**STORE MANAGEMENT SYSTEM USING SQL**

**PROJECT REPORT**

*Submitted by*

**SATVIK SHARMA (RA2211029010003)**

**S GAGAN (RA2211029010010)**

**ANITEJ MISHRA (RA2211029010023)**

*Under the guidance of*

**Dr P. Mahalakshmi**

**Assistant Professor, Department of Networking and Communications**

*In partial satisfaction of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**with specialization in Computer Networking**

****

**DEPARTMENT OF NETWORKING AND COMMUNICATIONS**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR-603 203**

**MAY 2024**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR-603 203**

**BONAFIDE CERTIFICATE**

Certified that this Project Report titled “**STORE MANAGEMENT SYSTEM USING SQL**” is the bonafide work done by:

**SATVIK SHARMA (RA2211029010003)**

**S GAGAN (RA2211029010010)**

**ANITEJ MISHRA (RA2211029010023)**

who completed the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

**SIGNATURE**

Dr P. Mahalakshmi

**DBMS-Course Faculty**

Assistant Professor

Department of Networking and Communications

**SIGNATURE**

Dr Annapurani Panaiyappan

**Head of the Department**

Department of Networking and Communications

SRMIST

SRMIST

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Ch. No.** | **CONTENT** | **PAGE NO.** |
|  | Abstract | 3 |
| 1. | Introduction | 4 |
| 2. | Literature Survey | 5 |
| 3. | Entity-Relationship Diagram | 6 |
| 4. | System Requirements | 12 |
| 5. | Use of Design Thinking Approach | 13 |
| 6. | List of Tables | 15 |
| 7. | Complex Queries | 23 |
| 8. | Pitfalls, Functional Dependencies and Normalization | 31 |
| 9. | Concurrency Control | 42 |
| 10. | API using Python | 46 |
|  | Conclusion | 48 |
|  | References | 49 |

**ABSTRACT**

As the world is continuously advancing and software to automate everything is available already. Stores are a very basic need of every citizen as they provide a variety of services like stationary, grocery, daily necessities etc. So, an efficient way to manage and run a general store is very important. Also, the paper bills are not very handy and are not reliable as well as they degrade overtime, stock calculations get unmanageable and hard to keep records of, the retailer also faces hardships on employee tracking. As a result, developing a Store Management System (hereafter simply referred to as SMS) to streamline the inventory tracking, sales recording, and customer management processes of a retail store is necessary, as it not only helps the store owner/manager but also increases the management efficiency of the store, and as a result the customer satisfaction increases, which in turn increases the stores popularity as well. Using an SMS has several direct and indirect advantages and resulting improvements. The system should enable store owners to efficiently manage product stock levels, record sales transactions, and maintain customer information. The goal is to enhance operational efficiency, improve customer satisfaction, and optimize inventory management processes within the retail store.

**Chapter 1**

**INTRODUCTION**

In today's fast-paced retail world, keeping a store running smoothly is super important. That's where a Store Management System (SMS) comes in handy. They're like high-tech toolkits designed to help stores manage everything from what they sell to how they treat customers. This project is all about creating one of these systems using SQL and Databases, focusing on making store management easier and more efficient.

At its heart, a Store Management System is like a big digital brain for a store. It's a bunch of software and databases that work together to handle all sorts of tasks, like keeping track of what's in stock, recording sales, managing staff, and even keeping customers happy.

The idea behind building this system is to tackle the tricky parts of running a store. By using SQL databases, we're aiming to build a solid foundation for storing and managing lots of data about the store. SQL is like a special language that helps us talk to databases, making it easier to find, change, and save information.

But this project isn't just about storing data; it's also about using it wisely. We're adding features to the system that help people make smart decisions based on real-time information. By adding tools like registers and bills management, we're giving store managers and owners the power to understand things like which products are selling best, how quickly items are flying off the shelves, and what customers are loving.

In the world of academics, this project is a chance to get hands-on with the stuff we've been learning about. It's a way to take all those theories and ideas and turn them into something practical and useful. By following a design thinking approach, we're not just building a system; we're solving problems. We're thinking about what store owners really need, how employees can work better, and how customers can have a smoother shopping experience.

In short, creating a Store Management System is all about mixing technology with practical solutions. It's about using our heads to make stores run better and make life easier for everyone involved. By combining a design thinking approach with database skills, we're aiming to make something that doesn't just look good on paper but works in the real world too.

**Chapter 2**

**LITERATURE SURVEY**

1. **“General Store Management System”**

Authors: Jatin Jangid, Sushma Khatri

Publication Year: 2022

Wereferred to this research paper to understand the methodology of store management and recreating it in our project using MySQL for developing an easy to use, efficient SMS.

1. **“Effective Use of Retail Store Management System for Small Retail Stores”**

Authors: Nirosha Wedasinghe and Devni Yasara

Publication Year: 2021

Wereferred to this research paper to understand the need of an effective SMS. The authors, based in Sri Lanka, have noted similar scenarios faced by store owners in Sri Lanka as well as India. We have used this to add and alter features to our SMS.

1. **“Stores Management System”**

Authors: A. Ganesan, S. Anupama, A. Benitsha

Publication Year: 2021

Wereferred to this research paper to learn how to make use of MySQL and database to make an SMS. It also helped us in areas like ER Diagrams and a possible implementation of a GUI.

1. **“Database System Concepts”**

Authors: Abraham Silberschatz, Henry F. Korth, S. Sudarshan

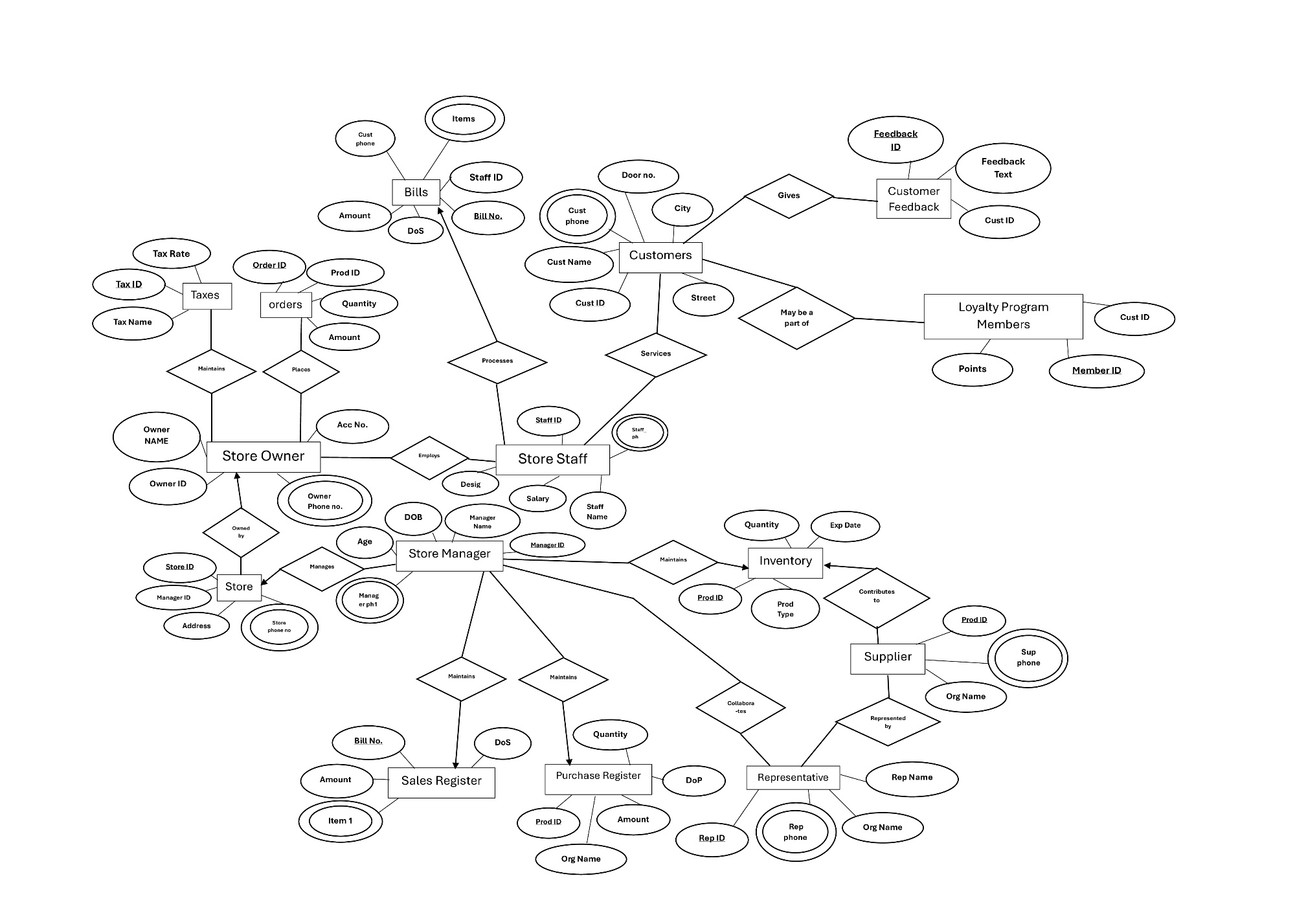
Edition: Sixth (Indian)

