

Table of Contents

1. Introduction	1
1.1. Background and Problem Statement.....	1
1.2. Objectives	1
2. Database Design.....	1
2.1. Entity Description.....	1
2.2. Relationship and Cardinality	2
2.3. Normalization	3
3. Tables	3
4. Transaction Handling and Triggers	3
4.1. Transactions	3
Transaction 1: Refund Processing.....	3
Transaction 2: Expired Medicine Cleanup.....	4
4.2. Trigger Description.....	4
Trigger 1: TR_Medicine_LowStock.....	4
Trigger 2: TR_Sales_PreventExpired	4
Trigger 3: TR_Sales_Insert.....	4
Trigger 4: TR_Sales_Update	4
Trigger 5: TR_Sales_Delete	4
5.1. Function: FN_CheckAvailability.....	5
5.2. Stored Procedure: SP_AddSale	5
6. Recovery Strategy.....	5
7. Security Strategy	5
8. Conclusion	6

1. Introduction

1.1. Background and Problem Statement

A well-managed pharmacy inventory and sales system is needed for ensuring both public safety and business efficiency. Pharmacies handle sensitive products such as medications that have expiration dates and they must be properly tracked to avoid errors like selling expired drugs. Managing stock levels and customer transactions accurately helps prevent financial losses and improves customer satisfaction. This project implements a relational database system that enhances inventory control, improves sales tracking, and enforces data integrity using automation tools such as triggers and stored procedures.

Manual or poorly designed systems often lead to critical errors like:

- ✓ Selling expired medications
- ✓ Running out of essential drugs (stockouts)
- ✓ Poor record-keeping
- ✓ Difficulty in tracking sales history
- ✓ Revenue mismanagement due to lack of automation

This project addresses these problems by designing a fully functional and normalized pharmacy management system.

1.2. Objectives

The primary objectives of the system are to build a normalized relational database to store medicine, customer, and sales data, automate low-stock alerts using triggers, enforce data integrity through the use of constraints and triggers, prevent expired medicines from being sold and enforce recoverability with backup and restore mechanisms and granting security and access control through permissions.

