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 & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

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

SRMIST

SIGNATURE

Dr Annapurani Panaiyappan

Head of the Department

Department of Networking and Communications

SRMIST

TABLE OF CONTENTS

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
   We referred 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.

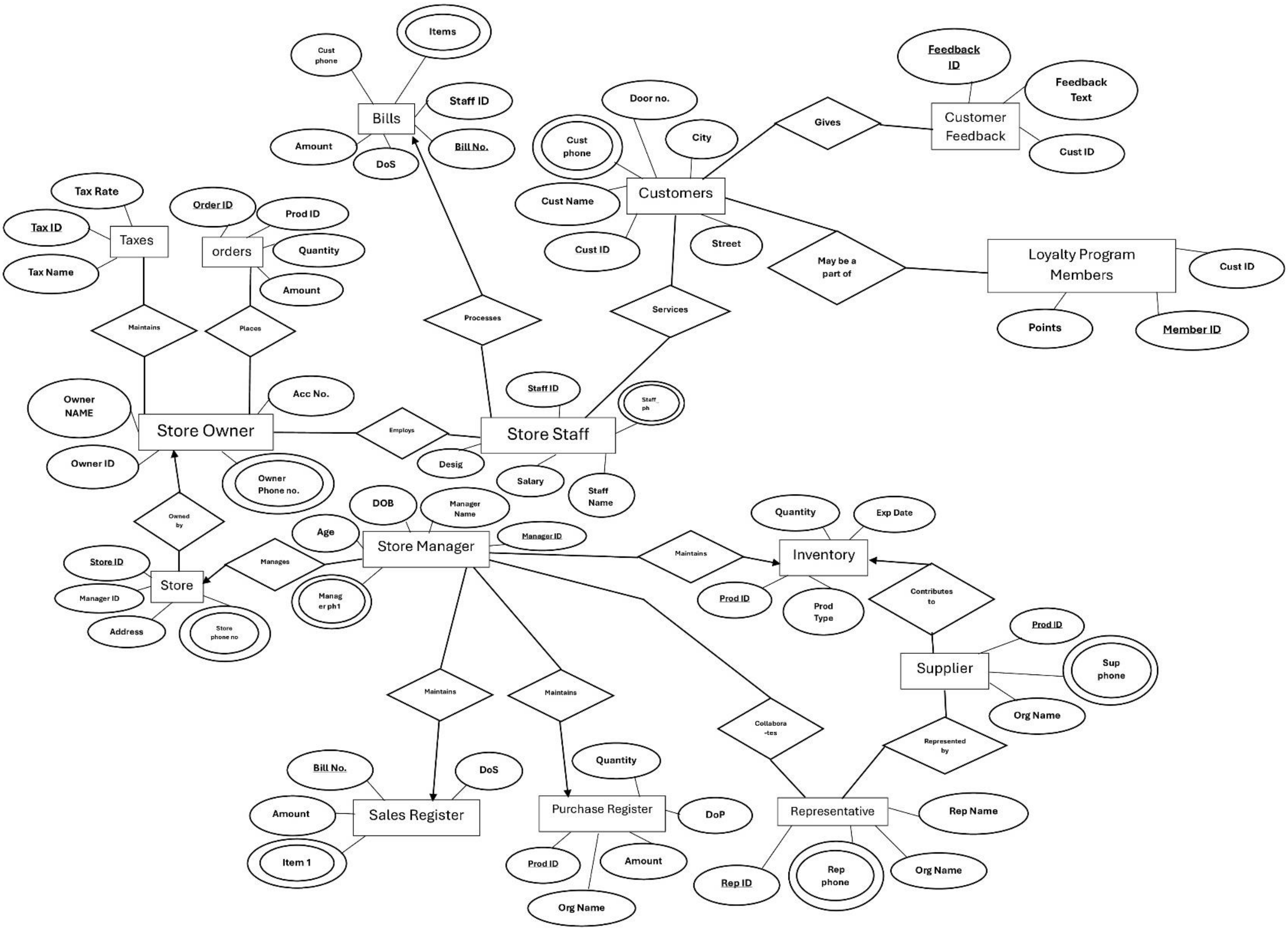
2. **“Effective Use of Retail Store Management System for Small Retail Stores”**
   Authors: Nirosha Wedasinghe and Devni Yasara
   Publication Year: 2021
   We referred to this research paper to understand the need of an effective SMS. The authors, based in Sri Lanka, have noted similar scenarios faced by store owners in Sri Lanka as well as India. We have used this to add and alter features to our SMS.

3. **“Stores Management System”**
   Authors: A. Ganesan, S. Anupama, A. Benitsha
   Publication Year: 2021
   We referred 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.

4. **“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.

4. **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.

5. **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.

6. **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.

7. **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.

8. **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.

9. **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.

10. **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.

11. **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.

12. **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.

13. **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.

14. **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.

15. **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.

2. **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.

3. **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.

4. **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.

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

6. **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.

2. **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.

3. **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.

4. **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.
5. **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.

6. **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.

7. **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.

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

9. **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	

2. **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	

3. **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	

4. **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	

5. **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	

6. **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	

7. **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	

8. **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	

9. 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	

10.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	

11.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	

12.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	

13.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	

14.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	

15.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)
);
```

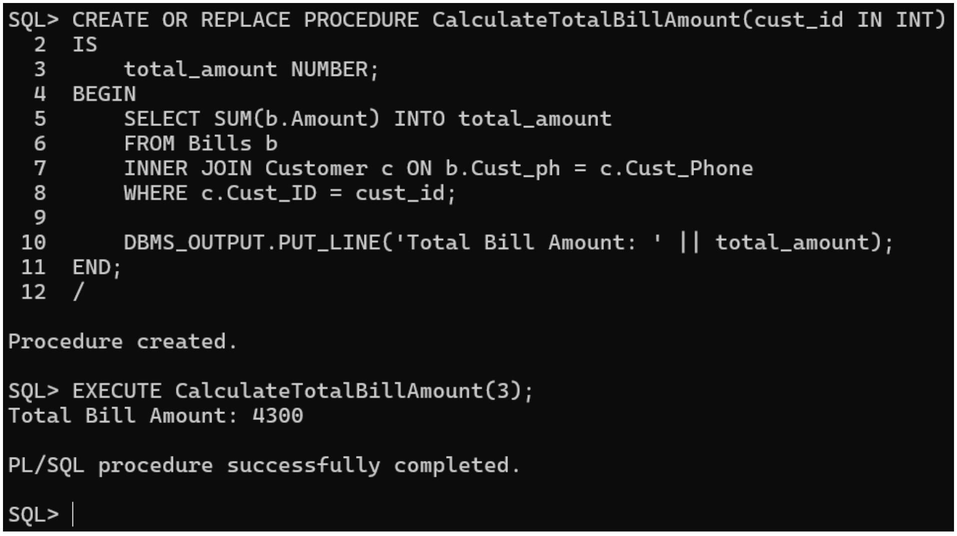
Field	Type	Null	Key	Default	Extra
Bill_no	int	NO	PRI	NULL	
Item1	varchar(255)	YES		NULL	
Item2	varchar(255)	YES		NULL	
Item3	varchar(255)	YES		NULL	
Item4	varchar(255)	YES		NULL	
Item5	varchar(255)	YES		NULL	
Amount	int	YES		NULL	
DoS	date	YES		NULL	

Chapter 7

COMPLEX QUERIES

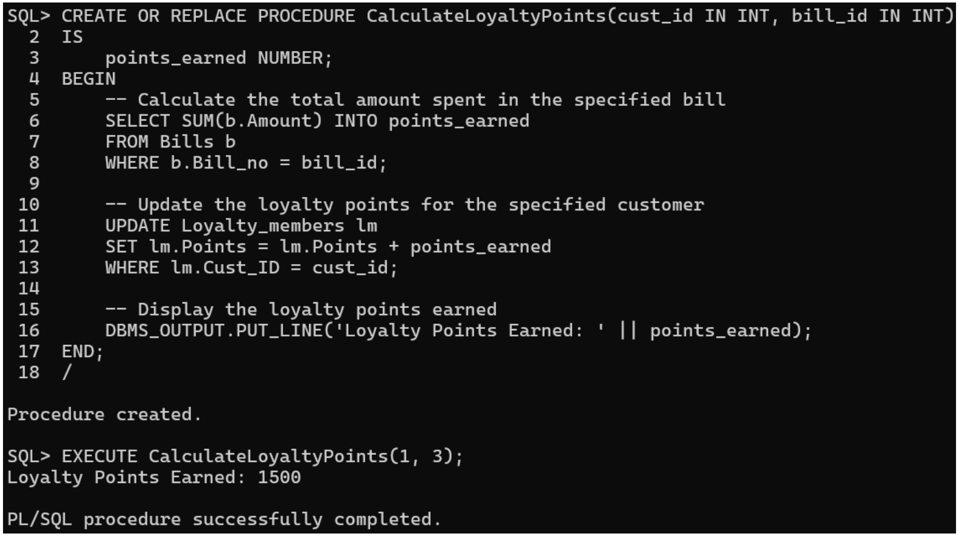
- **PL/SQL**

1. Calculate Total Sales for a given Customer



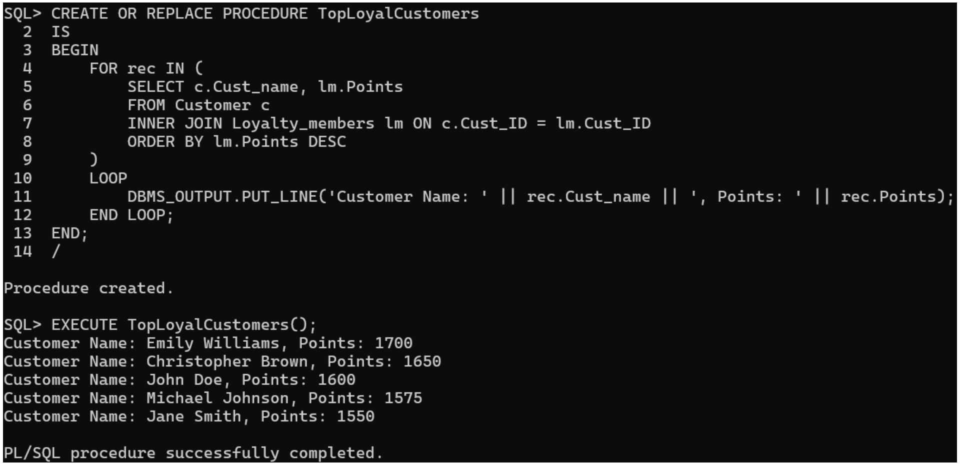
```
SQL> CREATE OR REPLACE PROCEDURE CalculateTotalBillAmount(cust_id IN INT)
IS
    total_amount NUMBER;
BEGIN
    SELECT SUM(b.Amount) INTO total_amount
    FROM Bills b
    INNER JOIN Customer c ON b.Cust_ph = c.Cust_Phone
    WHERE c.Cust_ID = cust_id;