Publication Year: 2013

Publisher: McGraw Hill Education

We referred to this book to understand the basic concepts of Databases, their management and how to apply them in our project to make an SMS and also use them in our advantage.

**Chapter 3**

**ENTITY-RELATIONSHIP DIAGRAM**

**ENTITIES AND THEIR ATTRIBUTES**

1. **STORE**
   * Attributes
     + Store\_ID (Primary Key)
     + Address
     + Manager\_ID (Foreign Key)
     + Store\_Phone
   * This contains the details of the store itself, which comes in handy if the store has multiple branches or outlets.
2. **STORE OWNER**
   * Attributes
     + Owner\_Name
     + Owner\_ID (Primary Key)
     + Owner\_Phone
     + Acc\_No
   * They are the “central authority” of the store, who manage taxes and place the orders.
   * They employ the staff who help run the store.
3. **STORE MANAGER**

* Attributes
* Manager\_ID (Primary Key)
* Manager\_Name
* Manager\_Phone (Multi-Valued)
* DOB
* Age (Derived from DOB)
* They are the head of the store staff, who manage the store, inventory, and the registers.

1. **STORE STAFF**

* Attributes
  + Staff\_ID (Primary Key)
  + Staff\_Name
  + Designation
  + Salary
  + Staff\_Phone (Multi-Valued)
* They are the supporting employees who help run the store and perform essential tasks like helping the customers, running the cash register, making the bills, etc.

1. **SUPPLIER**

* Attributes
  + Org\_Name (Primary Key)
  + Sup\_Phone
  + Prod\_ID
* They are the organizations, companies and brands which provide the store with products to sell.
* Each supplier has a representative who stays connected with the store.

1. **REPRESENTATIVE**
   * Attributes
     + Rep\_Name
     + Rep\_Phone
     + Rep\_ID (Primary Key)
     + Org\_Name (Foreign Key)
   * As stated earlier, they are the representatives of the suppliers who collaborate and communicate with the store manager on the behalf of their organisation.
2. **CUSTOMER**

* Attributes
  + Customer\_Name
  + Customer\_ID (Primary Key)
  + Address (Composite)
    - Door No.
    - Street
    - City
  + Customer\_Phone
* The regular people who visit the store and buy various products from the store.
* It is the job of the Store Staff to help and service the customers in tasks like bills.
* Some customers who regularly shop at the stores can opt to become loyalty program members to entail special offers like discounts.
* Customers can also give their feedback of the store.

1. **LOYALTY PROGRAM MEMBERS**

* Attributes
  + Member\_ID (Primary Key)
  + Customer\_ID (Foreign Key)
  + Points
* These are regular customers at the store who can entail loyalty benefits.
* Their rewards and benefits are based on their accumulated points.

1. **CUSTOMER FEEDBACK**

* Attributes
  + Feedback\_ID (Primary Key)
  + Customer\_ID (Foreign Key)
  + Feedback\_Text
* This is used to record the feedback given by the customers which the store owners and managers can use to make constructive changes.

1. **INVENTORY**

* Attributes
  + Product\_ID (Primary Key)
  + Quantity
  + Product\_Type
  + Expiry\_Date
* The store manager keeps a track of the available products using the Inventory.
* It makes a note of all available products, their quantity and other important info like Date of Purchase, Expiry, etc.

1. **ORDERS**

* Attributes
  + Order\_ID (Primary Key)
  + Product\_ID (Foreign Key)
  + Quantity
  + Amount
* The store owner places orders from the suppliers to buy the goods that will be sold in their store.

1. **TAXES**

* Attributes
  + Tax\_ID (Primary Key)
  + Tax\_Rate
  + Tax\_Name
* The store owners must pay taxes on the transactions related to the store, and this entity stores a simplified version of that.

1. **BILLS**

* Attributes
  + Bill\_No (Primary Key)
  + Customer\_Phone
  + Discount
  + Amount
  + Date\_of\_Sale
  + Items (Multivalued)
* These are the invoices made by the store staff and given to the customers making a note of their purchases.
* It helps the store owner in keeping a track of what was sold to whom.

1. **PURCHASE REGISTER**

* Attributes
  + Org\_Name
  + Quantity
  + Amount
  + Date\_of\_Purchase
  + Prod\_ID (Primary Key)
* They are a record of “bills” for the store owner to keep a track of what they ordered from the supplier to be sold in the store.

1. **SALES REGISTER**

* Attributes
  + Bill\_no (Primary Key)
  + Items (Multivalued)
  + Amount
  + Date\_of\_sale
* These are a record of what was sold by the store to the customers in each transaction.

**UNDERSTANDING THE ENTITY-RELATIONSHIP DIAGRAM**

The Entity-Relationship (ER) model we have developed for the store management system provides a comprehensive overview of the key entities and their relationships within the system. At its core, the system revolves around the Store Owner/Manager, who acts as the central authority responsible for managing various aspects of the store, including inventory, staff, suppliers, and customer transactions.

The entities such as Store Staff, Supplier, Customer, Inventory, Bills, Purchase Register, and Sales Register encapsulate the essential components of the store's operations. Each entity plays a specific role in the system, contributing to the overall functioning and organization of the store. For instance, the Store Staff entity represents the employees responsible for assisting customers, processing transactions, and ensuring smooth day-to-day operations. Meanwhile, the Supplier entity reflects the external entities that provide products to the store, while the Inventory entity tracks the availability and details of products within the store's stock.

The relationships established between these entities further define the interactions and dependencies within the system. For example, the relationship between the Store Manager and Store Staff signifies the employment hierarchy, where the manager oversees and supervises the staff members. Similarly, the relationships between the Store Manager and entities like Purchase Register and Sales Register highlight the managerial oversight of procurement and sales activities. Overall, the ER model offers a structured representation of the store management system, facilitating effective understanding and implementation of its functionalities.

**Chapter 4**

**SYSTEM REQUIREMENTS**

1. **OPERATING SYSTEM**

The SMS can be used on various operating systems, including Windows, macOS, and Linux. We can choose the one that we are most comfortable with. We recommend using Windows 10 or Windows 11.

1. **DEVELOPMENT ENVIRONMENT**

We can use a variety of databases for creating a SMS, like MySQL, Oracle Database, etc. For our project we have chosen MySQL.