2. Database Design

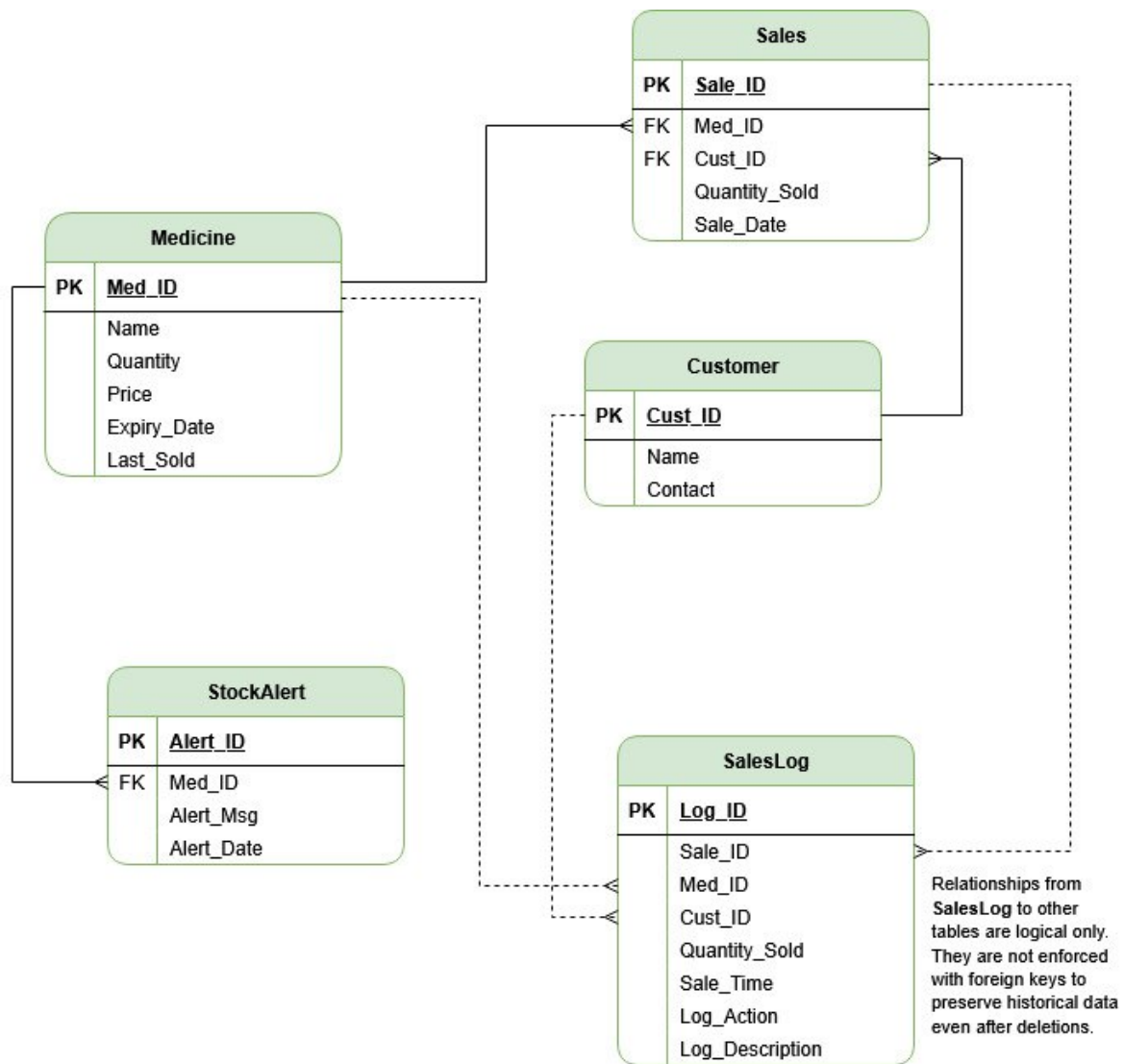
2.1. Entity Description

The system is designed around five main tables: *Medicine*, *Customer*, *Sales*, *StockAlert*, and *SalesLog*. The *Medicine* table stores drug details including name, quantity, price, expiry date, and last sold timestamp. *Customer* table contains customer information such as name and contact number. *Sales* table records customer purchases. *StockAlert* stores alerts when medicine quantity

falls below a defined threshold and the SalesLog table records all sales related insertions, deletions, and updates for audit purposes.

2.2. Relationship and Cardinality

The relationships between these entities are explained in the following ER Diagram:



From this diagram we can see that:

- ✓ One Customer can make many Sales.
- ✓ One Medicine can be sold in many Sales.
- ✓ One Medicine can trigger multiple StockAlerts.
- ✓ Each Sale is linked to multiple logs in SalesLog.

- ✓ Foreign key constraints ensure referential integrity between Sales, Medicine, StockAlert and Customer tables.

2.3. Normalization

The design follows Third Normal Form (3NF) which means no redundant data is stored, each table represents a single subject and all non-key attributes are fully dependent on the primary key, ensuring data consistency.

3. Tables

SQL CREATE TABLE statements are provided for all five main tables, including necessary constraints like primary keys and foreign keys where applicable in addition to 'NOT NULL' to ensure mandatory fields and Indexes added on frequently queried fields (like Med_ID and Cust_ID) to optimize query performance.

Sample records were inserted into the Medicine, Customer, and Sales tables for testing. The StockAlert and SalesLog tables were left empty initially to be populated by triggers.

4. Transaction Handling and Triggers

4.1. Transactions

Transaction 1: Refund Processing

If a customer returns a product (Sale_ID = 2), this transaction deletes the sale and updates the medicine quantity. It checks if a sale exists with Sale_ID = 2. If found, it fetches the medicine ID and quantity sold, deletes the sale record, updates the medicine quantity (restores stock) and commits the transaction.

It follows **ACID** principles:

- ✓ **Atomicity:** Sale is only refunded if all steps succeed
- ✓ **Consistency:** Data remains valid post-refund.
- ✓ **Isolation:** Uses READ COMMITTED, ensuring no dirty reads.
- ✓ **Durability:** Changes are committed, becoming permanent.

Transaction 2: Expired Medicine Cleanup

This deletes expired medicines and their associated sales and logs. It deletes related records from SalesLog, then Sales, then Medicine (if expired). It uses TRY...CATCH with rollback on failure.

This transaction is also ACID-compliant:

- ✓ **Atomicity:** All deletes are part of one transaction.
- ✓ **Consistency:** Ensures no sales/logs exist for expired medicine.
- ✓ **Isolation:** READ COMMITTED level.
- ✓ **Durability:** On success, changes persist.

