relational database, the system enables **fast and accurate retrieval of customer data, purchase history, medicine availability, and billing details**, leading to more efficient operations and better decision-making.
- **Enhanced Data Security:**
Role-based access is enforced through admin and user privileges, ensuring that **only authorized personnel** can view or modify sensitive data like supplier details, invoices, or stock records. This safeguards data integrity and confidentiality.
- **Real-Time Data Access and Updates:**
Inventory levels, purchase transactions, and invoice records are **updated in real-time**, allowing the pharmacy to maintain up-to-date records and make timely restocking decisions.
- **Centralized Database Management:**
All data—including customer records, medicine details, supplier information, and transaction logs—is stored in a **centralized MySQL database**, which promotes **data consistency, reduces redundancy**, and simplifies data maintenance.

Chapter-3 : Proposed Work

This chapter outlines the design and structure of the proposed Pharmacy Management System, detailing how the various entities interact, how data is modeled, and how normalization principles are applied. The design focuses on efficiently organizing and managing essential pharmacy operations such as customer registration, medicine inventory management, purchase tracking, and invoice generation. The Entity-Relationship Diagram (ERD) visually represents the database structure, illustrating entities, their attributes, and interrelationships. The ER model includes core entities such as Admin, Customer, Medicine, Supplier, Purchase, Invoice, Stock and Payment. Relationships are established through the use of foreign keys and associative (junction) tables, especially for handling many-to-many relationships. Fig 3.1.1 shows the ER Diagram of the Pharmacy Management System, representing the entities, their attributes, and the relationships among them.