    DBMS_OUTPUT.PUT_LINE('Total Bill Amount: ' || total_amount);
END;
/

Procedure created.

SQL> EXECUTE CalculateTotalBillAmount(3);
Total Bill Amount: 4300

PL/SQL procedure successfully completed.

SQL> |
```

2. Update Loyalty Points for a Customer after a Purchase



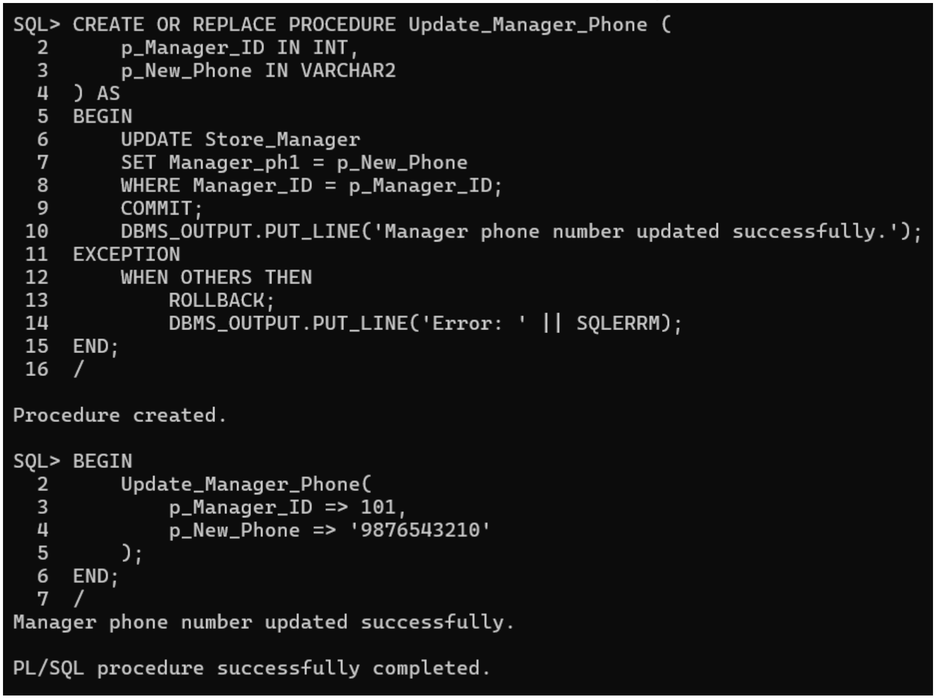
```
SQL> CREATE OR REPLACE PROCEDURE CalculateLoyaltyPoints(cust_id IN INT, bill_id IN INT)
IS
    points_earned NUMBER;
BEGIN
    -- Calculate the total amount spent in the specified bill
    SELECT SUM(b.Amount) INTO points_earned
    FROM Bills b
    WHERE b.Bill_no = bill_id;

    -- Update the loyalty points for the specified customer
    UPDATE Loyalty_members lm
    SET lm.Points = lm.Points + points_earned
    WHERE lm.Cust_ID = cust_id;

    -- Display the loyalty points earned
    DBMS_OUTPUT.PUT_LINE('Loyalty Points Earned: ' || points_earned);
END;
/

Procedure created.

SQL> EXECUTE CalculateLoyaltyPoints(1, 3);
Loyalty Points Earned: 1500

PL/SQL procedure successfully completed.
```

3. Generate a Report of Top Loyal Customers



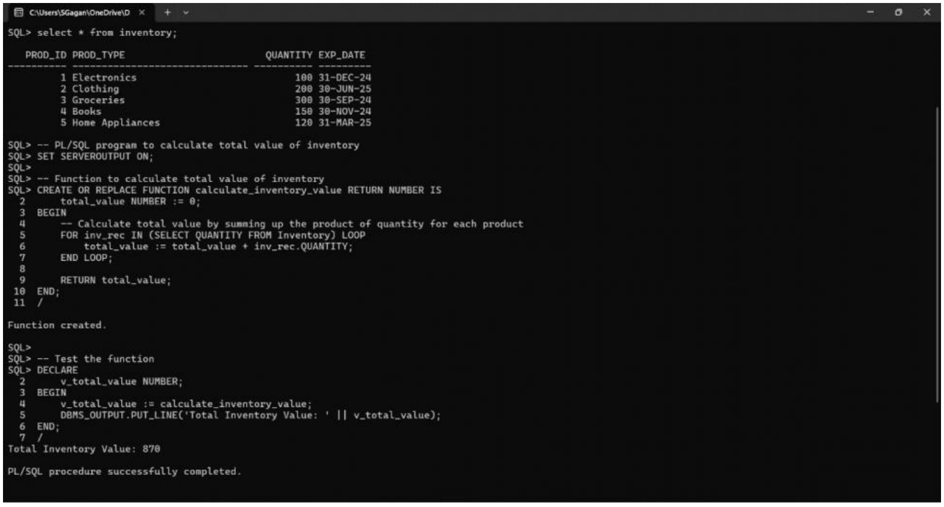
```
SQL> CREATE OR REPLACE PROCEDURE TopLoyalCustomers
IS
BEGIN
    FOR rec IN (
        SELECT c.Cust_name, lm.Points
        FROM Customer c
        INNER JOIN Loyalty_members lm ON c.Cust_ID = lm.Cust_ID
        ORDER BY lm.Points DESC
    )
    LOOP
        DBMS_OUTPUT.PUT_LINE('Customer Name: ' || rec.Cust_name || ', Points: ' || rec.Points);
    END LOOP;
END;
/

Procedure created.

SQL> EXECUTE TopLoyalCustomers();
Customer Name: Emily Williams, Points: 1700
Customer Name: Christopher Brown, Points: 1650
Customer Name: John Doe, Points: 1600
Customer Name: Michael Johnson, Points: 1575
Customer Name: Jane Smith, Points: 1550

PL/SQL procedure successfully completed.
```

4. Update the Phone Number of a Store Manager



```
SQL> CREATE OR REPLACE PROCEDURE Update_Manager_Phone (
         p_Manager_ID IN INT,
         p_New_Phone IN VARCHAR2
     ) AS
     BEGIN
         UPDATE Store_Manager
         SET Manager_ph1 = p_New_Phone
         WHERE Manager_ID = p_Manager_ID;
         COMMIT;
         DBMS_OUTPUT.PUT_LINE('Manager phone number updated successfully.');
     EXCEPTION
         WHEN OTHERS THEN
             ROLLBACK;
             DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
     END;
     /

Procedure created.

SQL> BEGIN
         Update_Manager_Phone(
             p_Manager_ID => 101,
             p_New_Phone => '9876543210'
         );
     END;
     /
Manager phone number updated successfully.