4.2. Trigger Description

There are five database triggers to automate essential operations and enforce business rules.

Trigger 1: TR_Medicine_LowStock

This automatically generates alerts for low-stock medicines when medicine stock levels fall below 10. It is activated after any insert or updates to Medicine. It also checks if similar alert exists to prevent duplicates. This mechanism ensures timely alerts for the pharmacies to restock.

Trigger 2: TR_Sales_PreventExpired

This is used to prevent insertion of sales for expired medicines by checking expiry before insertion to Sales table. If the expiry date is earlier than the current date, insertion is blocked. It is critical for customer safety and legal compliance.

Trigger 3: TR_Sales_Insert

It creates an audit log for every sale recorded in Sales table by copying sale information to SalesLog after INSERT on Sales. This enables traceability for business audits or error tracking.

Trigger 4: TR_Sales_Update

It operates in a similar way as the previous trigger but is triggered by update operations on sales records. It provides historical changes to detect tampering or corrections.

Trigger 5: TR_Sales_Delete

This logs and safely deletes sales records. Before any sale record is deleted, this trigger copies its data into the SalesLog table to ensure the transaction is not lost. This ensures deletions are tracked and it preserves historical integrity.

5. Functions and Stored Procedure

5.1. Function: FN_CheckAvailability

It returns the available stock of a medicine by Med_ID. First it checks if the medicine exists, if so, it returns the quantity or returns 0 if it doesn't exist.

5.2. Stored Procedure: SP_AddSale

This is used to add sale and update stock if conditions are valid (if medicine exists, not expired and has enough stock). Then it inserts into Sales, updates Medicine stock and Last_Sold or triggers may fire (e.g., low stock alert).

To handle errors it uses TRY...CATCH which rolls back on failure.

6. Recovery Strategy

Sudden shutdowns could happen and lead to uncommitted data loss so to ensure data integrity the pharmacy database system consists of several recovery strategies to address potential failures. Some other common risks include database crashes due to system errors, power failures that may lead to data corruption and improper shutdowns which can interrupt active transactions or damage indexes.

To control these issues, regular backups are performed using SQL commands such as `BACKUP DATABASE TO DISK = 'C:\Backups\PharmacyDB.bak'` ensuring that recent data states can be restored in case of failure. Additionally, the system utilizes SQL transactions with `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK` statements to maintain consistency, especially during sensitive operations like refunds or deletions. This allows partial or faulty operations to be safely rolled back without affecting the overall system.

Another critical measure used is logging mechanism through the SalesLog table, which automatically records every insert, update, and delete action performed on the Sales table via dedicated triggers. This ensures a complete audit trail for tracing changes, identifying errors, and supporting recovery efforts if data needs to be reconstructed or verified.

7. Security Strategy

The security strategy for the pharmacy management system is designed to protect data integrity, and control over user actions. A login creation using the command `CREATE LOGIN pharmacy_user WITH PASSWORD = 'iAmThePharmacist!'`, which uses a strong password

containing uppercase, lowercase, and special characters to prevent easy compromise. This login is then bound to a database user with `CREATE USER pharmacy_user FOR LOGIN pharmacy_user;` effectively assigning a clear scope of access within the database.

Role-based access control is enforced by granting only essential permissions. For instance, `pharmacy_user` is allowed to perform `SELECT` and `INSERT` operations on the `Sales` table, `SELECT` on the `Medicine` table, and `EXECUTE` permissions on the stored procedure `SP_AddSale`. More sensitive actions such as `DELETE`, `UPDATE`, or schema changes like `DROP` are restricted. The `SP_AddSale` procedure encapsulates critical logic, helping validate inputs and reducing the risk of misuse or accidental damage.

This security approach ensures that users only have access to what is strictly necessary for their role.

8. Conclusion

This project aimed to design Pharmacy Management System that ensures reliable inventory and sales tracking. It successfully designed a normalized, efficient relational database that manages medicine stock, customer information, and transaction records including key features like automated low-stock alerts, prevention of expired medicine sales through triggers, and detailed logging of operations via audit trails. Additionally, backup and recovery strategies were included to handle potential system failures, while basic access control was applied to secure sensitive operations. Overall, this system addresses the common challenges found in manual pharmacy setups and demonstrates how SQL-based automation can improve efficiency, accuracy, and safety in pharmaceutical environments