1. **HARDWARE**

We don't need a high-end computer for this SMS. A basic desktop or laptop with at least 4GB of RAM and a modern multi-core processor should suffice.

1. **GRAPHICS**

The SMS is not a very graphics-demanding system, so we don't need a powerful graphics card. Integrated graphics on most modern computers will be more than enough.

1. **STORAGE**

We don't need much storage space for code and assets. A few gigabytes should be sufficient.

1. **REPORTS**

To create reports and store data, we can use Microsoft Excel spreadsheets (.xlsx) and CSV files (Comma Separated Values, .csv).

**Chapter 5**

**USE OF DESIGN THINKING APPROACH**

1. **DESIGN THE PROBLEM and EMPATHISE**

* Many store owners still use pen and paper, or basic operating system files to manage stock, bills, employees, sales, and purchase records. It harms their business as it is inefficient, slow, and tedious to maintain.
* Making a SMS is crucial, as it will help the business, the store owner, and the customers too indirectly in the long run.

1. **RESEARCH, IDEATION and DEFINE**

* When comparing similar businesses, some of which use SMS, and others which don’t, the businesses using SMS are “infrastructurally” better, efficient, and faster for both the owner and customers.
* Store owners not using SMS remarked that their business is slowing down and they’re losing customers, compared to the ones using SMS who are gaining customers. Again, implementation of an SMS is important.

1. **PROTOTYPING**

* To make an SMS for this problem, we need to identify the stakeholders first.
* Stakeholders include the owner/manager, suppliers, customers, and the store staff.
* We must identify how the current situation affects these stakeholders (an ER diagram will be useful in this case), and how implementing an SMS will positively impact them, then we can make a basic SMS to test it out.

1. **USER FEEDBACK**

* Once the SMS is implemented for the first time, we can note the owner’s remarks on how it makes his tasks easier and quicker, like inventory and staff management.
* In the long run, it can be seen how the business has been positively affected.
* Based on the owner’s feedback, the SMS can be simplified and improved to better fit the owner’s capabilities.

1. **IMPLEMENTATION**

* We must identify the best approach to make the SMS and its databases. Again, using ER Diagrams and Databases schemas can help. We have picked MySQL for our project.

1. **TESTING**

* Once the project is made, it must be tested in all possible cases and scenarios for debugging and improvements. Getting preliminary beta feedback for users and building on that is also helpful.

1. **DOCUMENTATION**

* Creating meaningful Reports, Presentations and README files to help users understand the SMS is crucial. Without understanding how something works, a user cannot obviously use the system properly.

1. **REFLECTION and ITERATION**

* Once again, gather feedback and iterate through possible cases to identify areas of improvement or errors. Adjust the system accordingly.

1. **FINAL PRESENTATION**

* To make this, we must reflect on every step that has come before this. We must highlight key features and designs in our final PPT.

**Chapter 6**

**LIST OF TABLES**

1. **STORE**

Schema: Store(Store\_ID, Address, Manager\_ID, Store\_phone\_no)

Query

CREATE TABLE Store (

Store\_ID INT PRIMARY KEY,

Store\_number INT,

Street VARCHAR(255),

City VARCHAR(255),

Manager\_ID INT,

Store\_phone\_no BIGINT CHECK (LENGTH(CAST(Store\_phone\_no AS CHAR)) = 10),

FOREIGN KEY (Manager\_ID) REFERENCES Store\_Manager(Manager\_ID)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Store\_ID | int | NO | PRI | NULL |  |
| Store\_number | int | YES |  | NULL |  |
| Street | varchar(255) | YES |  | NULL |  |
| City | varchar(255) | YES |  | NULL |  |
| Manager\_ID | int | YES |  | NULL |  |
| Store\_phone\_no | bigint | YES |  | NULL |  |

1. **STORE OWNER**

Schema: Store Owner(Owner\_name, Owner\_ID, Owner\_phone, Acc\_no)

Query

CREATE TABLE Store\_Owner (

Owner\_ID INT PRIMARY KEY,

Owner\_name VARCHAR(255),

Owner\_ph BIGINT CHECK (LENGTH(CAST(Owner\_ph AS CHAR)) = 10),

Acc\_no VARCHAR(255)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Owner\_ID | int | NO | PRI | NULL |  |
| Owner\_name | varchar(255) | YES |  | NULL |  |
| Owner\_ph | bigint | YES |  | NULL |  |
| Acc\_no | varchar(255) | YES |  | NULL |  |

1. **STORE MANAGER**

Schema: Store Manager(Manager\_Name, Manager\_ID, DOB, Age, Manager\_ph1, Manager\_ph2)

Query

CREATE TABLE Store\_Manager (

Manager\_ID INT PRIMARY KEY,

Manager\_name VARCHAR(255),

DOB DATE,

Age INT,

Manager\_ph1 BIGINT CHECK (LENGTH(CAST(Manager\_ph1 AS CHAR)) = 10),

Manager\_ph2 BIGINT CHECK (LENGTH(CAST(Manager\_ph2 AS CHAR)) = 10)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Manager\_ID | int | NO | PRI | NULL |  |
| Manager\_name | varchar(255) | YES |  | NULL |  |
| DOB | date | YES |  | NULL |  |
| Age | int | YES |  | NULL |  |
| Manager\_ph1 | bigint | YES |  | NULL |  |
| Manager\_ph2 | bigint | YES |  | NULL |  |

1. **STORE STAFF**

Schema: Staff(Staff\_ID, Staff\_name, Staff\_ph1, Staff\_ph2, Designation, Salary)

Query

CREATE TABLE Staff (

Staff\_ID INT PRIMARY KEY,

Staff\_name VARCHAR(255),

Staff\_ph1 BIGINT CHECK (LENGTH(Staff\_ph1) = 10),

Staff\_ph2 BIGINT CHECK (LENGTH(Staff\_ph2) = 10),

Designation VARCHAR(255),

Salary INT

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Staff\_ID | int | NO | PRI | NULL |  |
| Staff\_name | varchar(255) | YES |  | NULL |  |
| Staff\_ph1 | bigint | YES |  | NULL |  |
| Staff\_ph2 | bigint | YES |  | NULL |  |
| Designation | varchar(255) | YES |  | NULL |  |
| Salary | int | YES |  | NULL |  |

1. **SUPPLIER**

Schema: Supplier(Org\_name, Prod\_ID, Sup\_Phone)

Query

CREATE TABLE Supplier (

Org\_name VARCHAR(255) PRIMARY KEY,

Prod\_ID INT,

Sup\_Phone BIGINT CHECK (LENGTH(CAST(Sup\_Phone AS CHAR)) = 10)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Org\_name | varchar(255) | NO | PRI | NULL |  |
| Prod\_ID | int | YES | MUL | NULL |  |
| Sup\_Ph | bigint | YES |  | NULL |  |

1. **REPRESENTATIVE**

Schema: Representative(Rep\_Name, Rep\_phone, Rep\_ID, Org\_name)

Query

CREATE TABLE Representative (

Rep\_ID INT PRIMARY KEY,

Rep\_name VARCHAR(255),

Rep\_phone BIGINT CHECK (LENGTH(CAST(Rep\_phone AS CHAR)) = 10),

Org\_name VARCHAR(255),

FOREIGN KEY (Org\_name) REFERENCES Supplier(Org\_name)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Rep\_ID | int | NO | PRI | NULL |  |
| Rep\_name | varchar(255) | YES |  | NULL |  |
| Rep\_phone | bigint | YES |  | NULL |  |
| Org\_name | varchar(255) | YES |  | NULL |  |

1. **CUSTOMER**

Schema: Customer(Cust\_ID, Cust\_name, Door\_no, Street, City, Cust\_Ph)