PL/SQL procedure successfully completed.
```

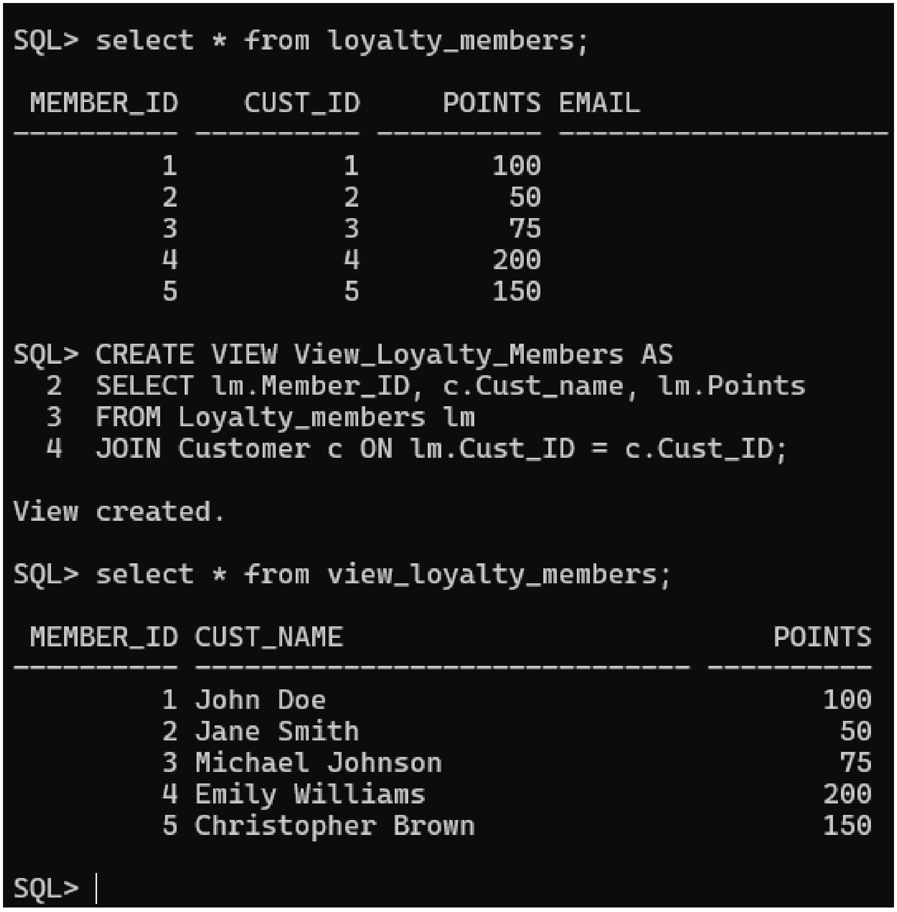
5. Fetching the Total Quantity of the Inventory



```
SQL> select * from inventory;

   PROD_ID PROD_TYPE                        QUANTITY EXP_DATE
---------- ------------------------------ ---------- ---------
         1 Electronics                           100 31-DEC-24
         2 Clothing                              200 30-JUN-25
         3 Groceries                             300 30-SEP-24
         4 Books                                 150 30-NOV-24
         5 Home Appliances                       120 31-MAR-25

SQL> -- PL/SQL program to calculate total value of inventory
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> -- Function to calculate total value of inventory
SQL> CREATE OR REPLACE FUNCTION calculate_inventory_value RETURN NUMBER IS
         total_value NUMBER := 0;
     BEGIN
         -- Calculate total value by summing up the product of quantity for each product
         FOR inv_rec IN (SELECT QUANTITY FROM Inventory) LOOP
             total_value := total_value + inv_rec.QUANTITY;
         END LOOP;

         RETURN total_value;
     END;
     /

Function created.

SQL>
SQL> -- Test the function
SQL> DECLARE
         v_total_value NUMBER;
     BEGIN
         v_total_value := calculate_inventory_value;
         DBMS_OUTPUT.PUT_LINE('Total Inventory Value: ' || v_total_value);
     END;
     /
Total Inventory Value: 870

PL/SQL procedure successfully completed.
```

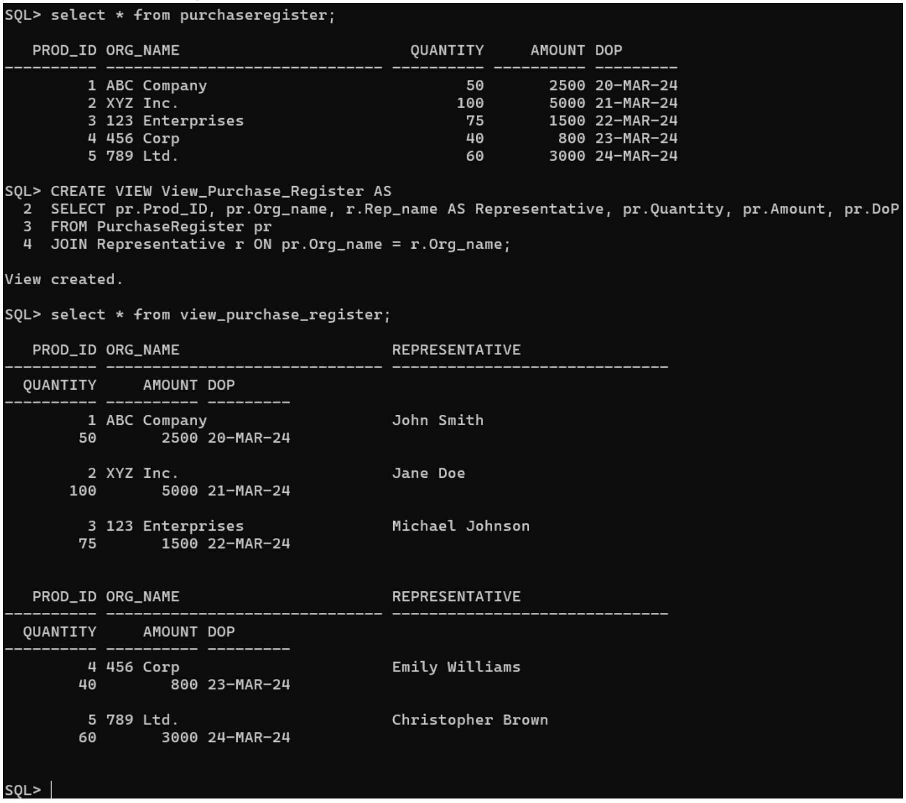
- **Views**

1. Loyalty Members



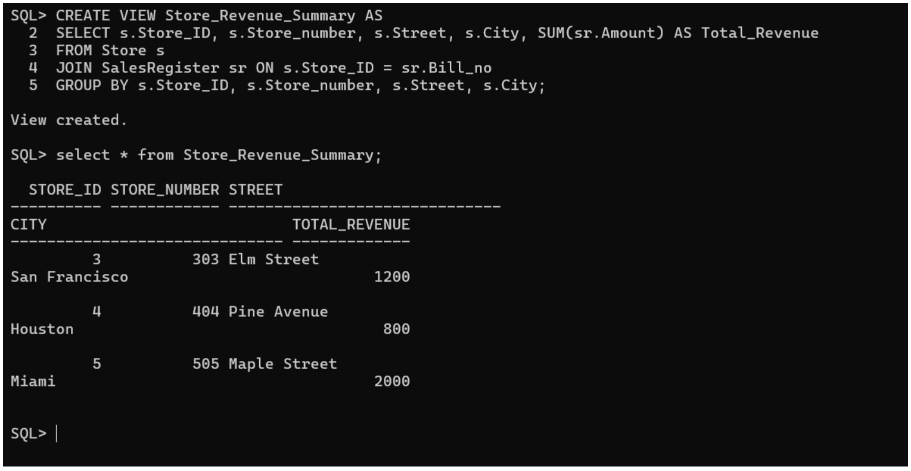
```
SQL> select * from loyalty_members;

 MEMBER_ID    CUST_ID     POINTS EMAIL
---------- ---------- ---------- --------------------
         1          1        100
         2          2         50
         3          3         75
         4          4        200
         5          5        150

SQL> CREATE VIEW View_Loyalty_Members AS
  2  SELECT lm.Member_ID, c.Cust_name, lm.Points
  3  FROM Loyalty_members lm
  4  JOIN Customer c ON lm.Cust_ID = c.Cust_ID;

View created.

