**Database Fundamentals (CYBS-1913)**

**Database Project (Expanding Rough Draft Report)**

**Student: Daren Diaz**

**OU ID: 113487126**

**Professor: Dr. Muhammad Riaz**

**Table of Contents**

1. **Introduction**
   * Overview of the Medication Inventory System Database
   * Objectives and Purpose
   1. **Expanded Idea**
   * Detailed Description of the Medication Inventory System Database
   * Features and Functionalities
     + Core Functionalities
     + Optional Advanced Features

**7. Visualization Example for Optional Advanced Features**

**8-10 Database Schema Design**

* + Tables and Attributes
  + Relationships Between Tables

**11-13. Example SQL Queries**

* + Retrieve Low-Stock Medications
  + List Expired Medications
  + Track Dispensed Medications
  + Supplier Order History
  + Medication Usage Trends

**14. Implementation Steps**

* + Database Design and Creation
  + Backend Integration
  + Frontend Development (Optional)
  + Testing and Optimization
  + Deliverables
  + ER Diagram
  + SQL Scripts
  + Optional Demo Application

**15. Real-World Medication Inventory Systems**

* 1. Overview of Existing Systems
  2. Key Features of Popular Systems
  3. Challenges in Current Solutions

**16. Conclusion**

* 1. Summary and Future Opportunities for the Project

**17-20. Step by Step Guide to Execute Project**

**Expanded Idea: Medication Inventory System Database**

The **Medication Inventory System Database** is a database-focused project designed to manage the stock, supply, and dispense of medications in a pharmacy or hospital setting. This system ensures optimal inventory levels, reduces wastage from expired medications, and improves operational efficiency.

**Project Overview**

* **Objective:** Create a database system to track medications, suppliers, prescriptions, and stock levels.
* **Purpose:** Prevent stockouts, manage expirations, and streamline dispensing processes.
* **Target Audience:** Hospitals, pharmacies, and healthcare facilities.

**Features and Functionalities**

**Core Functionalities**

* **Medication Inventory Management:**
  + Track medication details (name, type, dosage, batch number, etc.).
  + Maintain stock levels for each medication.
  + Monitor expiration dates to reduce wastage.
* **Supplier Management:**
  + Record supplier details (name, contact, address).
  + Link medications to their respective suppliers for reorder purposes.
* **Stock Replenishment:**
  + Automatically generate reorder alerts when stock levels fall below a set threshold.
  + Record order histories for audit purposes.
* **Dispensing and Tracking:**
  + Track medications are dispensed to patients.
  + Record prescription details to link medications to patient records.
* **Reporting and Analytics:**
  + Generate reports on:
    - Low-stock medications.
    - Expiring medications within a specific time frame.
    - Monthly or quarterly usage trends.

**Optional Advanced Features (4 Examples)**

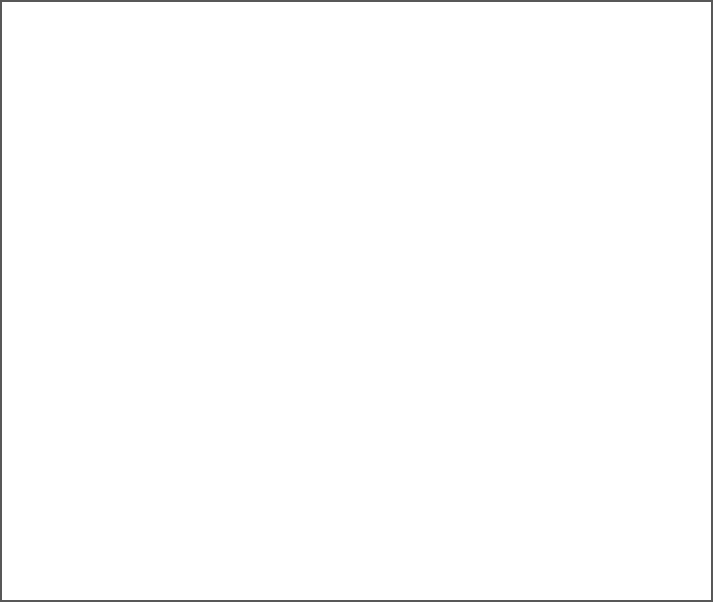
* **Integration with a Point-of-Sale (POS) system** for seamless dispensing.