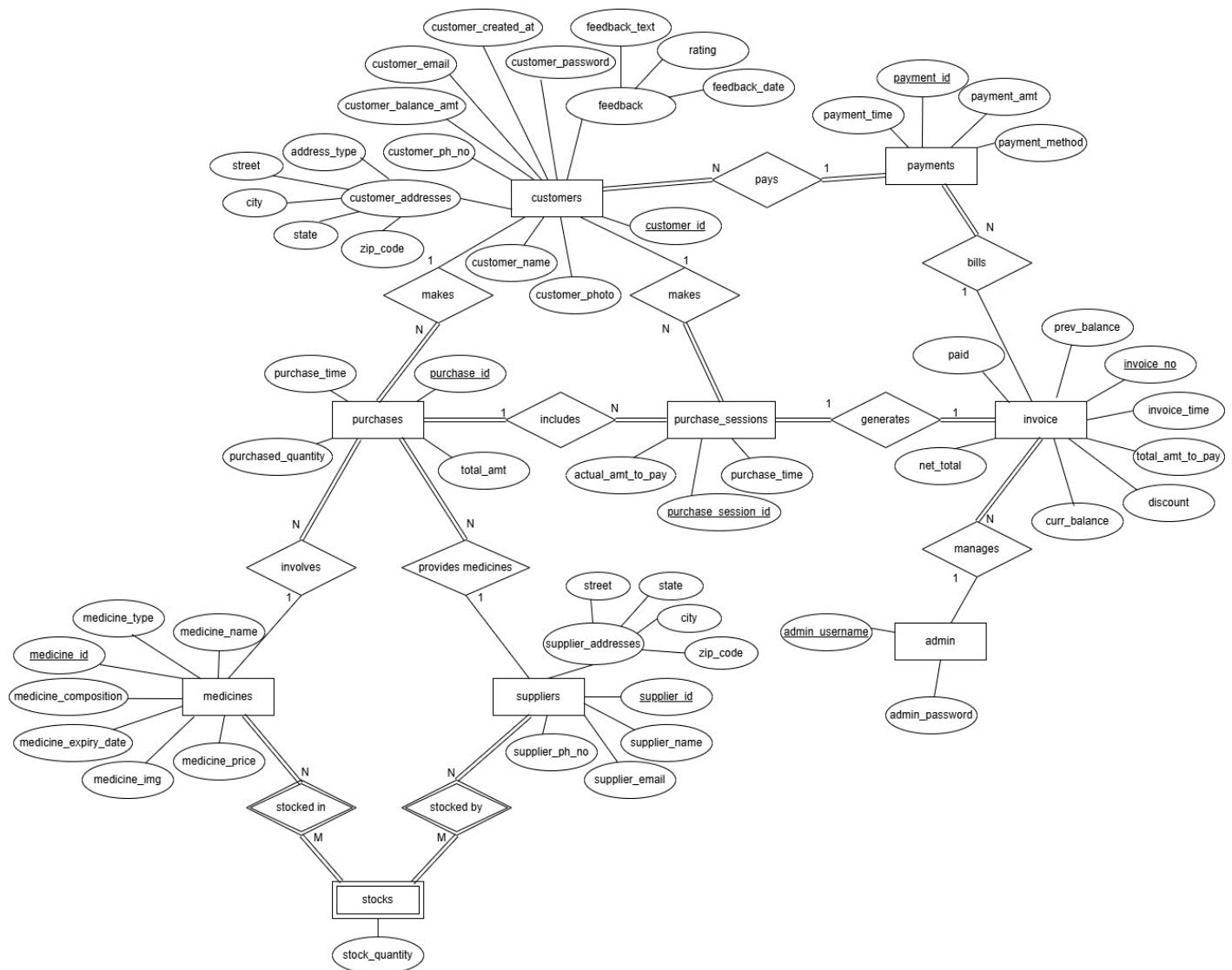


Figure 3.1.1 Entity Relationship Diagram of Pharmacy Management System

Tables and Relationships – Pharmacy Management System

1. Admin

- **Entities:** Stores login credentials for administrators.
- **Fields:** admin_username (PK), admin_password
- **Relationships:**
 1. One-to-many with Invoice (admin_username)

2. Customers

- **Entities:** Stores registered customer information.
- **Fields:** customer_id (PK), customer_created_at, customer_name, customer_email, customer_ph_no, customer_photo, customer_password, customer_balance_amt
- **Relationships:**
 1. One-to-many with Customer Addresses
 2. One-to-many with Feedbacks
 3. One-to-many with Purchases
 4. One-to-many with Purchase Sessions
 5. One-to-many with Payments

3. Customer Addresses

- **Entities:** Stores address information of customers.
- **Fields:** customer_address_id (PK), address_type, customer_id, street, city, state, zip_code
- **Relationships:**
 1. Many-to-one with Customers (customer_id)

4. Feedbacks

- **Entities:** Stores customer reviews and ratings.
- **Fields:** feedback_id (PK), customer_id, rating, feedback_text, feedback_date
- **Relationships:**
 1. Many-to-one with Customers (customer_id)

5. Medicines

- **Entities:** Contains information about medicines available for sale.
- **Fields:** medicine_id (PK), medicine_name, medicine_composition, medicine_price, medicine_type, medicine_expiry_date, medicine_img
- **Relationships:**
 1. One-to-many with Stocks
 2. One-to-many with Purchases

6. Suppliers

- **Entities:** Holds supplier data.
- **Fields:** supplier_id (PK), supplier_name, supplier_email, supplier_ph_no

- **Relationships:**
 1. One-to-many with Supplier Addresses
 2. One-to-many with Stocks
 3. One-to-many with Purchases

7. Supplier Addresses

- **Entities:** Contains supplier location information.
- **Fields:** supplier_address_id (PK), supplier_id, street, city, state, zip_code
- **Relationships:**
 1. Many-to-one with Suppliers (supplier_id)

8. Stocks

- **Entities:** Represents medicine stock provided by suppliers.
- **Fields:** medicine_id, supplier_id, stock_quantity
- **Relationships:**
 1. Many-to-one with Medicines (medicine_id)
 2. Many-to-one with Suppliers (supplier_id)

9. Purchases

- **Entities:** Records individual medicine purchase transactions.
- **Fields:** purchase_id (PK), purchase_time, customer_id, medicine_id, supplier_id, purchased_quantity, total_amt
- **Relationships:**
 1. Many-to-one with Customers (customer_id)
 2. Many-to-one with Medicines (medicine_id)
 3. Many-to-one with Suppliers (supplier_id)

10. Purchase Sessions

- **Entities:** Represents grouped purchases made by a customer during a session.
- **Fields:** purchase_session_id (PK), customer_id, purchase_time, actual_amt_to_pay
- **Relationships:**
 1. Many-to-one with Customers (customer_id)
 2. Many-to-one with purchases(purchase_sessions)
 3. One-to-one with Invoice

11. Payments

- **Entities:** Logs payment transactions by customers.
- **Fields:** payment_id (PK), customer_id, payment_amt, payment_time, payment_method
- **Relationships:**
 1. Many-to-one with Customers (customer_id)
 2. One-to-one with Invoice

12. Invoice

- **Entities:** Final billing information including payment and discount details.
- **Fields:** invoice_no (PK), invoice_time, purchase_session_id, admin_username, discount, payment_id, prev_balance, total_amt_to_pay, net_total, curr_balance
- **Relationships:**
 1. Many-to-one with Purchase Sessions (purchase_session_id)
 2. Many-to-one with Admin (admin_username)
 3. Many-to-one with Payments (payment_id)

3.2 Relational Model

The relational model defines the structure of the database in terms of tables, their attributes, and the relationships between them. The **Pharmacy Management System** is developed using relational tables to efficiently store, retrieve, and manage various types of pharmacy-related data, such as customer details, medicines, purchases, suppliers, stocks, payments, and invoices. These tables are properly normalized and are interconnected using primary keys and foreign keys to ensure **data integrity, consistency**, and to minimize **data redundancy**. This structure provides a solid foundation for performing essential operations such as medicine purchasing, stock tracking, payment processing, and invoice generation in an efficient and organized manner.