SQL> select * from view_loyalty_members;

 MEMBER_ID CUST_NAME                          POINTS
---------- ------------------------------ ----------
         1 John Doe                              100
         2 Jane Smith                             50
         3 Michael Johnson                        75
         4 Emily Williams                        200
         5 Christopher Brown                     150

SQL> |
```

2. Purchase Register



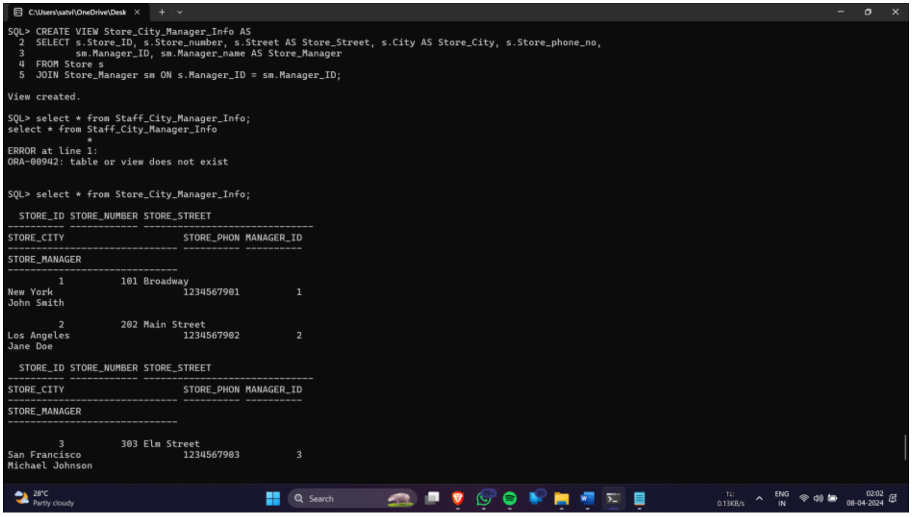
```
SQL> select * from purchaseregister;

   PROD_ID ORG_NAME                         QUANTITY     AMOUNT DOP
---------- ------------------------------ ---------- ---------- ---------
         1 ABC Company                            50       2500 20-MAR-24
         2 XYZ Inc.                              100       5000 21-MAR-24
         3 123 Enterprises                        75       1500 22-MAR-24
         4 456 Corp                               40        800 23-MAR-24
         5 789 Ltd.                               60       3000 24-MAR-24

SQL> CREATE VIEW View_Purchase_Register AS
  2  SELECT pr.Prod_ID, pr.Org_name, r.Rep_name AS Representative, pr.Quantity, pr.Amount, pr.DoP
  3  FROM PurchaseRegister pr
  4  JOIN Representative r ON pr.Org_name = r.Org_name;

View created.

SQL> select * from view_purchase_register;

   PROD_ID ORG_NAME                       REPRESENTATIVE
---------- ------------------------------ ------------------------------
  QUANTITY     AMOUNT DOP
---------- ---------- ---------
         1 ABC Company                    John Smith
        50       2500 20-MAR-24

         2 XYZ Inc.                       Jane Doe
       100       5000 21-MAR-24

         3 123 Enterprises                Michael Johnson
        75       1500 22-MAR-24


   PROD_ID ORG_NAME                       REPRESENTATIVE
---------- ------------------------------ ------------------------------
  QUANTITY     AMOUNT DOP
---------- ---------- ---------
         4 456 Corp                       Emily Williams
        40        800 23-MAR-24

         5 789 Ltd.                       Christopher Brown
        60       3000 24-MAR-24


SQL> |
```

3. Store Revenue Summary



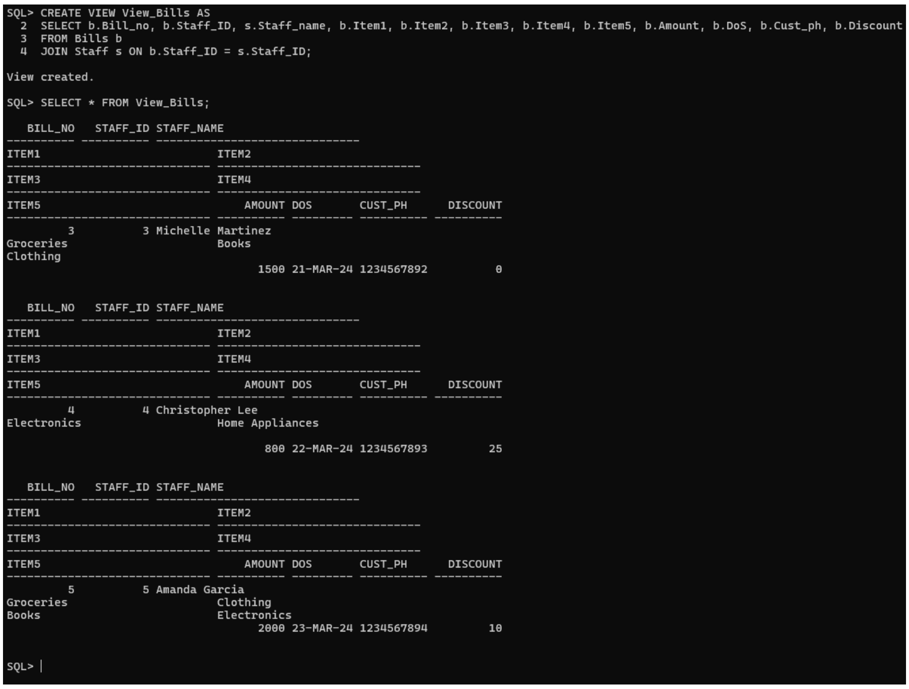
```
SQL> CREATE VIEW Store_Revenue_Summary AS
     SELECT s.Store_ID, s.Store_number, s.Street, s.City, SUM(sr.Amount) AS Total_Revenue
     FROM Store s
     JOIN SalesRegister sr ON s.Store_ID = sr.Bill_no
     GROUP BY s.Store_ID, s.Store_number, s.Street, s.City;

View created.

SQL> select * from Store_Revenue_Summary;

  STORE_ID STORE_NUMBER STREET
---------- ------------ ------------------------------
CITY                           TOTAL_REVENUE
------------------------------ -------------
         3          303 Elm Street
San Francisco                           1200

         4          404 Pine Avenue
Houston                                  800

         5          505 Maple Street
Miami                                   2000


SQL> |
```

4. Store Information with City and Manager



```
SQL> CREATE VIEW Store_City_Manager_Info AS
     SELECT s.Store_ID, s.Store_number, s.Street AS Store_Street, s.City AS Store_City, s.Store_phone_no,
            sm.Manager_ID, sm.Manager_name AS Store_Manager
     FROM Store s
     JOIN Store_Manager sm ON s.Manager_ID = sm.Manager_ID;

View created.

SQL> select * from Staff_City_Manager_Info;
select * from Staff_City_Manager_Info
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> select * from Store_City_Manager_Info;

  STORE_ID STORE_NUMBER STORE_STREET
---------- ------------ ------------------------------
STORE_CITY                     STORE_PHON MANAGER_ID
------------------------------ ---------- ----------
STORE_MANAGER
------------------------------
         1          101 Broadway
New York                       1234567901          1
John Smith

         2          202 Main Street
Los Angeles                    1234567902          2
Jane Doe

  STORE_ID STORE_NUMBER STORE_STREET
---------- ------------ ------------------------------
STORE_CITY                     STORE_PHON MANAGER_ID
------------------------------ ---------- ----------
STORE_MANAGER
------------------------------

         3          303 Elm Street
San Francisco                  1234567903          3
Michael Johnson
```

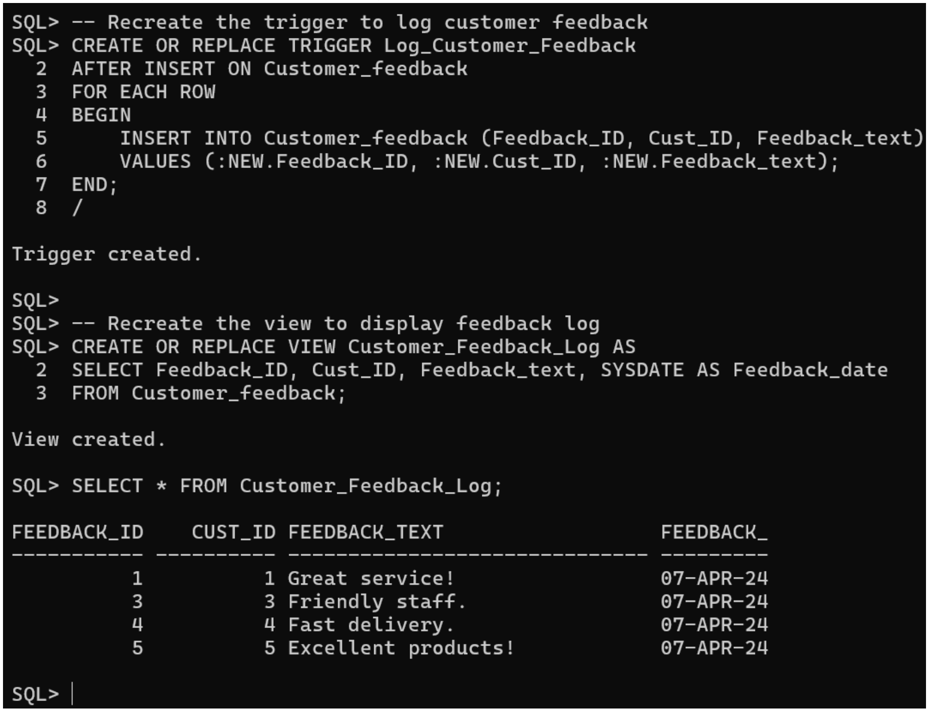
5. Bills



```
SQL> CREATE VIEW View_Bills AS
     SELECT b.Bill_no, b.Staff_ID, s.Staff_name, b.Item1, b.Item2, b.Item3, b.Item4, b.Item5, b.Amount, b.DoS, b.Cust_ph, b.Discount
     FROM Bills b
     JOIN Staff s ON b.Staff_ID = s.Staff_ID;

View created.