**Example:** *A pharmacy sells medications over the counter. When a customer purchases a medication:*

* The POS system automatically deducts the sold quantity from the database’s current\_stock field in the **Medications** table.
* The system generates a receipt that includes details like:
  + - 1. Medication name, quantity, and price.
      2. Date and time of the transaction.
* Integration ensures real-time updates to the inventory database, avoiding manual entry errors and ensuring stock accuracy.

**Use Case:**

* **Scenario:** A customer buys 2 packs of Paracetamol (500mg).



**Database Update Query:**

UPDATE Medications

SET current\_stock = current\_stock - 2

WHERE medication\_id = 1;

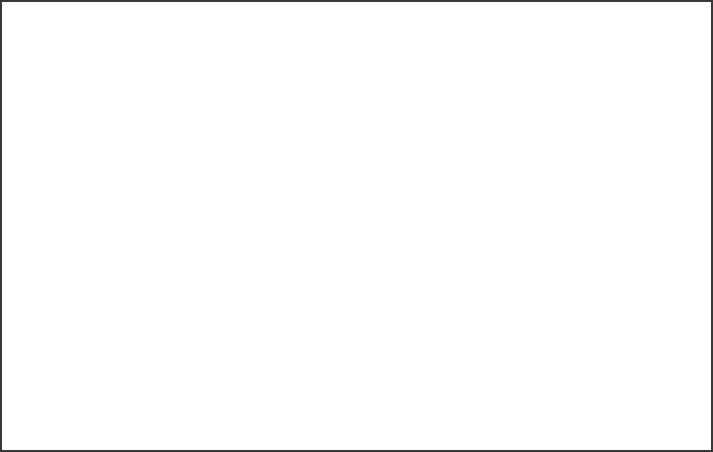




* **Barcode or QR Code Scanning** for quick medication identification and tracking.

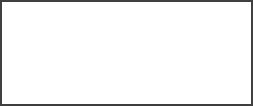
**Example:** *When restocking or dispensing medications:*

* Staff can scan a barcode or QR code on the medication packaging using a scanner or a mobile device.
* The system retrieves details like **medication\_id, name, batch\_number**, and **expiration\_date** directly from the database.
* This reduces errors, speeds up operations, and ensures proper medication identification.



**Use Case:**

* **Scenario:** A technician scans a barcode for Ibuprofen (200mg).
* **Result:** The system displays:



Medication Name: Ibuprofen

Type: Tablet

Expiration Date: 2026-03-15

Current Stock: 300

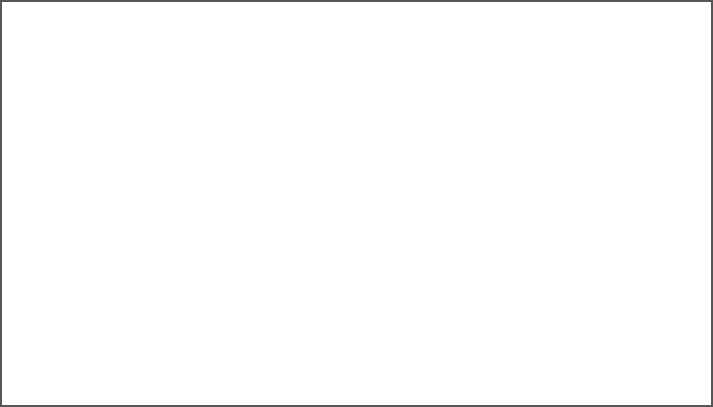
* **User Authentication:** Role-based access (e.g., pharmacists, admins, technicians).

**Example:** *Different user roles (e.g., pharmacists, admins, and technicians) have varying levels of access:*

* **Admins:** Full access to the system, including managing users, adding medications, and viewing audit trails.
* **Pharmacists:** Can view and update medication stock, dispense medications, and generate reports.
* **Technicians:** Can only view and update inventory stock levels but cannot access sensitive patient or financial data.

**Use Case:**

* **Scenario:** A pharmacist logs in to dispense a medication.



* **Role Verification:** The system checks the pharmacist’s role before granting access to the dispensing module.