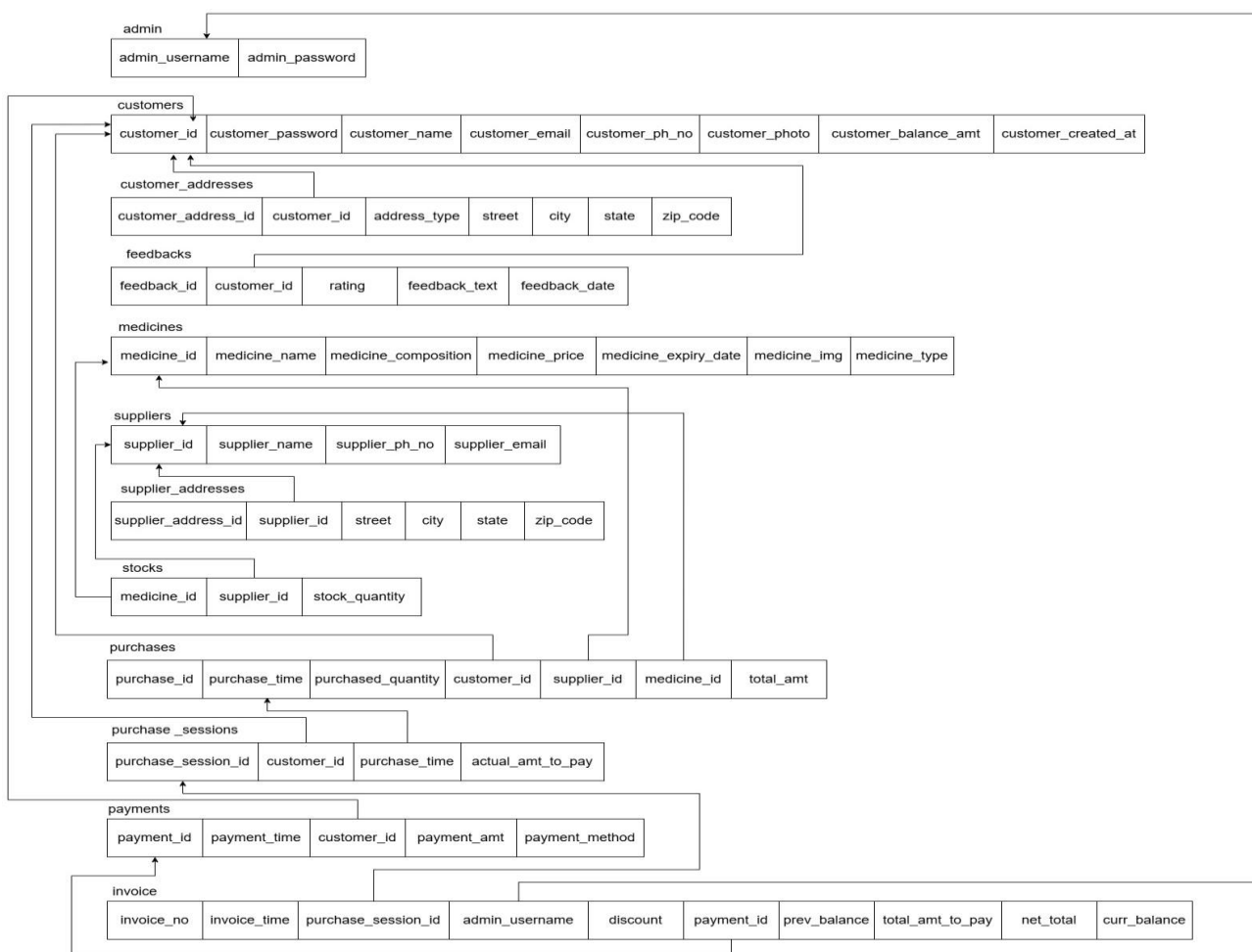


Figure 3.2.1 Relational Schema Diagram of Pharmacy Management System

3.3 Relationships and Cardinalities

This section describes the relationships between the various entities (tables) in the Pharmacy Management System, along with their cardinalities and participation. These relationships ensure proper linkage and interaction between data points such as customers, medicines, purchases, suppliers, stocks, and payments.

1. **Customers (1) to Customer_Addresses (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** One customer can have multiple addresses (e.g., home, work), but each address is associated with exactly one customer.