Query

CREATE TABLE Customer (

Cust\_ID INT PRIMARY KEY,

Cust\_name VARCHAR(255),

Door\_no INT,

Street VARCHAR(255),

City VARCHAR(255),

Cust\_Phone BIGINT CHECK (LENGTH(Cust\_Ph) = 10)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Cust\_ID | int | NO | PRI | NULL |  |
| Cust\_name | varchar(255) | YES |  | NULL |  |
| Door\_no | int | YES |  | NULL |  |
| Street | varchar(255) | YES |  | NULL |  |
| City | varchar(255) | YES |  | NULL |  |
| Cust\_Phone | bigint | YES |  | NULL |  |

1. **LOYALTY PROGRAM MEMBERS**

Schema: Loyalty\_Members(Member\_ID, Cust\_ID, Points)

Query

CREATE TABLE Loyalty\_members (

Member\_ID INT PRIMARY KEY,

Cust\_ID INT,

Points INT,

FOREIGN KEY (Cust\_ID) REFERENCES Customer(Cust\_ID)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Member\_ID | int | NO | PRI | NULL |  |
| Cust\_ID | int | YES |  | NULL |  |
| Points | int | YES |  | NULL |  |

1. **CUSTOMER FEEDBACK**

Schema: Customer Feedback(Feedback\_ID, Cust\_ID, Feedback text)

Query

CREATE TABLE Customer\_feedback (

Feedback\_ID INT,

Cust\_ID INT,

Feedback\_text VARCHAR(255),

PRIMARY KEY (Feedback\_ID, Cust\_ID),

FOREIGN KEY (Cust\_ID) REFERENCES Customer(Cust\_ID)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Feedback\_ID | int | NO | PRI | NULL |  |
| Cust\_ID | int | YES |  | NULL |  |
| Feedback\_text | varchar(255) | YES |  | NULL |  |

1. **INVENTORY**

Schema: Inventory(Prod\_ID, Prod\_type, Quantity, Exp\_date)

Query

CREATE TABLE Inventory (

Prod\_ID INT PRIMARY KEY,

Prod\_type VARCHAR(255),

Quantity INT,

Exp\_date DATE

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Prod\_ID | int | NO | PRI | NULL |  |
| Prod\_type | varchar(255) | YES |  | NULL |  |
| Quantity | int | YES |  | NULL |  |
| Exp\_date | date | YES |  | NULL |  |

1. **ORDERS**

Schema: Orders(Order\_ID, Prod\_ID, Quantity, Amount)

Query

CREATE TABLE Orders (

Order\_ID INT PRIMARY KEY,

Prod\_ID INT,

Quantity INT,

Amount INT,

FOREIGN KEY (Prod\_ID) REFERENCES Inventory(Prod\_ID)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Order\_ID | int | NO | PRI | NULL |  |
| Prod\_ID | int | YES |  | NULL |  |
| Quantity | int | YES |  | NULL |  |
| Amount | int | YES |  | NULL |  |

1. **TAXES**

Schema: Taxes(Tax ID, Tax Rate, Tax Name)

Query

CREATE TABLE Taxes (

Tax\_ID INT PRIMARY KEY,

Tax\_rate INT,

Tax\_name VARCHAR(255)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Tax\_ID | int | NO | PRI | NULL |  |
| Tax\_rate | int | YES |  | NULL |  |
| Tax\_name | varchar(255) | YES |  | NULL |  |

1. **BILLS**

Schema: Bills(Bill\_no, Staff\_ID, Item1, Item2, Item3, Item4, Item5, Amount, DoS, Cust\_ph, Discount)

Query

CREATE TABLE Bills (

Bill\_no INT PRIMARY KEY,

Staff\_ID INT,

Item1 VARCHAR(255),

Item2 VARCHAR(255),

Item3 VARCHAR(255),

Item4 VARCHAR(255),

Item5 VARCHAR(255),

Amount INT,

DoS DATE,

Cust\_ph BIGINT CHECK (LENGTH(Cust\_ph) = 10),

Discount INT,

FOREIGN KEY (Staff\_ID) REFERENCES Staff(Staff\_ID)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Bill\_no | int | NO | PRI | NULL |  |
| Staff\_ID | int | YES |  | NULL |  |
| Item1 | varchar(255) | YES |  | NULL |  |
| Item2 | varchar(255) | YES |  | NULL |  |
| Item3 | varchar(255) | YES |  | NULL |  |
| Item4 | varchar(255) | YES |  | NULL |  |
| Item5 | varchar(255) | YES |  | NULL |  |
| Amount | int | YES |  | NULL |  |
| DoS | date | YES |  | NULL |  |
| Cust\_ph | bigint | YES |  | NULL |  |
| Discount | int | YES |  | NULL |  |

1. **PURCHASE REGISTER**

Schema: Purchase\_register(Prod\_ID, Org\_name, Quantity, Amount, DoP)

Query

CREATE TABLE PurchaseRegister (

Prod\_ID INT,

Org\_name VARCHAR(255),

Quantity INT,

Amount INT,

DoP DATE,

FOREIGN KEY (Prod\_ID) REFERENCES Inventory(Prod\_ID),

FOREIGN KEY (Org\_name) REFERENCES Supplier(Org\_name)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Prod\_ID | int | YES | PRI | NULL |  |
| Org\_name | varchar(255) | YES |  | NULL |  |
| Quantity | int | YES |  | NULL |  |
| Amount | int | YES |  | NULL |  |
| DoP | date | YES |  | NULL |  |

1. **SALES REGISTER**

Schema: Sales\_register(Bill\_no, Item1, Item2, Item3, Item4, Item5, Amount, DoS)

Query

CREATE TABLE SalesRegister (

Bill\_no INT PRIMARY KEY,

Item1 VARCHAR(255),

Item2 VARCHAR(255),

Item3 VARCHAR(255),

Item4 VARCHAR(255),

Item5 VARCHAR(255),

Amount INT,

DoS DATE,

FOREIGN KEY (Bill\_no) REFERENCES Bills(Bill\_no)

);

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| Bill\_no | int | NO | PRI | NULL |  |
| Item1 | varchar(255) | YES |  | NULL |  |
| Item2 | varchar(255) | YES |  | NULL |  |
| Item3 | varchar(255) | YES |  | NULL |  |
| Item4 | varchar(255) | YES |  | NULL |  |
| Item5 | varchar(255) | YES |  | NULL |  |
| Amount | int | YES |  | NULL |  |
| DoS | date | YES |  | NULL |  |