**SQL Query for Role-Based Access:**



SELECT role

FROM Users

WHERE user\_id = 101;



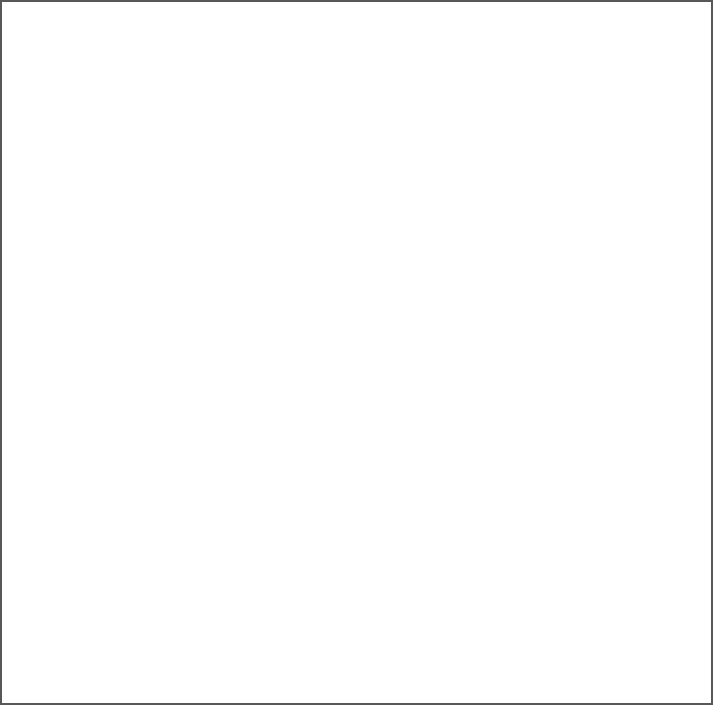
* **Audit Trails:** Log every action for compliance and accountability.

**Example:** *Every action performed in the system is logged for accountability and compliance with regulations such as HIPAA. Actions logged might include*:

* Medication stock updates.
* User logins and logouts.
* Prescription dispensing.

**Use Case:**

* **Scenario:** A technician restocks 50 units of Amoxicillin (250mg).
* **Audit Trail Entry:**



User: Technician01



Action: Restock

Medication: Amoxicillin (250mg)

Quantity: 50

Date/Time: 2025-01-16 14:30

**SQL Example to Log Actions:**

INSERT INTO Audit\_Trails (user\_id, action, description, action\_time)

VALUES (201, 'Restock', 'Added 50 units of Amoxicillin (250mg)', NOW());

**Visualization Example for Advanced Features**

**1. Barcode Scanning Workflow Diagram**

* **Step 1:** Scan the medication.
* **Step 2:** System retrieves medication details from the database.
* **Step 3:** Perform the required action (dispense or restock).

**2. Audit Trail Table Example**

| **Log ID** | **User ID** | **Action** | **Description** | **Timestamp** |
| --- | --- | --- | --- | --- |
| 1001 | 201 | Login | User Technician01 logged in | 2025-01-16 14:00 |
| 1002 | 201 | Restock | Added 50 units of Amoxicillin | 2025-01-16 14:30 |
| 1003 | 301 | Dispense | Dispensed 2 units of Paracetamol | 2025-01-16 15:00 |

**3. Role-Based Access Flowchart**

* **Start** → User Logs In → System Verifies Role:
  + If "Admin" → Grant Full Access.
  + If "Pharmacist" → Grant Limited Access (Stock and Dispense).
  + If "Technician" → View-Only Access.

**Database Schema Design**

**(Tables)**

1. **Medications**:

medication\_id (Primary Key)

name

type (e.g., tablet, injection)

dosage

batch\_number

expiration\_date

current\_stock

reorder\_level

supplier\_id (Foreign Key)

| **medication\_id** | **name** | **type** | **dosage** | **batch\_number** | **expiration\_date** | **current\_stock** | **reorder\_level** | **supplier\_id** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Paracetamol | Tablet | 500mg | BATCH001 | 2025-06-30 | 120 | 0 | 101 |
| 2 | Amoxicillin | Capsule | 250mg | BATCH002 | 2024-12-15 | 40 | 20 | 102 |
| 3 | Ibuprofen | Tablet | 200mg | BATCH003 | 2026-03-15 | 300 | 100 | 103 |
| 4 | Insulin | Injection | 10mL | BATCH004 | 2024-10-05 | 15 | 5 | 104 |