2. **Customers (1) to Feedbacks (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** A customer can give multiple feedbacks, but each feedback is given by one customer.

3. **Customers (1) to Purchase_Sessions (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** A customer can initiate multiple purchase sessions, but each session belongs to exactly one customer.

4. **Customers (1) to Purchases (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** A customer can make many purchases, but each purchase is made by one customer.

5. **Medicines (1) to Purchases (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** One medicine can be purchased multiple times, but each purchase is related to one specific medicine.

6. **Medicines (1) to Stocks (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** A single medicine can be stocked by multiple suppliers, but each stock entry refers to one specific medicine.

7. **Suppliers (1) to Stocks (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** One supplier can supply multiple types of medicines, but each stock entry refers to one supplier.

8. **Suppliers (1) to Supplier_Addresses (N)**

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** A supplier can have multiple addresses, but each address belongs to one supplier.

9. Suppliers (1) to Purchases (N)

- **Relationship:** One-to-Many (nullable)
- **Cardinality:** 0..1:N
- **Description:** A supplier may be involved in many purchases, but a purchase may also occur without a supplier (e.g., direct sale or stock already available).

10. Purchase_Sessions (1) to Purchases (N) (*via trigger or reference*)

- **Relationship:** One-to-Many (logical via timestamp or triggers)
- **Cardinality:** 1:N
- **Description:** One purchase session can include multiple purchases, although this relationship is not enforced directly in schema but handled logically or via triggers.

11. Customers (1) to Payments (N)

- **Relationship:** One-to-Many
- **Cardinality:** 1:N
- **Description:** A customer can make many payments, but each payment is done by a single customer.

12. Purchase_Sessions (1) to Invoice (1)

- **Relationship:** One-to-One
- **Cardinality:** 1:1
- **Description:** Each purchase session results in exactly one invoice, and each invoice is linked to one specific session.

13. Payments (1) to Invoice (1)

- **Relationship:** One-to-One
- **Cardinality:** 1:1
- **Description:** Each payment is used in one invoice, and each invoice is associated with one payment.

14. Admin (1) to Invoice (N)

- **Relationship:** One-to-Many (nullable participation)
- **Cardinality:** 0..1:N
- **Description:** An admin may generate many invoices, but each invoice may or may not be associated with an admin.

3.3 NORMALIZATION

Normalization is a vital database design technique used to minimize data redundancy and avoid undesirable anomalies like Insertion, Update, and Deletion issues. It involves decomposing large, complex tables into smaller, well-structured ones, linked through relationships, ensuring data consistency and integrity.

In the **Pharmacy Management System**, normalization was applied to organize data efficiently across multiple related tables (e.g., medicines, suppliers, purchases, customers, stocks, etc.), driven by **Functional Dependencies** and the identification of **primary keys**.

Objectives of Normalization

1. To reduce redundancy in the database.
2. To eliminate insertion, deletion, and update anomalies.
3. To ensure each table has a single, clear purpose.

Functional Dependency

A **functional dependency** (denoted as $X \rightarrow Y$) implies that the value of attribute Y is determined by the value of attribute X. In our case, for example:

- $\text{medicine_id} \rightarrow \text{medicine_name, category, price}$
- $\text{customer_id} \rightarrow \text{customer_name, phone, email}$ This ensures that a unique identifier (like a primary key) defines the associated attribute values.

Database Normal Forms in Our System

a) First Normal Form (1NF)

A table is in 1NF if:

- All attributes contain **atomic** (indivisible) values.
- Each column contains values of a single type.
- There are **no repeating groups or arrays**.
- Each row is **uniquely identified** by a primary key.

Compliance in Our System:

- All tables (e.g., medicines, customers, stocks, suppliers) have clearly defined **primary keys**.
- No multivalued fields or repeating groups.
- All fields store single, atomic values (e.g., medicine_name, customer_phone).

b) Second Normal Form (2NF)

A table is in 2NF if:

- It is already in **1NF**.
- Every **non-prime attribute** is fully functionally dependent on the **entire primary key**.

Compliance in Our System:

- Tables with **composite keys** (like stocks with medicine_id + supplier_id) store only attributes **fully dependent on both keys**.
- Data like stock_quantity or stock_date depend on both medicine_id and supplier_id, not partially.
- Tables like purchases and purchase_sessions have been separated to avoid partial dependencies.

c) Third Normal Form (3NF)

A table is in 3NF if:

- It is in **2NF**.
- There are **no transitive dependencies**, i.e., non-key attributes depend **only** on the primary key, not on other non-key attributes.