SQL> SELECT * FROM View_Bills;

   BILL_NO   STAFF_ID STAFF_NAME
---------- ---------- ------------------------------
ITEM1                          ITEM2
------------------------------ ------------------------------
ITEM3                          ITEM4
------------------------------ ------------------------------
ITEM5                              AMOUNT DOS       CUST_PH      DISCOUNT
------------------------------ ---------- --------- ---------- ----------
         3          3 Michelle Martinez
Groceries                      Books
Clothing
                                     1500 21-MAR-24 1234567892          0


   BILL_NO   STAFF_ID STAFF_NAME
---------- ---------- ------------------------------
ITEM1                          ITEM2
------------------------------ ------------------------------
ITEM3                          ITEM4
------------------------------ ------------------------------
ITEM5                              AMOUNT DOS       CUST_PH      DISCOUNT
------------------------------ ---------- --------- ---------- ----------
         4          4 Christopher Lee
Electronics                    Home Appliances

                                      800 22-MAR-24 1234567893         25


   BILL_NO   STAFF_ID STAFF_NAME
---------- ---------- ------------------------------
ITEM1                          ITEM2
------------------------------ ------------------------------
ITEM3                          ITEM4
------------------------------ ------------------------------
ITEM5                              AMOUNT DOS       CUST_PH      DISCOUNT
------------------------------ ---------- --------- ---------- ----------
         5          5 Amanda Garcia
Groceries                      Clothing
Books                          Electronics
                                     2000 23-MAR-24 1234567894         10


SQL> |
```

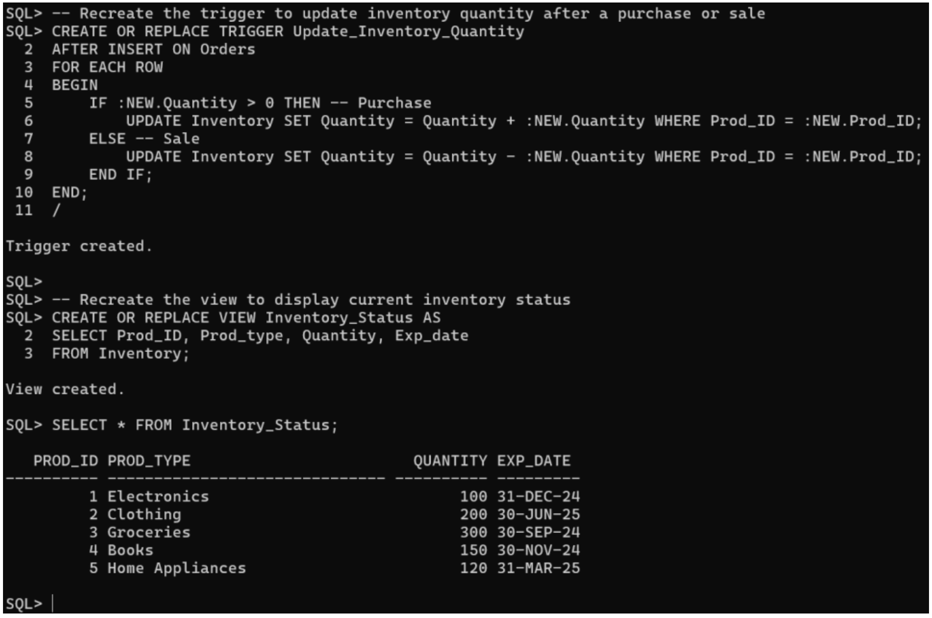
- **Triggers**

1. Capturing and Displaying Customer Feedback



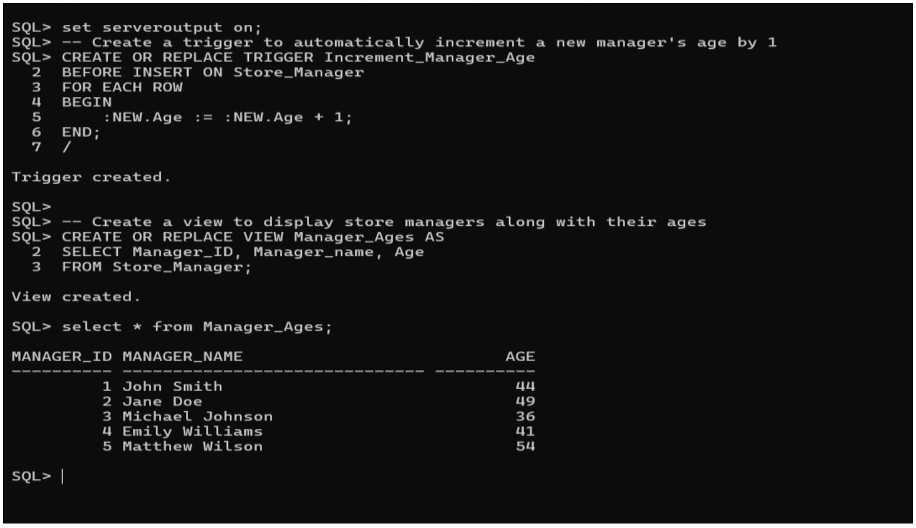
```
SQL> -- Recreate the trigger to log customer feedback
SQL> CREATE OR REPLACE TRIGGER Log_Customer_Feedback
     AFTER INSERT ON Customer_feedback
     FOR EACH ROW
     BEGIN
         INSERT INTO Customer_feedback (Feedback_ID, Cust_ID, Feedback_text)
         VALUES (:NEW.Feedback_ID, :NEW.Cust_ID, :NEW.Feedback_text);
     END;
     /

Trigger created.

SQL>
SQL> -- Recreate the view to display feedback log
SQL> CREATE OR REPLACE VIEW Customer_Feedback_Log AS
     SELECT Feedback_ID, Cust_ID, Feedback_text, SYSDATE AS Feedback_date
     FROM Customer_feedback;

View created.

SQL> SELECT * FROM Customer_Feedback_Log;

FEEDBACK_ID    CUST_ID FEEDBACK_TEXT                  FEEDBACK_
----------- ---------- ------------------------------ ---------
          1          1 Great service!                 07-APR-24
          3          3 Friendly staff.                07-APR-24
          4          4 Fast delivery.                 07-APR-24
          5          5 Excellent products!            07-APR-24

SQL> |
```

2. Updating Inventory after a Purchase or a Sale



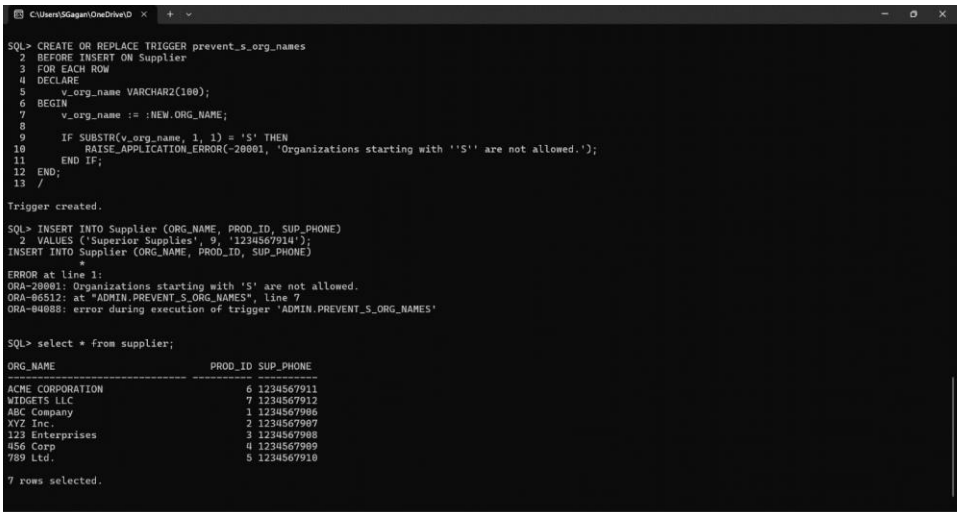
```
SQL> -- Recreate the trigger to update inventory quantity after a purchase or sale
SQL> CREATE OR REPLACE TRIGGER Update_Inventory_Quantity
     AFTER INSERT ON Orders
     FOR EACH ROW
     BEGIN
         IF :NEW.Quantity > 0 THEN -- Purchase
             UPDATE Inventory SET Quantity = Quantity + :NEW.Quantity WHERE Prod_ID = :NEW.Prod_ID;
         ELSE -- Sale
             UPDATE Inventory SET Quantity = Quantity - :NEW.Quantity WHERE Prod_ID = :NEW.Prod_ID;
         END IF;
     END;
     /

Trigger created.