1. **Suppliers**:

supplier\_id (Primary Key)

name

contact\_number

email

address

| **supplier\_id** | **name** | **contact\_number** | **email** | **address** |
| --- | --- | --- | --- | --- |
| 101 | MedSupplies Inc. | 123-456-7890 | contact@medsupplies.com | 123 Healthcare Blvd, NY |
| 102 | PharmaSource Co. | 987-654-3210 | info@pharmasource.com | 45 Pharma Lane, LA |
| 103 | Global Pharma Supplies | 456-789-1230 | support@globalpharma.com | 789 Industry St, Chicago |
| 104 | Local Distributors LLC | 321-654-9870 | sales@localdist.com | 67 Local Rd, Houston |

1. **Dispensed\_Medications**:

dispense\_id (Primary Key)

medication\_id (Foreign Key)

patient\_id (Foreign Key)

quantity

dispensed\_by (Pharmacist or system user ID)

dispense\_date

| **dispense\_id** | **medication\_id** | **patient\_id** | **quantity** | **dispensed\_by** | **dispense\_date** |
| --- | --- | --- | --- | --- | --- |
| 5001 | 1 | 3001 | 2 | 101 | 2025-01-01 |
| 5002 | 2 | 3002 | 1 | 102 | 2025-01-02 |
| 5003 | 3 | 3003 | 3 | 103 | 2025-01-03 |
| 5004 | 4 | 3004 | 1 | 104 | 2025-01-04 |

1. **Patients** (Optional):

patient\_id (Primary Key)

first\_name

last\_name

date\_of\_birth

contact\_number

| **patient\_id** | **first\_name** | **last\_name** | **date\_of\_birth** | **contact\_number** |
| --- | --- | --- | --- | --- |
| 3001 | John | Doe | 1980-05-15 | 123-456-7890 |
| 3002 | Jane | Smith | 1990-11-22 | 987-654-3210 |
| 3003 | Emily | Johnson | 1975-03-10 | 456-789-1234 |
| 3004 | Michael | Brown | 1985-07-18 | 321-654-9876 |

1. **Orders**:

order\_id (Primary Key)

medication\_id (Foreign Key)

supplier\_id (Foreign Key)

quantity\_ordered

order\_date

delivery\_date

| **dispense\_id** | **medication\_id** | **patient\_id** | **quantity** | **dispensed\_by** | **dispense\_date** |
| --- | --- | --- | --- | --- | --- |
| 5001 | 1 | 3001 | 2 | 101 | 2025-01-01 |
| 5002 | 2 | 3002 | 1 | 102 | 2025-01-02 |
| 5003 | 3 | 3003 | 3 | 103 | 2025-01-03 |
| 5004 | 4 | 3004 | 1 | 104 | 2025-01-04 |

**(Relationships)**

* **Medications** → **Suppliers**: One-to-Many

A single supplier can provide multiple medications, but each medication is associated with one primary supplier. This relationship allows accurate tracking of medication sources and facilitates streamlined reordering processes.

* **Dispensed\_Medications** → **Medications**: Many-to-One

Multiple dispensing records may refer to the same medication. This relationship ensures that inventory data is consistent, enabling accurate tracking of stock usage and dispensing trends for each medication.

* **Dispensed\_Medications** → **Patients**: Many-to-One

A single patient may have multiple medications dispensed over time. This relationship links dispensing records to individual patients, allowing for detailed analysis of patient medication histories and ensuring compliance with healthcare regulations.

* **Orders** → **Medications** and **Suppliers**: Many-to-One

Each order is associated with a specific medication and its respective supplier. This relationship provides a comprehensive view of order history and inventory procurement, supporting better supplier management and medication stock control.

**Why Do These Relationships Matter?**

* **Data Integrity:** Relationships enforce constraints to prevent errors, such as assigning medication to multiple primary suppliers.
* **Query Efficiency:** Structured relationships simplify complex queries, making it easier to retrieve meaningful insights (e.g., supplier-specific medication reports or patient dispensing histories).
* **Scalability:** The schema is designed to handle increasing data volumes by logically separating entities and their relationships, minimizing redundancy.
* **Real-World Alignment:** These relationships mirror real-world operations, ensuring the database accurately models how medications, suppliers, patients, and orders interact in a healthcare environment.

**Example SQL Queries**

1. **Retrieve Low-Stock Medications:**

SELECT name, current\_stock, reorder\_level

FROM Medications

WHERE current\_stock <= reorder\_level;

* **Purpose:** Identify medications that need to be restocked.
* **Query Explanation:** This query selects the name, current\_stock, and reorder\_level of medications where the current stock is less than or equal to the reorder level. It helps ensure timely replenishment of critical stock.