Compliance in Our System:

- In customers, attributes like email or phone depend only on customer_id, not on customer_name.
- supplier_address details are stored in a separate supplier_addresses table instead of duplicating in suppliers.
- Similarly, customer_address is maintained in a separate table to avoid transitive redundancy.

d) Boyce-Codd Normal Form (BCNF)

A table is in BCNF if:

- It is in **3NF**.
- For every functional dependency $X \rightarrow Y$, **X is a super key**.

Compliance in Our System:

- All tables follow strict dependency rules where **every determinant is a candidate key**.
- Example: In stocks, the combination (medicine_id, supplier_id) is a super key determining stock-related data.
- No overlapping keys or ambiguous dependencies exist.

e) Fourth Normal Form (4NF)

A table is in 4NF if:

- It is in **BCNF**.
- It contains **no multi-valued dependencies**.

Compliance in Our System:

- There are no multi-valued attributes. For instance, if a supplier supplies multiple medicines, this is recorded **once per medicine** in the stocks table, not as a list.

f) Fifth Normal Form (5NF)

A table is in 5NF if:

- It is in **4NF**.
- All **join dependencies** are preserved through decomposition without causing **loss of data**.

Compliance in Our System:

- Tables are properly decomposed (e.g., invoices, payments, purchase_sessions) to ensure **lossless joins**.
- Ensures rejoining related tables does **not result in spurious tuples** or information loss.

3.4 TRIGGERS

A **trigger** is a special database object that automatically executes a predefined set of actions when a particular event (like **INSERT**, **UPDATE**, or **DELETE**) occurs on a table. Triggers help enforce business rules, maintain data integrity, and automate essential processes.

Types of Triggers Used:

1. **BEFORE Trigger** – Executes before the triggering event occurs.
2. **AFTER Trigger** – Executes after the event occurs.

Triggers Implemented in Pharmacy Management System:

1. check_expiry_before_insert

- **Trigger Type:** BEFORE INSERT on medicines table
- **Purpose:**
Prevents insertion of medicine records if the expiry date is earlier than the current date.
- **When It Executes:**
 - On INSERT operations
 - Before inserting a medicine
 - Aborts the operation if the medicine is already expired
- **Business Logic:**
Ensures expired medicines are not stored in the system.

2. calculate_total_amt_before_insert / calculate_total_amt_before_update

- **Trigger Type:**
 - BEFORE INSERT and BEFORE UPDATE on purchases table
- **Purpose:**
Automatically calculates the total amount (total_amt) based on the medicine price and quantity.

3. reduce_stock_on_purchase_before_insert / reduce_stock_on_purchase_before_update

- **Trigger Type:**
 - BEFORE INSERT and BEFORE UPDATE on purchases table
- **Purpose:**

Updates the stock quantity by reducing the stock for the purchased medicine from the corresponding supplier.
- **When It Executes:**
 - Before a new purchase is inserted
 - When an existing purchase is updated
- **Business Logic:**

Maintains accurate stock levels in real time.

4. restore_stock_on_purchase_delete

- **Trigger Type:** AFTER DELETE on purchases table
- **Purpose:**

Automatically restores the stock quantity when a purchase is deleted.
- **When It Executes:**
 - After a purchase record is deleted
- **Business Logic:**

Prevents accidental loss of stock data due to deletions.

5. create_purchase_session / update_purchase_session

- **Trigger Type:**
 - AFTER INSERT and AFTER UPDATE on purchases table
- **Purpose:**

Assigns purchases to a session and updates the actual amount to be paid for the session.
- **When It Executes:**
 - After a purchase is added or updated
- **Business Logic:**

Groups multiple purchases under a session and keeps the amount accurate.

6. calculate_total_amt_to_pay_before_insert

- **Trigger Type:** BEFORE INSERT on invoice table
- **Purpose:**

Calculates the total amount to pay by combining purchase amount with any pending customer balance.
- **When It Executes:**
 - Before inserting a new invoice
- **Business Logic:**

Ensures the total amount reflects any outstanding dues of the customer.

7. update_customer_balance_after_invoice_insert

- **Trigger Type:** BEFORE INSERT on invoice table
- **Purpose:**

Updates the customer's balance based on payment amount and total invoice amount.
- **When It Executes:**
 - Before inserting an invoice
- **Business Logic:**

Keeps the customer's account balance up-to-date for accurate financial tracking.

Chapter-4 : Result

The **Pharmacy Management System (PMS)** was successfully implemented and thoroughly tested using sample data including medicines, suppliers, customers, purchases, and invoices. The system efficiently handles core pharmacy operations such as medicine stock management, purchase tracking, customer billing, payment processing, and balance management, while maintaining data consistency and integrity.