SQL>
SQL> -- Recreate the view to display current inventory status
SQL> CREATE OR REPLACE VIEW Inventory_Status AS
     SELECT Prod_ID, Prod_type, Quantity, Exp_date
     FROM Inventory;

View created.

SQL> SELECT * FROM Inventory_Status;

   PROD_ID PROD_TYPE                        QUANTITY EXP_DATE
---------- ------------------------------ ---------- ---------
         1 Electronics                           100 31-DEC-24
         2 Clothing                              200 30-JUN-25
         3 Groceries                             300 30-SEP-24
         4 Books                                 150 30-NOV-24
         5 Home Appliances                       120 31-MAR-25

SQL> |
```

3. Trigger for Incrementing Manager’s Age



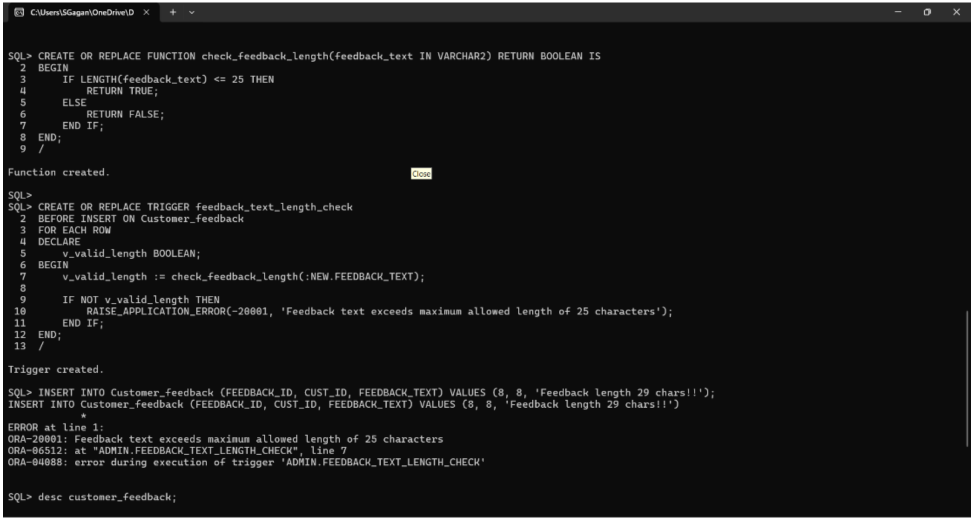
```
SQL> set serveroutput on;
SQL> -- Create a trigger to automatically increment a new manager's age by 1
SQL> CREATE OR REPLACE TRIGGER Increment_Manager_Age
  2  BEFORE INSERT ON Store_Manager
  3  FOR EACH ROW
  4  BEGIN
  5      :NEW.Age := :NEW.Age + 1;
  6  END;
  7  /

Trigger created.

SQL>
SQL> -- Create a view to display store managers along with their ages
SQL> CREATE OR REPLACE VIEW Manager_Ages AS
  2  SELECT Manager_ID, Manager_name, Age
  3  FROM Store_Manager;

View created.

SQL> select * from Manager_Ages;

MANAGER_ID MANAGER_NAME                                  AGE
---------- ------------------------------ ----------
         1 John Smith                                     44
         2 Jane Doe                                       49
         3 Michael Johnson                                36
         4 Emily Williams                                 41
         5 Matthew Wilson                                 54

SQL> |
```

4. Trigger for not allowing Insertion of Organizations whose names start with ‘S’



```
SQL> CREATE OR REPLACE TRIGGER prevent_s_org_names
  2  BEFORE INSERT ON Supplier
  3  FOR EACH ROW
  4  DECLARE
  5      v_org_name VARCHAR2(100);
  6  BEGIN
  7      v_org_name := :NEW.ORG_NAME;
  8
  9      IF SUBSTR(v_org_name, 1, 1) = 'S' THEN
 10          RAISE_APPLICATION_ERROR(-20001, 'Organizations starting with ''S'' are not allowed.');
 11      END IF;
 12  END;
 13  /

Trigger created.

SQL> INSERT INTO Supplier (ORG_NAME, PROD_ID, SUP_PHONE)
  2  VALUES ('Superior Supplies', 9, '1234567914');
INSERT INTO Supplier (ORG_NAME, PROD_ID, SUP_PHONE)
            *
ERROR at line 1:
ORA-20001: Organizations starting with 'S' are not allowed.
ORA-06512: at "ADMIN.PREVENT_S_ORG_NAMES", line 7
ORA-04088: error during execution of trigger 'ADMIN.PREVENT_S_ORG_NAMES'


SQL> select * from supplier;

ORG_NAME                          PROD_ID SUP_PHONE
------------------------------ ---------- ----------
ACME CORPORATION                        6 1234567911
WIDGETS LLC                             7 1234567912
ABC Company                             1 1234567906
XYZ Inc.                                2 1234567907
123 Enterprises                         3 1234567908
456 Corp                                4 1234567909
789 Ltd.                                5 1234567910

7 rows selected.
```

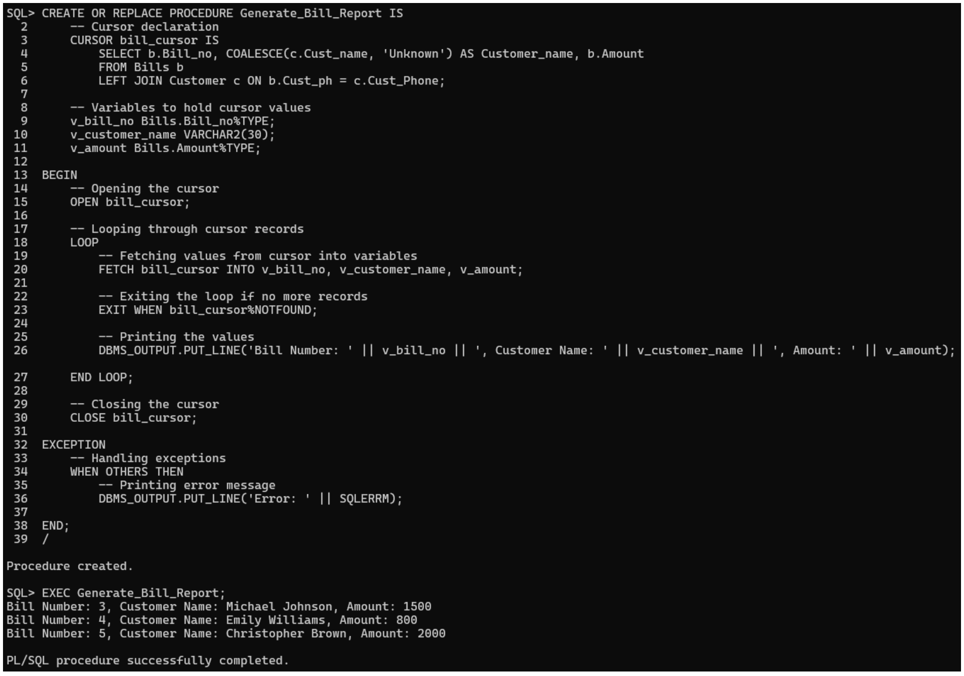
5. Trigger for Limiting Customer Feedback Size to 25 Characters



```
SQL> CREATE OR REPLACE FUNCTION check_feedback_length(feedback_text IN VARCHAR2) RETURN BOOLEAN IS
  2  BEGIN
  3      IF LENGTH(feedback_text) <= 25 THEN
  4          RETURN TRUE;
  5      ELSE
  6          RETURN FALSE;
  7      END IF;
  8  END;
  9  /

Function created.

SQL>
SQL> CREATE OR REPLACE TRIGGER feedback_text_length_check
  2  BEFORE INSERT ON Customer_feedback
  3  FOR EACH ROW
  4  DECLARE
  5      v_valid_length BOOLEAN;
  6  BEGIN
  7      v_valid_length := check_feedback_length(:NEW.FEEDBACK_TEXT);
  8
  9      IF NOT v_valid_length THEN
 10          RAISE_APPLICATION_ERROR(-20001, 'Feedback text exceeds maximum allowed length of 25 characters');
 11      END IF;
 12  END;
 13  /

Trigger created.