1. **List Expired Medications:**

SELECT name, batch\_number, expiration\_date

FROM Medications

WHERE expiration\_date < CURDATE();

* **Purpose:** Find medications that are no longer usable due to expiration.
* **Query Explanation:** This query retrieves the name, batch\_number, and expiration\_date of medications where the expiration date has already passed (compared to the current date). This ensures expired medications can be removed from inventory.

1. **Track Dispensed Medications:**

SELECT p.first\_name, p.last\_name, m.name AS medication\_name, d.quantity, d.dispense\_date

FROM Dispensed\_Medications d

JOIN Patients p ON d.patient\_id = p.patient\_id

JOIN Medications m ON d.medication\_id = m.medication\_id

WHERE d.dispense\_date BETWEEN '2025-01-01' AND '2025-01-31';

* **Purpose:** Monitor medications dispensed within a specific date range.
* **Query Explanation:** This query retrieves the first and last names of patients, the medication name, quantity dispensed, and the dispensing date. It uses JOIN statements to combine data from Dispensed\_Medications, Patients, and Medications. The WHERE clause filters results in a specific date range.

1. **Supplier Order History:**

SELECT s.name AS supplier\_name, o.medication\_id, m.name AS medication\_name, o.quantity\_ordered, o.order\_date

FROM Orders o

JOIN Suppliers s ON o.supplier\_id = s.supplier\_id

JOIN Medications m ON o.medication\_id = m.medication\_id

ORDER BY o.order\_date DESC;

* **Purpose:** Review the history of medication orders placed with suppliers.
* **Query Explanation:** This query retrieves the supplier's name, medication ID, medication name, quantity ordered, and order date. It uses JOIN statements to link the Orders, Suppliers, and Medications tables. Results are ordered by order\_date in descending order to show the most recent orders first.

1. **Medication Usage Trends:**

SELECT m.name AS medication\_name, SUM(d.quantity) AS total\_dispensed

FROM Dispensed\_Medications d

JOIN Medications m ON d.medication\_id = m.medication\_id

GROUP BY m.name

ORDER BY total\_dispensed DESC;

* **Purpose:** Analyze which medications are most frequently dispensed.
* **Query Explanation:** This query calculates the total quantity dispensed for each medication by summing up the quantities in the Dispensed\_Medications table. It uses a GROUP BY clause to group results by medication name and orders the output in descending order of total dispensed quantity, highlighting the most-used medications.

**Implementation Steps**

1. **Database Design:**

* Use an ERD tool to design the schema.
* Normalize tables to eliminate redundancy.

1. **Database Creation:**

* Use a relational database system (e.g., MySQL, PostgreSQL).
* Write SQL scripts to create and populate tables.

1. **Backend Integration:**

* Use Python (Flask/Django) or Node.js to build APIs for interacting with the database.

1. **Frontend (Optional):**

* Create a simple UI for pharmacists/admins using React, Angular, or any preferred framework.

1. **Testing and Optimization:**

* Test SQL queries for efficiency.
* Optimize using indexing, partitioning, or caching if necessary.

**Deliverables**

* **ER Diagram** showing the relationships between tables.
* **SQL Scripts** for creating tables, inserting sample data, and executing queries.
* **Demo Application (Optional):** A basic interface to demonstrate the database functionalities.

**Real-World Medication Inventory System Databases**

Medication inventory system databases are widely implemented in healthcare and pharmaceutical industries to manage medication stock, track dispensing, and optimize operations. Prominent examples include:

1. **Pyxis MedStation (BD Pyxis):**
   * A leading automated medication dispensing system used in hospitals and pharmacies.
   * It integrates with electronic health records (EHR) to track medication usage and ensure accurate dispensing.
   * The system's database monitors stock levels, expiration dates, and usage trends in real-time, reducing errors and waste.
2. **Omnicell Inventory Management:**
   * A comprehensive medication and supply management system that combines databases with automation.
   * It includes features for inventory tracking, automated restocking, and reporting on medication usage patterns.
   * Omnicell systems emphasize compliance with healthcare regulations like HIPAA and focus on patient safety.
3. **McKesson Pharmacy Systems:**
   * Offers solutions for retail and hospital pharmacies, focusing on inventory control and supply chain efficiency.
   * Their database-driven systems support advanced analytics to optimize procurement and reduce costs while ensuring medication availability.
4. **Epic Willow:**
   * A module of the Epic EHR system specifically designed for pharmacy and medication management.
   * Its database tracks inventory, prescriptions, and patient information, ensuring seamless integration across healthcare facilities.
   * Willow emphasizes interoperability, allowing healthcare providers to make data-driven decisions.
5. **Cerner PharmNet:**
   * A pharmacy management solution that supports inventory control, prescription tracking, and dispensing workflows.
   * The database includes advanced features for monitoring medication interactions and improving operational efficiency.