Test users found the system interface intuitive and easy to navigate, providing smooth access to modules like **medicine inventory, purchase entry, supplier tracking, invoice generation, and customer payment history**. Real-time updates on **medicine availability, stock levels, and purchase sessions** improved the overall user experience for both pharmacy staff and administrators.

The system generates detailed reports on **current stock levels, purchase transactions, customer balances, invoice summaries, and supplier-wise inventory**, offering valuable insights for pharmacy managers. These analytics assist in **inventory forecasting, supplier evaluation, and financial decision-making**.

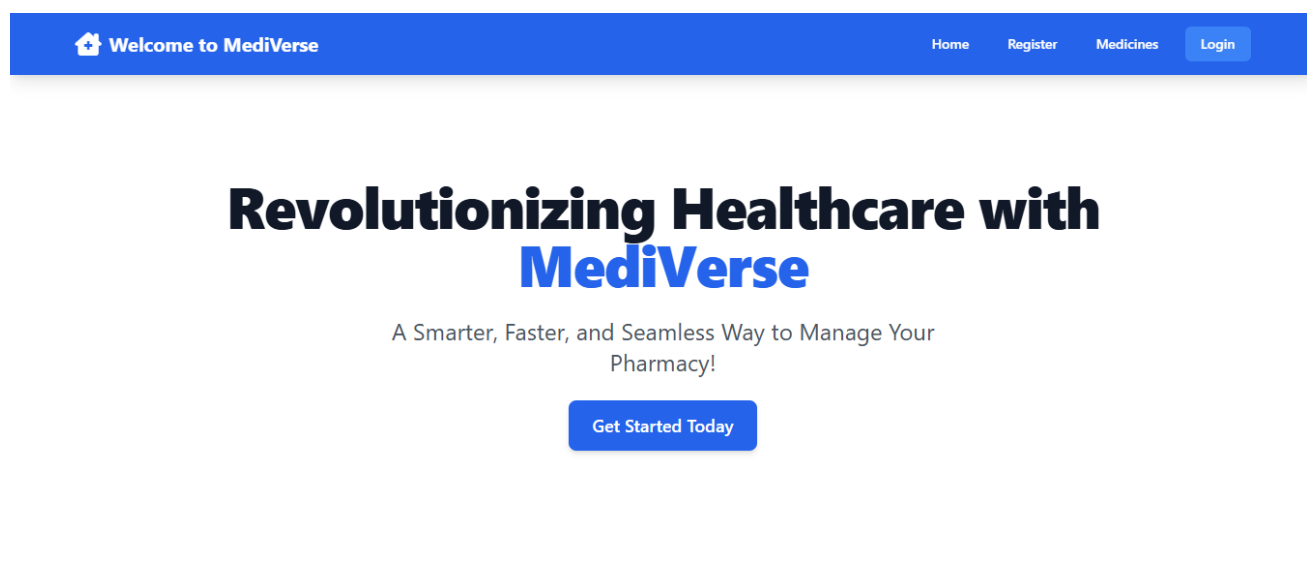
The backend is built with **MySQL**, employing relational tables, stored procedures, and triggers to manage transactions securely and efficiently. The system ensures automated updates of stock, purchase totals, and customer balances through database triggers, reducing manual intervention and error rates.

Overall, the **Pharmacy Management System** streamlines the workflow of pharmacy operations by ensuring **real-time data synchronization, accurate billing, and efficient stock management**. It serves as a **centralized, scalable, and secure** solution for managing all pharmacy-related activities, benefitting both staff and customers.

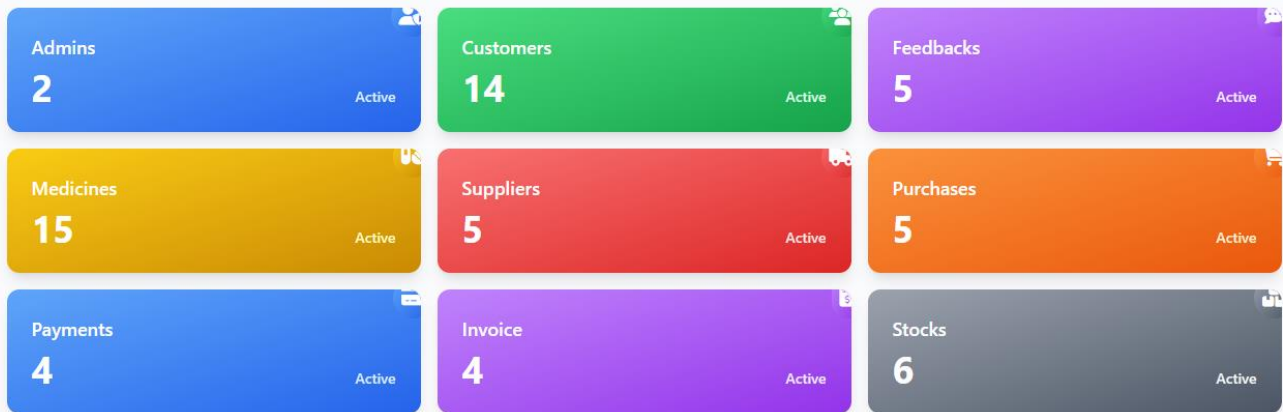
4.1 Screenshots:

The following section presents a series of screenshots showcasing the key features and interfaces of the developed Pharmacy Management System.