SQL> INSERT INTO Customer_feedback (FEEDBACK_ID, CUST_ID, FEEDBACK_TEXT) VALUES (8, 8, 'Feedback length 29 chars!!');
INSERT INTO Customer_feedback (FEEDBACK_ID, CUST_ID, FEEDBACK_TEXT) VALUES (8, 8, 'Feedback length 29 chars!!')
            *
ERROR at line 1:
ORA-20001: Feedback text exceeds maximum allowed length of 25 characters
ORA-06512: at "ADMIN.FEEDBACK_TEXT_LENGTH_CHECK", line 7
ORA-04088: error during execution of trigger 'ADMIN.FEEDBACK_TEXT_LENGTH_CHECK'


SQL> desc customer_feedback;
```

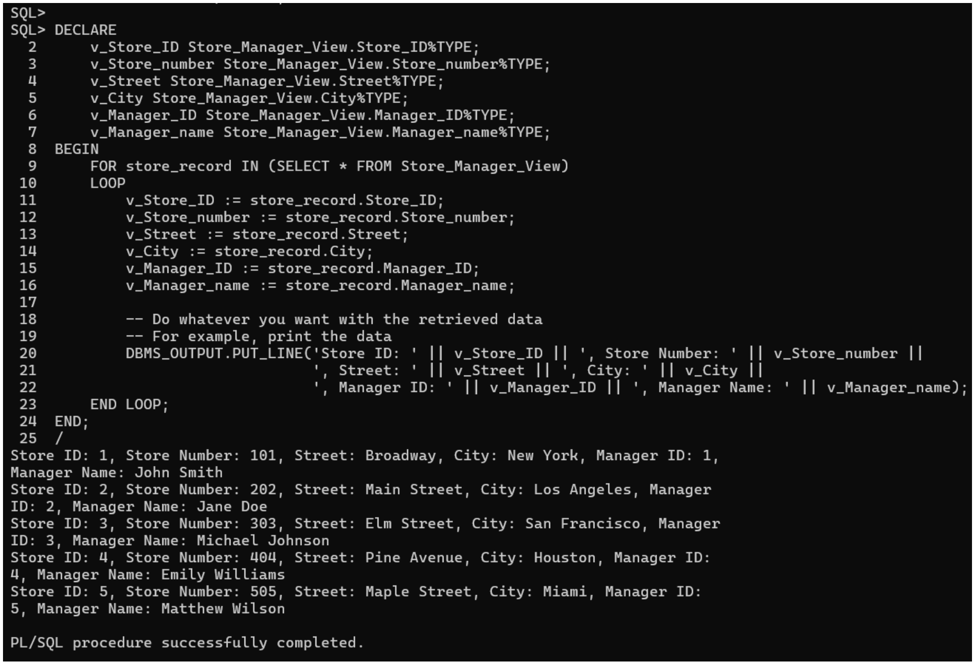
- **Cursors**

1. Generating a Report of Bills with Customer Details



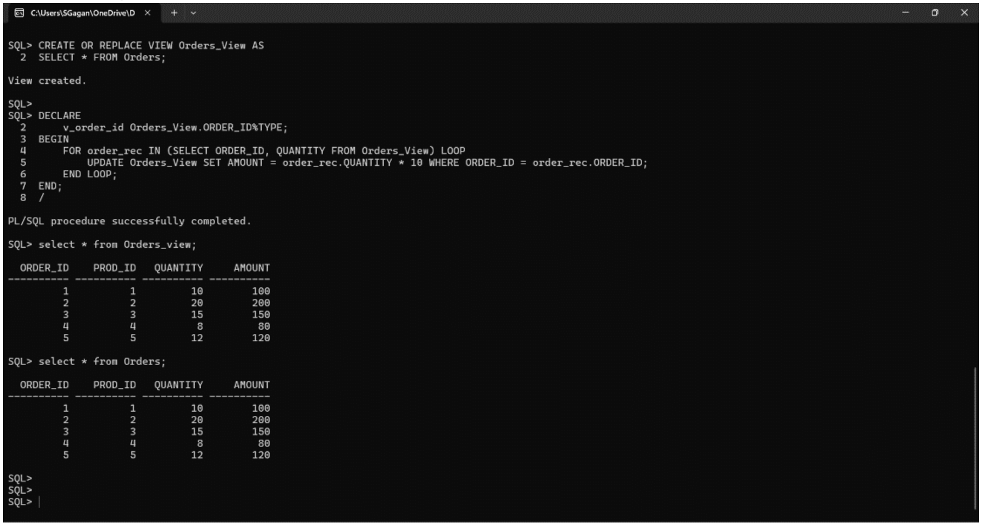
```
SQL> CREATE OR REPLACE PROCEDURE Generate_Bill_Report IS
         -- Cursor declaration
         CURSOR bill_cursor IS
             SELECT b.Bill_no, COALESCE(c.Cust_name, 'Unknown') AS Customer_name, b.Amount
             FROM Bills b
             LEFT JOIN Customer c ON b.Cust_ph = c.Cust_Phone;

         -- Variables to hold cursor values
         v_bill_no Bills.Bill_no%TYPE;
         v_customer_name VARCHAR2(30);
         v_amount Bills.Amount%TYPE;

     BEGIN
         -- Opening the cursor
         OPEN bill_cursor;

         -- Looping through cursor records
         LOOP
             -- Fetching values from cursor into variables
             FETCH bill_cursor INTO v_bill_no, v_customer_name, v_amount;

             -- Exiting the loop if no more records
             EXIT WHEN bill_cursor%NOTFOUND;

             -- Printing the values
             DBMS_OUTPUT.PUT_LINE('Bill Number: ' || v_bill_no || ', Customer Name: ' || v_customer_name || ', Amount: ' || v_amount);

         END LOOP;

         -- Closing the cursor
         CLOSE bill_cursor;

     EXCEPTION
         -- Handling exceptions
         WHEN OTHERS THEN
             -- Printing error message
             DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);

     END;
     /

Procedure created.

SQL> EXEC Generate_Bill_Report;
Bill Number: 3, Customer Name: Michael Johnson, Amount: 1500
Bill Number: 4, Customer Name: Emily Williams, Amount: 800
Bill Number: 5, Customer Name: Christopher Brown, Amount: 2000

PL/SQL procedure successfully completed.
```

2. Retrieving Store Information with the Details of the Store Manager



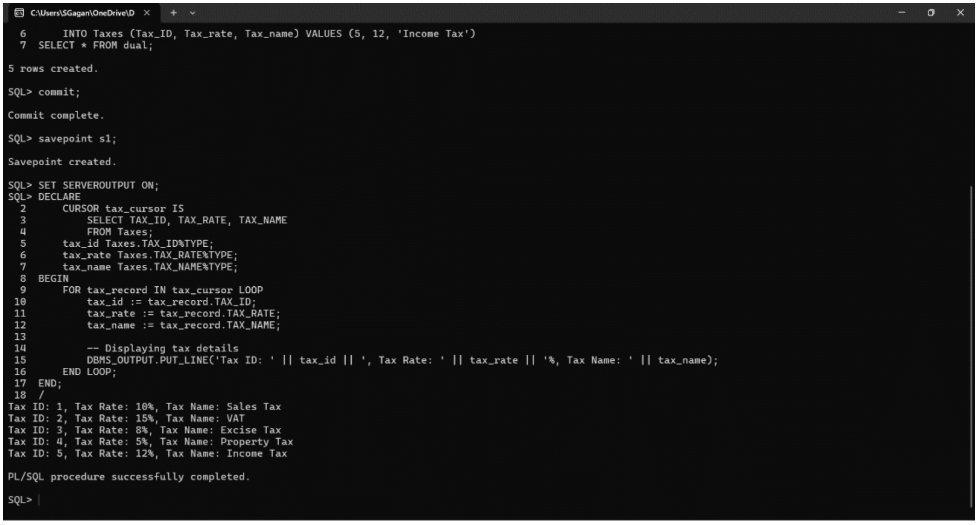
```
SQL>
SQL> DECLARE
         v_Store_ID Store_Manager_View.Store_ID%TYPE;
         v_Store_number Store_Manager_View.Store_number%TYPE;
         v_Street Store_Manager_View.Street%TYPE;
         v_City Store_Manager_View.City%TYPE;
         v_Manager_ID Store_Manager_View.Manager_ID%TYPE;
         v_Manager_name Store_Manager_View.Manager_name%TYPE;
     BEGIN
         FOR store_record IN (SELECT * FROM Store_Manager_View)
         LOOP
             v_Store_ID := store_record.Store_ID;
             v_Store_number := store_record.Store_number;
             v_Street := store_record.Street;
             v_City := store_record.City;
             v_Manager_ID := store_record.Manager_ID;
             v_Manager_name := store_record.Manager_name;

             -- Do whatever you want with the retrieved data
             -- For example, print the data
             DBMS_OUTPUT.PUT_LINE('Store ID: ' || v_Store_ID || ', Store Number: ' || v_Store_number ||
                                  ', Street: ' || v_Street || ', City: ' || v_City ||
                                  ', Manager ID: ' || v_Manager_ID || ', Manager Name: ' || v_Manager_name);
         END LOOP;
     END;
     /
Store ID: 1, Store Number: 101, Street: Broadway, City: New York, Manager ID: 1,
Manager Name: John Smith
Store ID: 2, Store Number: 202, Street: Main Street, City: Los Angeles, Manager
ID: 2, Manager Name: Jane Doe
Store ID: 3, Store Number: 303, Street: Elm Street, City: San Francisco, Manager
ID: 3, Manager Name: Michael Johnson
Store ID: 4, Store Number: 404, Street: Pine Avenue, City: Houston, Manager ID:
4, Manager Name: Emily Williams
Store ID: 5, Store Number: 505, Street: Maple Street, City: Miami, Manager ID:
5, Manager Name: Matthew Wilson

PL/SQL procedure successfully completed.
```

3. Creating an Orders View with Amount=Quantity*10



```
SQL> CREATE OR REPLACE VIEW Orders_View AS
  2  SELECT * FROM Orders;

View created.