**Key Features of Existing Systems**

* **Real-Time Tracking:** Continuous monitoring of inventory levels to prevent stockouts and overstocking.
* **Integration with EHRs:** Seamless integration with patient records for accurate and safe dispensing.
* **Analytics and Reporting:** Built-in analytics tools to identify trends, optimize inventory, and predict future needs.
* **Compliance and Security:** Adherence to regulations like HIPAA and measures to secure sensitive data.

**Challenges in Existing Systems**

While effective, these systems face challenges such as high implementation costs, complexity in scaling, and limited customization options for smaller clinics or pharmacies. Addressing these gaps presents opportunities for innovative, cost-effective solutions tailored to specific needs, such as open-source or cloud-based inventory systems.

By analyzing these real-world systems, we can derive inspiration for developing a scalable and efficient medication inventory system tailored to specific healthcare settings.

**Conclusion**

The **Medication Inventory System Project** demonstrates the importance and versatility of database systems in real-world healthcare applications. As discussed in Chapter 1 of *Fundamentals of Database Systems*, databases are integral to modern society, facilitating activities like banking, online shopping, and healthcare management. This project aligns with traditional database applications, which involve the storage and retrieval of textual and numeric data, while addressing the specific needs of medication management in hospitals and pharmacies.

A database is more than a collection of data; it is a logically coherent structure that reflects real-world processes, known as the miniworld or universe of discourse (UoD). In this project, the database mirrors the workflows of medication inventory, supplier relationships, and patient dispensing. By designing structured relationships—such as between medications and suppliers or patients and dispensed medications—the project ensures data integrity, efficient querying, and real-time updates to reflect changes in the miniworld.

The significance of database fundamentals, including data abstraction and program-data independence, is evident in this project. For example, SQL queries provide a high-level interface for users to retrieve information on low-stock medications or track dispensing trends, abstracting the complexities of physical data storage. Furthermore, the self-describing nature of databases ensures the system remains flexible and scalable as new requirements emerge, such as integrating advanced features like barcode scanning or analytics.

By leveraging database management principles, this project addresses critical challenges in healthcare, such as preventing stockouts, reducing medication waste, and improving operational efficiency. It also highlights how the principles of database design—such as normalization, enforcing constraints, and supporting multiple user views—create a robust and reliable system that aligns with the broader role of databases in supporting decision-making and improving service quality in various domains.

In conclusion, this project is a practical application of database concepts that not only fulfills real-world need but also provides a foundation for exploring more advanced systems, such as big data and analytics in healthcare. It underscores how database technology transforms complex real-world interactions into manageable, efficient, and meaningful systems.

**Steps to Complete the Medication Inventory System Project Using MySQL**

**1. Plan and Design the Project**

* **Understand Requirements:**
  + Define the scope, features, and functionalities (e.g., inventory tracking, user roles, barcode scanning).
* **Create an ER Diagram:**
  + Design the database schema, including tables, attributes, and relationships.
  + Tools like MySQL Workbench, Lucidchart, or Draw.io can be used to create the ER diagram.
* **Identify Advanced Features:**
  + Decide on optional features like POS integration, audit trails, or user authentication.