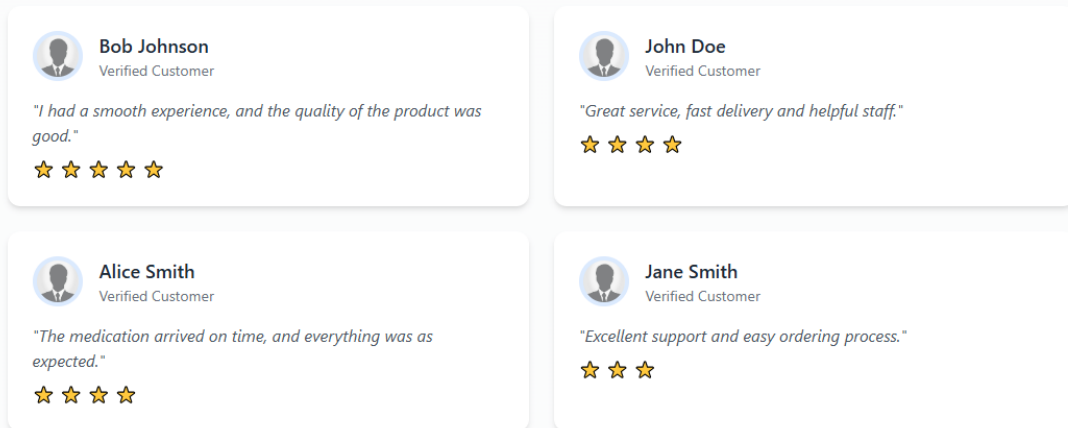
Home page



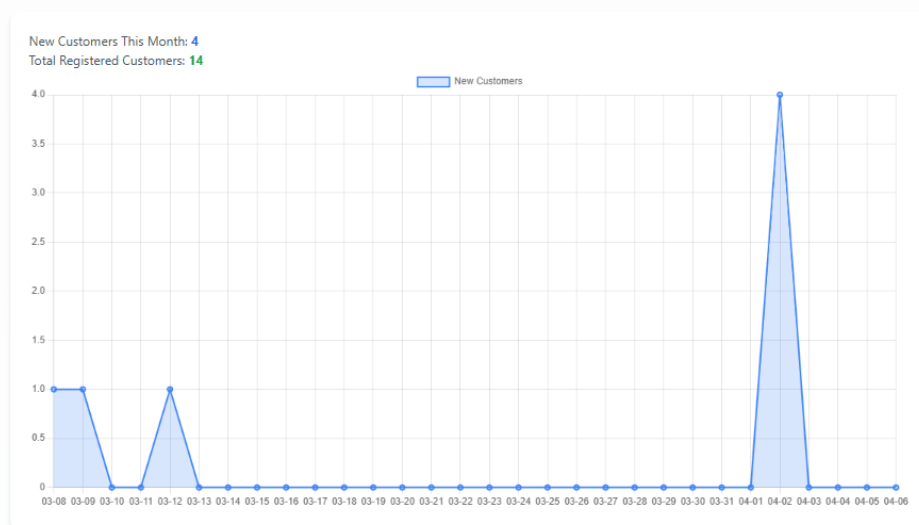
Dashboard Overview




What Our Customers Say




Customer Growth



Customer Registration Page

 Welcome to MediVerse


HomeRegisterMedicinesLogin




Create Account

Register for a new account


Your Name *

 Enter your full name

Email Address *

 Enter your email address


Phone Number *

 Enter your phone number


Upload Photo Only image files are allowed!

Choose FileNo file chosen


Address Type

 Home


Street

 Enter your street


City

 Enter your city


State

 Enter your state

Zip Code


 Enter your zip code

Password *


 Enter your password

Create Account

Login Page

 Welcome to MediVerse

HomeRegisterMedicinesLogin




Welcome Back

Sign in to your account


Login as *

Administrator

Username / Email *

 john.doe@example.com

Password *




☒ Remember Me

[Forgot Password?](#)


Sign in

Don't have an account? [Create Account](#)

Contact Page

 Welcome to MediVerse

HomeRegisterMedicinesLogin



Contact Me

Have questions? We'd love to hear from you.