**Chapter 7**

**COMPLEX QUERIES**

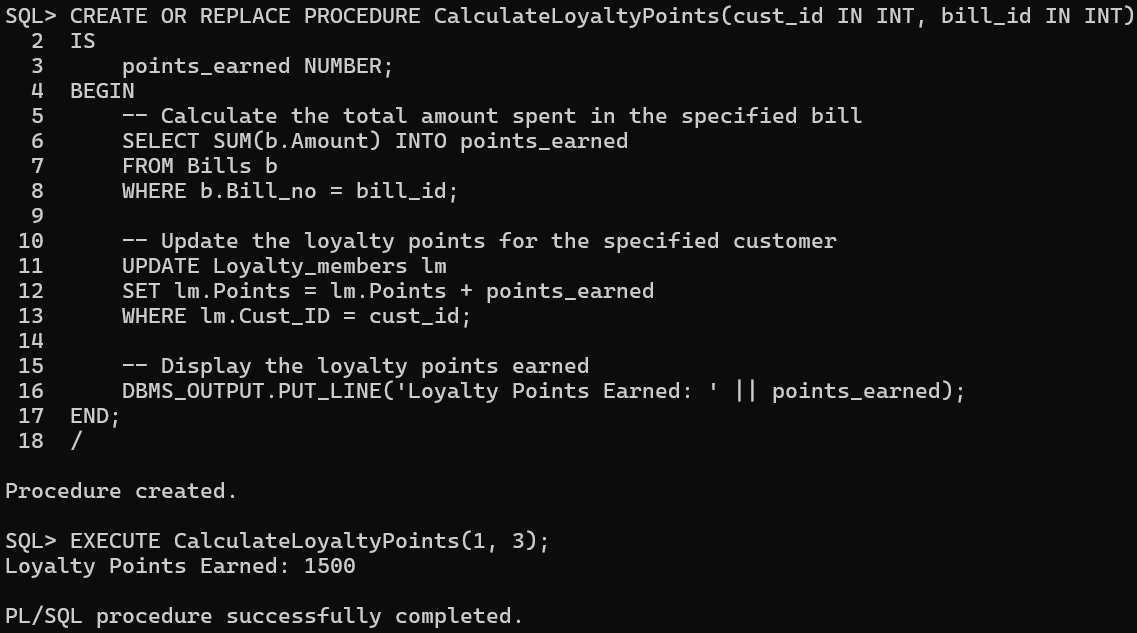
* **PL/SQL**

1. Calculate Total Sales for a given Customer

A screenshot of a computer program

Description automatically generated

1. Update Loyalty Points for a Customer after a Purchase



1. Generate a Report of Top Loyal Customers

A computer screen shot of a black screen

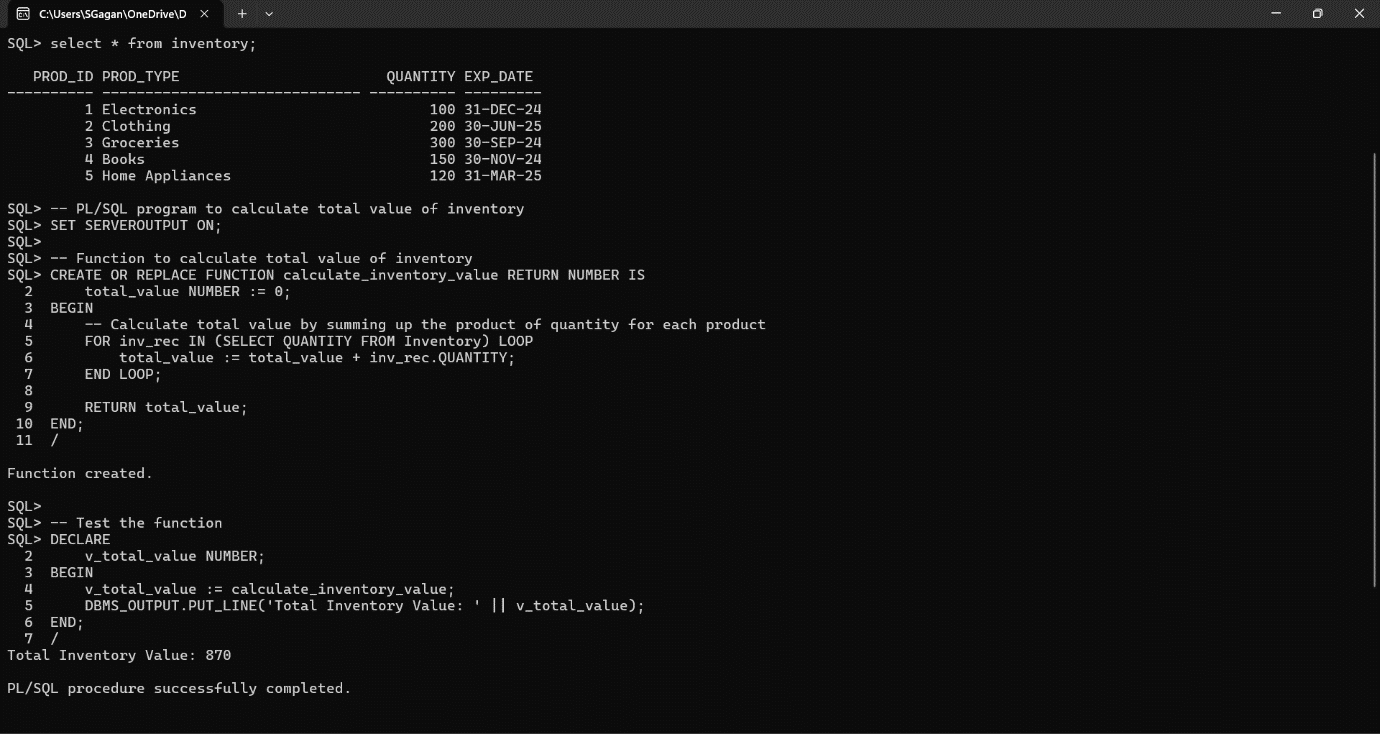
Description automatically generated

1. Update the Phone Number of a Store Manager

A screenshot of a computer

Description automatically generated

1. Fetching the Total Quantity of the Inventory



* **Views**

1. Loyalty MembersA screenshot of a computer

   Description automatically generated
2. Purchase RegisterA screenshot of a computer

   Description automatically generated
3. Store Revenue Summary

A screenshot of a computer screen

Description automatically generated

1. Store Information with City and Manager

A screenshot of a computer

Description automatically generated

1. Bills

A screenshot of a computer

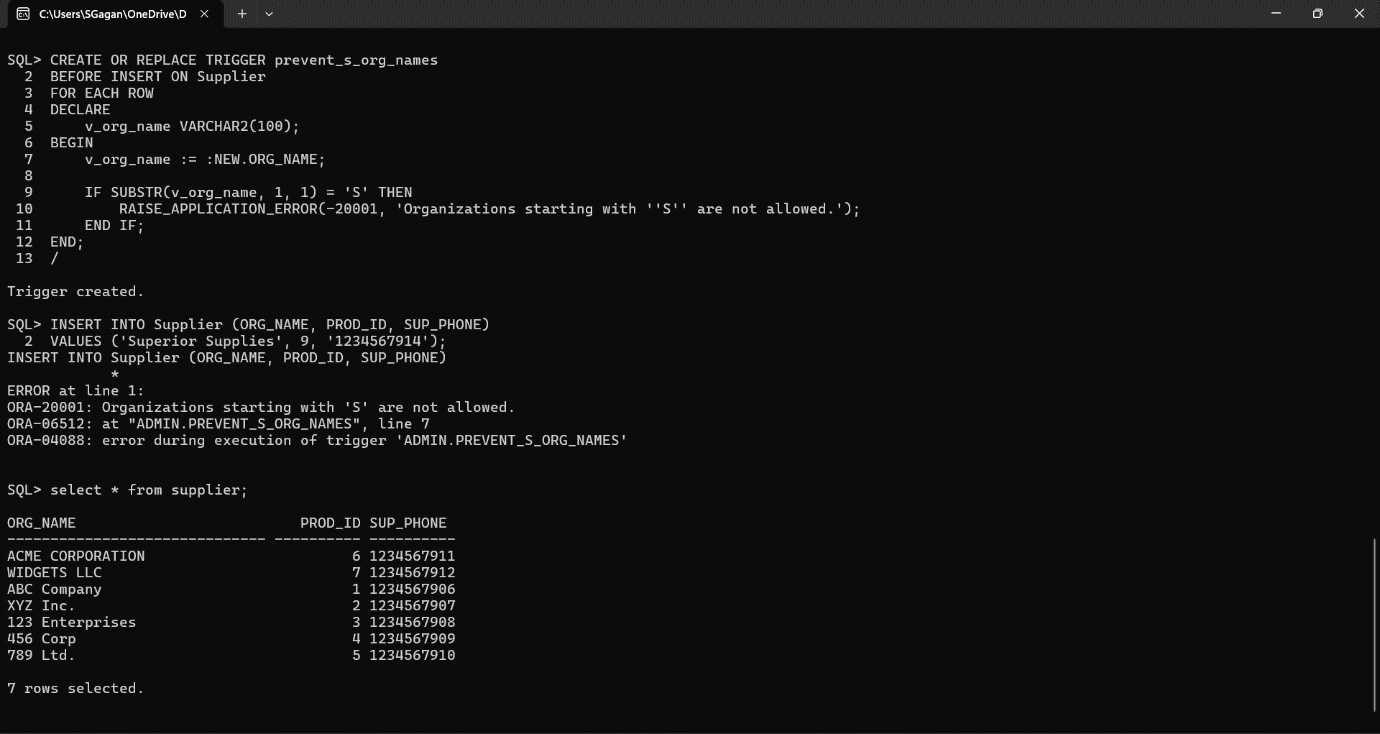
Description automatically generated

* **Triggers**

1. Capturing and Displaying Customer FeedbackA screenshot of a computer

   Description automatically generated
2. Updating Inventory after a Purchase or a SaleA screenshot of a computer