**2. Set Up the MySQL Environment**

* **Install MySQL:**
  + Download and install MySQL Server and MySQL Workbench from the [MySQL website](https://www.mysql.com/).
* **Create a New Database:**
  + Use the MySQL Workbench or the MySQL command line to create a database for the project:

CREATE DATABASE MedicationInventory;



* **Set Up Users and Roles:**
  + Create database users with specific permissions for admin, pharmacists, and technicians:

CREATE USER 'pharmacist'@'localhost' IDENTIFIED BY 'password';

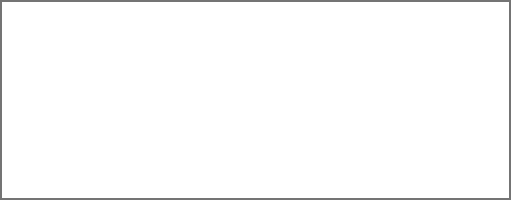


GRANT SELECT, UPDATE, INSERT ON MedicationInventory.\* TO 'pharmacist'@'localhost';

**3. Create Tables**

* **Define Schema:**
  + Create the necessary tables based on the design:
    - **Medications Table:**

CREATE TABLE Medications (



medication\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

type VARCHAR(50),

dosage VARCHAR(50),

batch\_number VARCHAR(50),

expiration\_date DATE,

current\_stock INT,

reorder\_level INT,

supplier\_id INT,

FOREIGN KEY (supplier\_id) REFERENCES Suppliers(supplier\_id)

);

* + - Repeat similar steps for **Suppliers**, **Dispensed\_Medications**, **Patients**, and **Orders** tables.
* **Test Table Creation:**
  + Run queries to ensure tables are created correctly:

SHOW TABLES;



**4. Populate Tables with Sample Data**

* **Insert Sample Records:**
  + Populate tables with initial data for testing:

INSERT INTO Suppliers (supplier\_id, name, contact\_number, email, address)



VALUES (101, 'MedSupplies Inc.', '123-456-7890', 'contact@medsupplies.com', '123 Healthcare Blvd, NY');

* **Add Medications:**
  + Populate the Medications table:

INSERT INTO Medications (name, type, dosage, batch\_number, expiration\_date, current\_stock, reorder\_level, supplier\_id)



VALUES ('Paracetamol', 'Tablet', '500mg', 'BATCH001', '2025-06-30', 120, 50, 101);

**5. Write SQL Queries**

* **Develop Core Queries:**
  + Write SQL scripts for functionalities like:
    - Retrieving low-stock medications:

SELECT name, current\_stock



FROM Medications

WHERE current\_stock <= reorder\_level;

* + - Listing expired medications:

SELECT name, expiration\_date



FROM Medications

WHERE expiration\_date < CURDATE();

* **Test Queries:**
  + Run and refine queries using MySQL Workbench.

**6. Implement Optional Advanced Features**

* **POS Integration:**
  + Simulate a transaction by updating the stock upon medication purchase.
  + UPDATE Medications
  + SET current\_stock = current\_stock - 1
  + WHERE medication\_id = 1;
* **User Authentication:**
  + Use MySQL’s role-based authentication for access control.
* **Audit Trails:**
  + Add an **Audit\_Trails** table to log actions:
  + CREATE TABLE Audit\_Trails (
  + log\_id INT AUTO\_INCREMENT PRIMARY KEY,
  + user\_id INT,
  + action VARCHAR(255),
  + description TEXT,
  + action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP
  + );

**7. Build the Frontend (Optional)**

* **Choose a Technology Stack:**
  + Use a programming language like Python (Flask or Django), JavaScript (Node.js), or PHP to create a user-friendly interface.
* **Connect to the Database:**
  + Use a library like mysql-connector-python in Python:
  + import mysql.connector
  + connection = mysql.connector.connect(
  + host="localhost",
  + user="root",
  + password="yourpassword",
  + database="MedicationInventory"
  + )

**8. Test the System**

* **Test Use Cases:**
  + Verify that core functionalities work as expected, such as:
    - Retrieving medication data.
    - Handling low-stock alerts.
    - Adding audit logs for changes.
* **Debug Errors:**
  + Resolve any SQL errors or logic issues.

**9. Optimize the Database**

* **Index Frequently Queried Columns:**
  + Add indexes to improve query performance:
  + CREATE INDEX idx\_medication\_name ON Medications(name);
* **Backup the Database:**
  + Regularly back up the database to prevent data loss:
  + mysqldump -u root -p MedicationInventory > backup.sql

**10. Deploy the System**

* **Host the Database:**
  + Deploy the database on a local server or cloud service like AWS RDS, Google Cloud, or Azure.
* **Provide User Training:**
  + Train end-users (e.g., pharmacists) on how to use the system.

**11. Maintain and Update**

* **Monitor System Performance:**
  + Regularly analyze query performance and resolve bottlenecks.
* **Add Features:**
  + Expand the system with new features like analytics dashboards or integration with EHRs.