Your Name *

Enter your name

Email Address *

Enter your email

Subject *


Enter message subject

Message *

Enter your message

Send Message


Customer Dashboard page

 Welcome John Doe

HomeRegisterMedicinesCustomerLogout

John Doe's Dashboard

Manage your profile, addresses, view your purchase history and purchase



John Doe
Member since 1/3/2025

1

john.doe@example.com

9876543210


Balance: ₹-15.00

AddressesFeedbackPurchasePurchase History

Purchase Medicine Here

Your Cart

Cart



Your cart is empty.

+ Add medicines to your cart

Dept Of CSE | DBMS-Mini-Project

-28-

Poovarasan S | U25UV22T064042

Payment Page

Welcome John Doe

HomeRegisterMedicinesCustomerLogout

Payment for Medicine

Please review your order details and proceed with the payment.

Medicine Details

Aspirin

Composition: Aspirin 100mg Supplier: Unknown

Expiry Date: 2026-06-30 Quantity: 1

Price: ₹5.00 Total: ₹5.00

Payment Summary

Total Amount: ₹5.00

Customer Balance: ₹-15.00

Final Amount: ₹-10

Payment Amount:

₹-10

Payment Method:

Credit Card

Make Payment

Admin Dashboard page

Welcome admin1

HomeRegisterMedicinesAdminLogout

Admin Panel

Total Income Your Income

₹425.00 ₹75.00

+ Add New Customer Filter

Manage Customers

John Doe

ID: 1

Joined: Sat Mar 01 2025 00:00:00 GMT+0530 (India Standard Time)

Balance

₹-15.00

Total Spent

₹80.00

john.doe@example.com

9876543210

Alice Smith

ID: 2

Joined: Sun Mar 02 2025 00:00:00 GMT+0530 (India Standard Time)

Balance

₹5.00

Total Spent

₹20.00

alice.smith@example.com

9876543211

Bob Johnson

ID: 3

Joined: Mon Mar 03 2025 00:00:00 GMT+0530 (India Standard Time)

Balance

₹-165.00

Total Spent

₹210.00

bob.johnson@example.com

9876543212

Dept Of CSE | DBMS-Mini-Project

-29-

Poovarasana S | U25UV22T064042

Chapter-5 : Conclusion And Future Enhancement

The **Pharmacy Management System (PMS)**, developed as part of the **DBMS mini-project under NEP 2021**, effectively demonstrates the real-world application of database management concepts in the domain of pharmacy operations. The system encompasses essential functionalities such as **medicine inventory management, supplier and customer tracking, purchase recording, invoice generation, and payment and balance management.**

Overall, this project highlights the practical application of key DBMS concepts, including:

- **Data storage, retrieval, and manipulation** using MySQL.
- **Transaction management** to maintain consistency and accuracy in records.
- **Use of triggers and stored procedures** for automation and enforcing business rules.
- **Normalization and indexing** to enhance query performance and avoid redundancy.
- **Error handling and data validation** to maintain data integrity and reliability.

Future Enhancements

While the current system successfully handles the core operations of a pharmacy, the following future enhancements could be considered to further improve its functionality and usability:

- **Integration with barcode scanning** for faster billing and stock updates.
- **Automated SMS/email alerts** for low-stock medicines or upcoming expiry dates.
- **Implementation of a role-based user system** (e.g., admin, cashier, pharmacist) for better access control.
- **Online ordering system** for customers to place medicine orders remotely.
- **Mobile-friendly web interface or a dedicated Android/iOS application** for managing the pharmacy on the go.
- **GST and tax handling features** for compliance and financial accuracy.
- **Detailed analytics and sales dashboards** for monthly or yearly reporting.

These enhancements would further streamline pharmacy operations, improve user experience, and make the system more adaptable to real-world scenarios.

Chapter-5 : Bibliography

- 1] Ramez Elmasri and Shamkant B Navathe, Fundamentals of Database Systems, 6th ed. Addison Wesley, 2009.
- [2] Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database System Concepts, 6th ed. Tata McGraw-Hill, 2010.
- [3] Fundamental of Database Systems by Ramez Elmasri and Shamkant B Navathe, Sixth Edition, Addison Wesley, 2011.
- [4] Database System Concepts, Sixth Edition, Abraham Silberschatz, Henry F. Korth, S. Sudarshan: Tata McGraw-Hill, 2010.
- [5]An Introduction to Database Systems by C.J. Date, A. Kannan, S. Swamynathan, 8th Edition, Pearson Education, 2006.
- [6] Flask Documentation
- [7] MySQL Documentation
- [8] Node Documentation