   Description automatically generated
3. Trigger for Incrementing Manager’s AgeA screenshot of a computer

   Description automatically generated
4. Trigger for not allowing Insertion of Organizations whose names start with ‘S’
5. Trigger for Limiting Customer Feedback Size to 25 Characters

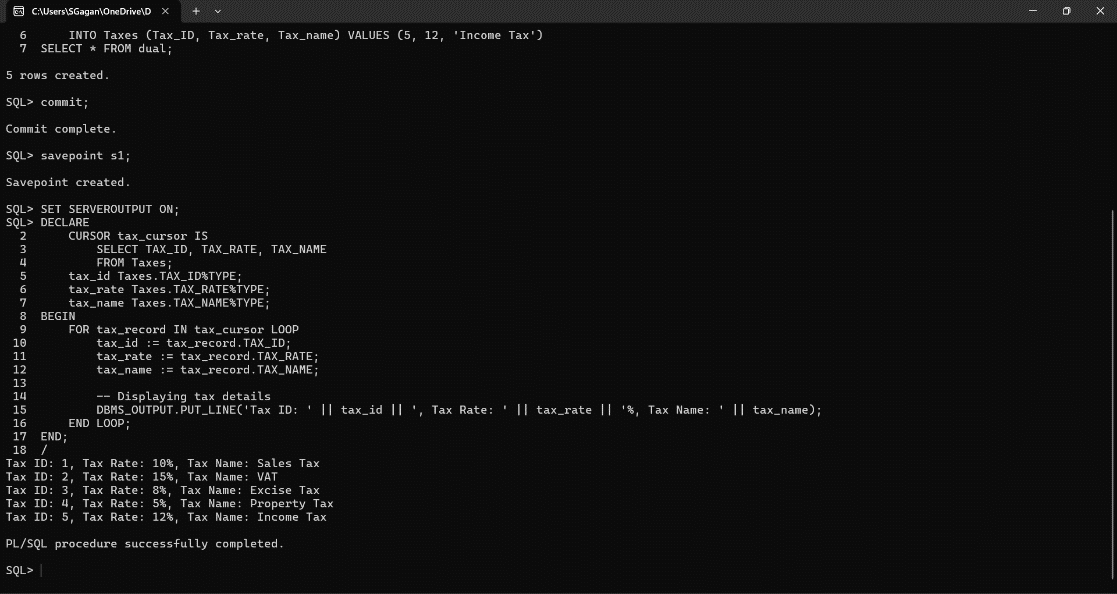


* **Cursors**

1. Generating a Report of Bills with Customer DetailsA screenshot of a computer program

   Description automatically generated
2. Retrieving Store Information with the Details of the Store ManagerA screenshot of a computer

   Description automatically generated
3. Creating an Orders View with Amount=Quantity\*10A screenshot of a computer

   Description automatically generated
4. Fetching Details of Taxes

**Chapter 8**

**PITFALLS, FUNCTIONAL DEPENDENCIES AND NORMALIZATION**

The 4 main types of Pitfalls in Relational Database Design and how they may occur in our SMS are given below:

1. **REDUNDANCY**

* The "Customer" table stores customer information such as name, address, and phone number. Redundancy might occur if the same customer information is stored in multiple tables or if there are redundant columns within a table.
* In the "Bills" table, the columns "Item1" through "Item5" may lead to redundancy if there are instances where not all items are used in a bill.

1. **INCONSISTENCY**

* Inconsistencies might arise if different parts of the database hold different versions of the same data. For example, if a customer's address is updated in one table but not in another, inconsistencies can occur.
* The "Inventory" table holds information about products, including their quantity and expiration date. Inconsistencies might occur if the quantity of a product in the "Inventory" table does not match the quantity of the same product in the "Orders" or "Sales\_Register" tables.

1. **INEFFICIENCY**

* Inefficiencies can arise due to poor database design leading to slower query performance and increased storage requirements.
* For example, having multiple columns for items in the "Bills" table might lead to inefficient queries, especially if the number of items varies greatly from one bill to another.

1. **COMPLEXITY**

* A complex database schema can be difficult to understand and maintain, leading to errors and inefficiencies.
* The schema includes multiple tables with various relationships, which might become challenging to manage as the database grows in size and complexity.

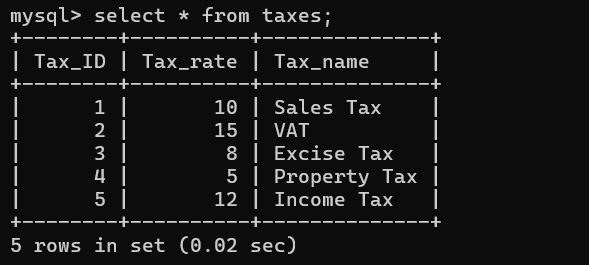
To mitigate these pitfalls, we have considered the following solutions:

* Normalizing our database schema to reduce redundancy and ensure data consistency.
* Using foreign key constraints to maintain referential integrity and prevent inconsistencies.
* Optimizing our schema for better query performance by avoiding unnecessary denormalization and ensuring appropriate indexing.
* Documenting our database schema and relationships to aid in understanding and maintenance.

By addressing these potential pitfalls, we can create a more robust and efficient Store Management System database.

To do so, we have normalized our tables as follows:

1. Taxes



Functional Dependencies:

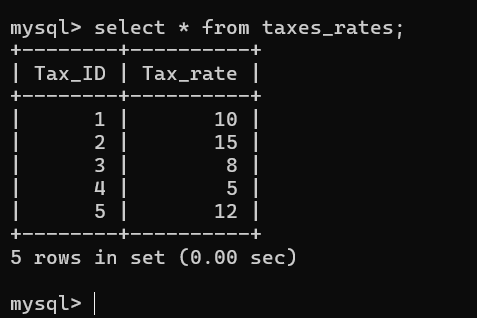
* + - Tax\_id → Tax\_rate
    - Tax\_id → Tax\_name

There is a transitive dependency in Tax\_id → Tax\_name:

* + - Tax\_id is not super key
    - Tax\_name is not prime

We can apply 3NF and decompose the above table into ‘Taxes\_names’ and ‘Taxes\_rates’.





1. Customer

**A screenshot of a computer screen

Description automatically generated**

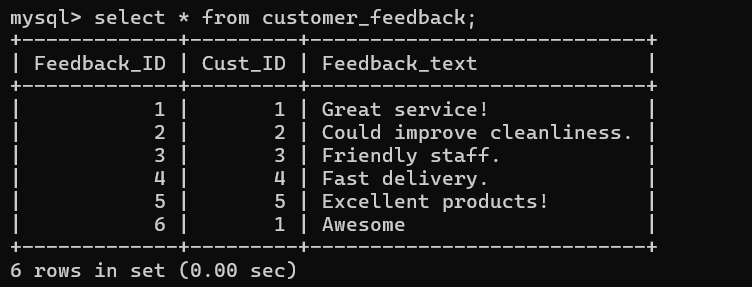
Functional Dependency

* + - Cust\_ID → Cust\_name, Door\_no, Street, City, Cust\_Phone

There is no partial, transitive, multi-valued or join dependencies.

* + - Cust\_id is a Super Key, hence it is a Primary key
    - The other attributes are fully functionally dependent on Cust\_ID

1. Customer\_Feedback



Functional Dependencies:

* + - Feedback\_id → Cust\_ID
    - Feedback\_id → feedback\_id

There is no partial, transitive, multi-valued or join dependencies.

* + - Feedback\_id is super key (hence primary key)
    - The other attributes are fully functionally dependent on Feedback\_id

1. Loyalty\_Members

**A screen shot of a computer code

Description automatically generated**

Functional Dependencies:

* + - Member\_ID → Cust\_ID, Points

There is no partial, transitive, multi-valued or join dependencies.

* + - Member\_ID is a Super Key, hence it is a Primary Key
    - The other attributes are fully functionally dependent on Member\_ID.

1. Store\_Manager

A screenshot of a computer screen

Description automatically generated

Functional Dependencies:

* + - Manager\_ID → Manager\_name, DOB, Age, Manager\_ph1, Manager\_ph2

There is no partial, transitive, multi-valued or join dependencies.

* + - Manager\_ID is a Super Key, hence it is a Primary Key
    - The other attributes are fully functionally dependent on Manager\_ID.

1. Store\_Owner

A screen shot of a computer

Description automatically generated

Functional Dependencies:

* + - Owner\_ID → Owner\_name, Owner\_ph, Acc\_no

There is no partial, transitive, multi-valued or join dependencies.

* + - Owner\_ID is a Super Key, hence it is a Primary Key
    - The other attributes are fully functionally dependent on Owner\_ID.

1. Store

A screenshot of a black screen

Description automatically generated

Functional Dependencies:

* + - Store\_ID → Store\_number, Street, City, Manager\_ID, Store\_phone\_no
    - Manager\_ID → Manager\_name, DOB, Age, Manager\_ph1, Manager\_ph2

This table needs normalization to remove partial dependencies. We can apply 2NF and decompose the above table into ‘Store\_Address’ and ‘Store1’.

A black and white screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

1. Store\_Staff

A screenshot of a computer screen

Description automatically generated

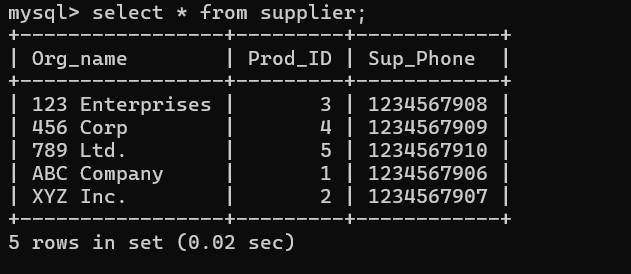
Functional Dependencies:

* + - Staff\_ID → Staff\_name, Staff\_ph1, Staff\_ph2, Designation, Salary

There is no partial, transitive, multi-valued or join dependencies.

* + - Staff\_ID is a Super Key, hence it is a Primary Key
    - The other attributes are fully functionally dependent on Staff\_ID.

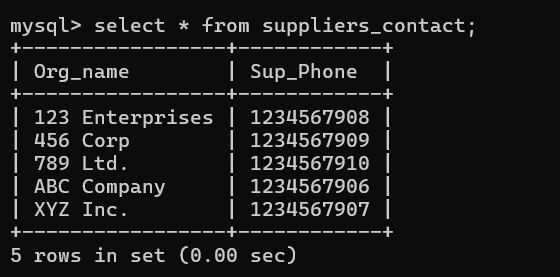
1. Supplier

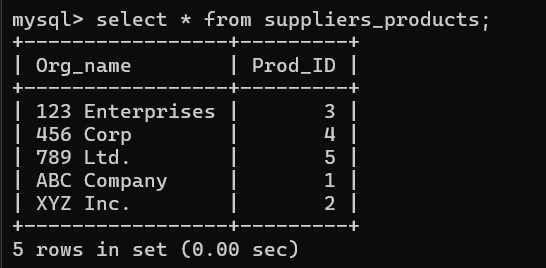


Functional dependencies

* + - Org\_name → Prod\_ID
    - Org\_name → Sup\_phone

There is a partial dependency between Org\_name and Prod\_ID. So, we can normalise it using 2NF form.





1. Representative

A screenshot of a computer

Description automatically generated

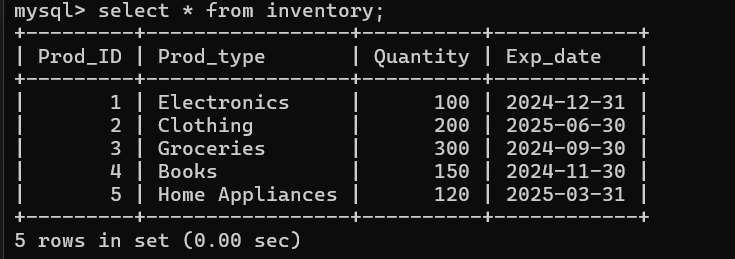
Functional Dependencies:

* + - Rep\_ID → Rep\_name, Rep\_phone, Org\_name

There is no partial, transitive, multi-valued or join dependencies.

* + - Rep\_ID is a Super Key, hence it is a Primary Key
    - The other attributes are fully functionally dependent on Rep\_ID.

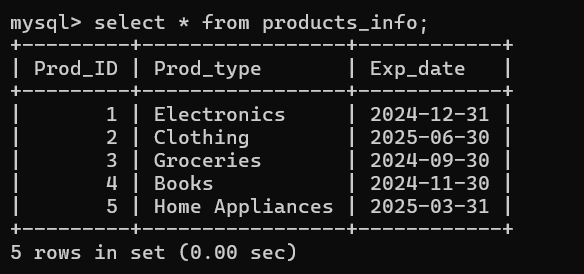
1. Inventory



Functional dependencies:

* + - Prod\_id → prod\_type
    - Prod\_id → quantity
    - Prod\_id → exp\_date

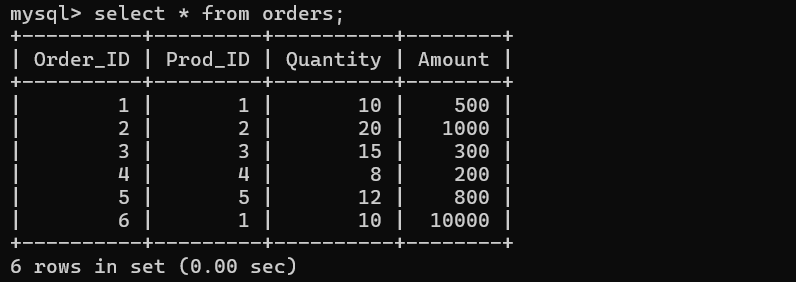
There is transitive dependency b/w Prod\_id and the non-prime attributes as Prod\_id is not super-key either.  
This can be normalised by applying 3NF



A screen shot of a computer

Description automatically generated

1. Orders



Functional Dependencies:

* + - Order\_id → Prod\_id
    - Order\_id → Quantity
    - Order\_id → Amount

There is no partial, transitive, multi-valued or join dependencies.

* + - Order\_id is super key (hence primary key)
    - The other attributes are fully functionally dependent on Order\_id

1. Bills

**A screen shot of a computer

Description automatically generated**

Functional Dependency:

* + - Bill\_no → Staff\_ID, Item1, Item2, Item3, Item4, Item5, Amount, DoS, Cust\_ph, Discount

This table has repeating groups.

This can be normalized by using 1NF.

A screenshot of a computer screen

Description automatically generated

1. Purchase\_Register

**A screen shot of a black screen

Description automatically generated**

Functional Dependencies:

* + - Prod\_ID → Org\_name
    - Prod\_ID → Quantity
    - Prod\_ID → Amount
    - Prod\_ID → DoP

This table needs normalization to remove partial dependencies. This can be done by using 2NF.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. Sales\_Register

**A screen shot of a computer

Description automatically generated**

Functional Dependencies:

* + - Bill\_no → Item1, Item2, Item3, Item4, Item5, Amount, DoS

This table needs normalization to remove repeating groups. This can be normalized by using 1NF.

A black screen with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

**Chapter 9**

**CONCURRENCY CONTROL**

Transactions in a database are of two types mainly:

* **Concurrent Transactions:** In a concurrent transaction schedule, multiple transactions can execute simultaneously. This allows for better utilization of system resources and can improve overall system throughput. However, concurrency introduces the possibility of interference between transactions, leading to issues such as lost updates, uncommitted data, and inconsistent reads.
* **Serial Transactions:** In a serial transaction schedule, transactions are executed one after the other in a sequential manner. Each transaction completes its execution before the next one begins. This ensures that transactions are isolated from each other, and their effects are visible to other transactions only after they have been committed.

In our project, we have decided to use mainly Serial Transaction scheduling more than Concurrent since data consistency is a major part of our SMS.

Some transactions in SMS are given hereafter:

**CONCURRENT TRANSACTIONS**

1. Update the tax rate for Sales Tax from 10 to 9, and update the tax rate for VAT from 15 to 14 in “taxes” table

A screenshot of a computer screen

Description automatically generated A screenshot of a computer screen

Description automatically generated

1. Update the phone number for John Doe to '9876543210', and update the phone number for Jane Smith to '9876543211' in “customer” table

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Update the discount for Bill 3 from 0 to 5, and update the discount for Bill 4 from 25 to 20 in “bills” table

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**SERIAL TRANSACTIONS**

1. Update the tax rate for Sales Tax from 10 to 9, and update the tax rate for VAT from 15 to 14 in “taxes” table

A screenshot of a computer screen

Description automatically generated

1. Update the phone number for John Doe to '9876543210', and update the phone number for Jane Smith to '9876543211' in “customer” table

A screenshot of a computer

Description automatically generated

1. Update the discount for Bill 3 from 0 to 5, and update the discount for Bill 4 from 25 to 20 in “bills” table

A screen shot of a computer

Description automatically generated

**Chapter 10**

**API USING PYTHON**

To make our SMS project easy to use, even for those who don’t have much knowledge about using computers, we have created a Python application, which connects to the SMS database.

It is a rudimentary approach to front-end and back-end application development, so currently it does only basic operations like adding values to and viewing particular tables.

Through Python’s Tkinter Library, it uses GUI elements like interactive buttons, text boxes and others to make interacting with our SMS database very simple and straightforward.

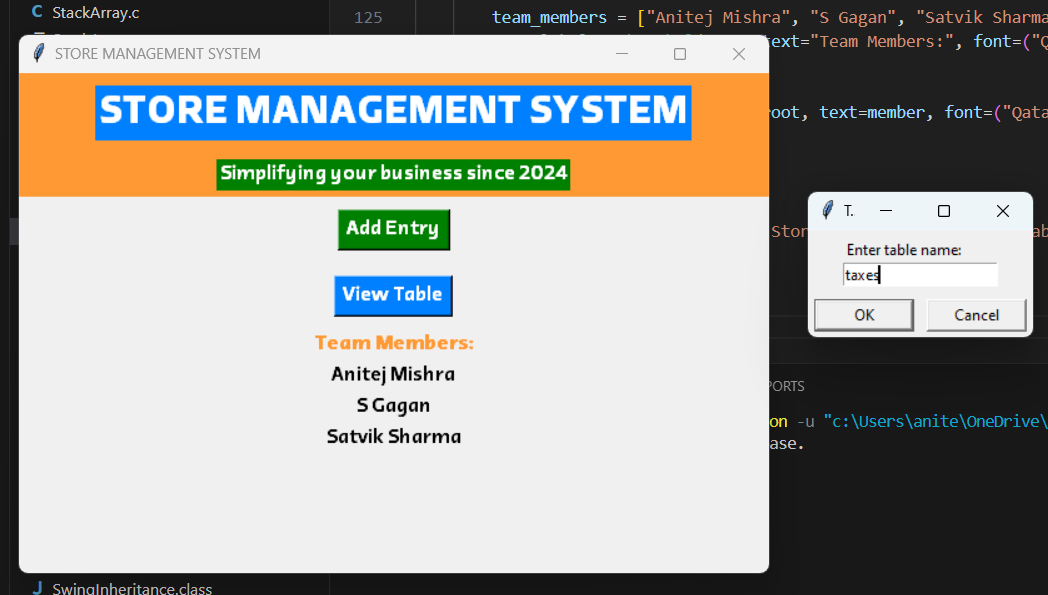
The screenshots of our Python application are given below:

* Home Screen

A screenshot of a computer

Description automatically generated

* Add Entry
  + Entering the table name…



* + Adding the values, with before and after… (Output in the command line client)

A screenshot of a computer

Description automatically generated

* View Table
  + Entering the table name…

A screenshot of a computer

Description automatically generated

* + The output…

A screenshot of a computer

Description automatically generated

**CONCLUSION**

Our simple Store Management System (SMS) leveraging SQL offers a robust solution for efficient store operations. Our SMS makes it easier for store owners to manage their daily operations easily and securely. By utilizing SQL's relational database management capabilities, the system effectively organizes and stores crucial data such as inventory, sales, and customer information. Through seamless integration with SQL, the SMS ensures data integrity, scalability, and reliability, enabling smooth day-to-day store management.

With SQL's querying power, the SMS facilitates quick access to information, empowering store managers to make informed decisions promptly. Additionally, SQL's transactional capabilities ensure the consistency of data, minimizing the risk of errors and discrepancies. The SMS's utilization of SQL enhances data security measures, safeguarding sensitive information from unauthorized access.

Our Python application also makes using and interacting with our SMS and its databases straightforward, simple and easy on the eyes. It isn’t too complicated and is very to use because of its usage of simple GUI elements like buttons, text boxes and confirmational windows. Even inexperienced users who might find it difficult to operate computers can use our SMS through its Python application with little to zero help required.

In conclusion, the Store Management System powered by SQL optimizes store operations, streamlines processes, and enhances overall efficiency. Its robust features make it an indispensable tool for modern retail businesses seeking to maximize productivity and customer satisfaction.

**REFERENCES**

1. **GeeksForGeeks**

https://www.geeksforgeeks.org/department-store-management-systemdsms-using-cpp/

1. **Database System Concepts**

By Abraham Silberschatz, Henry F. Korth and S. Sudharshan

Edition: Seventh

Published by: Tate-McGraw Hill

Publishing Year: 2019

https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/ccs.2020.0018

1. **Retail Store Management System**

By Srikant Surendra Rout

Indira Gandhi National Open University

Published in 2011

1. **Store Up-A Store Management System**

By Shrey Parihar

Acropolis Institute of Technology and Research, Indore

Published in 2022

**DEVELOPMENT ENVIRONMENTS**

* Amazon Web Services-AWS Academy Learner Lab
* Oracle SQL InstaClient (SQL Plus)
* MySQL 8.0 (Command Line Client)
* Visual Studio Code
  + Python 3.12-Tkinter