SQL>
SQL> DECLARE
  2      v_order_id Orders_View.ORDER_ID%TYPE;
  3  BEGIN
  4      FOR order_rec IN (SELECT ORDER_ID, QUANTITY FROM Orders_View) LOOP
  5          UPDATE Orders_View SET AMOUNT = order_rec.QUANTITY * 10 WHERE ORDER_ID = order_rec.ORDER_ID;
  6      END LOOP;
  7  END;
  8  /

PL/SQL procedure successfully completed.

SQL> select * from Orders_view;

  ORDER_ID    PROD_ID   QUANTITY     AMOUNT
---------- ---------- ---------- ----------
         1          1         10        100
         2          2         20        200
         3          3         15        150
         4          4          8         80
         5          5         12        120

SQL> select * from Orders;

  ORDER_ID    PROD_ID   QUANTITY     AMOUNT
---------- ---------- ---------- ----------
         1          1         10        100
         2          2         20        200
         3          3         15        150
         4          4          8         80
         5          5         12        120

SQL>
SQL>
SQL>
```

4. Fetching Details of Taxes



```
  6      INTO Taxes (Tax_ID, Tax_rate, Tax_name) VALUES (5, 12, 'Income Tax')
  7  SELECT * FROM dual;

5 rows created.

SQL> commit;

Commit complete.

SQL> savepoint s1;

Savepoint created.

SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2      CURSOR tax_cursor IS
  3          SELECT TAX_ID, TAX_RATE, TAX_NAME
  4          FROM Taxes;
  5      tax_id Taxes.TAX_ID%TYPE;
  6      tax_rate Taxes.TAX_RATE%TYPE;
  7      tax_name Taxes.TAX_NAME%TYPE;
  8  BEGIN
  9      FOR tax_record IN tax_cursor LOOP
 10          tax_id := tax_record.TAX_ID;
 11          tax_rate := tax_record.TAX_RATE;
 12          tax_name := tax_record.TAX_NAME;
 13
 14          -- Displaying tax details
 15          DBMS_OUTPUT.PUT_LINE('Tax ID: ' || tax_id || ', Tax Rate: ' || tax_rate || '%, Tax Name: ' || tax_name);
 16      END LOOP;
 17  END;
 18  /
Tax ID: 1, Tax Rate: 10%, Tax Name: Sales Tax
Tax ID: 2, Tax Rate: 15%, Tax Name: VAT
Tax ID: 3, Tax Rate: 8%, Tax Name: Excise Tax
Tax ID: 4, Tax Rate: 5%, Tax Name: Property Tax
Tax ID: 5, Tax Rate: 12%, Tax Name: Income Tax

PL/SQL procedure successfully completed.

SQL>
```

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.
2. **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.
3. **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.
4. **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:

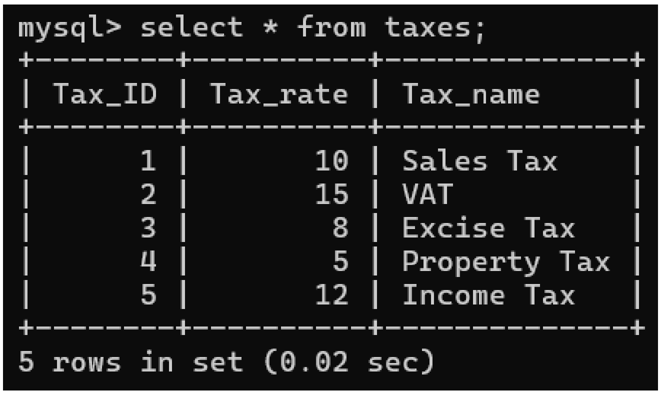
- Normalizing our database schema to reduce redundancy and ensure data consistency.

- 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



```
mysql> select * from taxes;
+--------+----------+--------------+
| Tax_ID | Tax_rate | Tax_name     |
+--------+----------+--------------+
|      1 |       10 | Sales Tax    |
|      2 |       15 | VAT          |
|      3 |        8 | Excise Tax   |
|      4 |        5 | Property Tax |
|      5 |       12 | Income Tax   |
+--------+----------+--------------+
5 rows in set (0.02 sec)
```

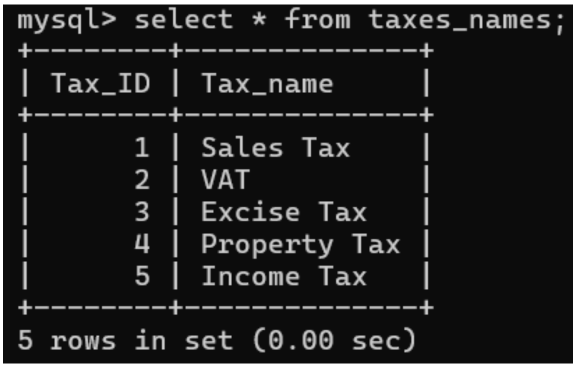
Functional Dependencies:

- Tax_id → Tax_rate
- Tax_id → Tax_name

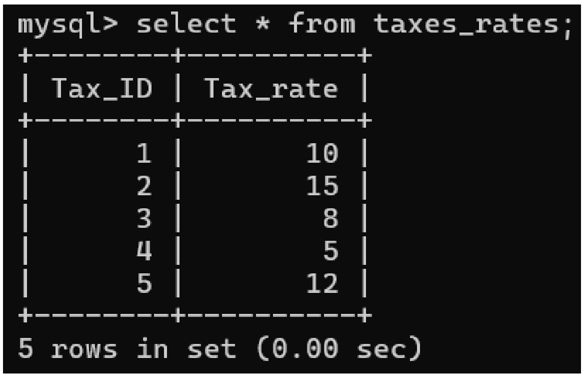
There is a transitive dependency in Tax_id → Tax_name:

- 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'.

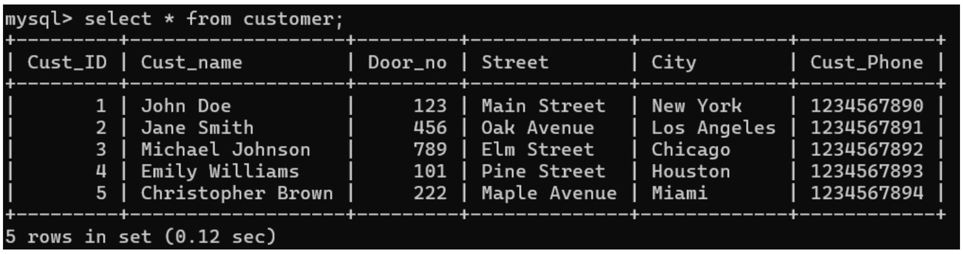


```
mysql> select * from taxes_names;
+--------+--------------+
| Tax_ID | Tax_name     |
+--------+--------------+
|      1 | Sales Tax    |
|      2 | VAT          |
|      3 | Excise Tax   |
|      4 | Property Tax |
|      5 | Income Tax   |
+--------+--------------+
5 rows in set (0.00 sec)
```



```
mysql> select * from taxes_rates;
+--------+----------+
| Tax_ID | Tax_rate |
+--------+----------+
|      1 |       10 |
|      2 |       15 |
|      3 |        8 |
|      4 |        5 |
|      5 |       12 |
+--------+----------+
5 rows in set (0.00 sec)
```

2. Customer



```
mysql> select * from customer;
+---------+---------------------+---------+--------------+-------------+------------+
| Cust_ID | Cust_name           | Door_no | Street       | City        | Cust_Phone |
+---------+---------------------+---------+--------------+-------------+------------+
|       1 | John Doe            |     123 | Main Street  | New York    | 1234567890 |
|       2 | Jane Smith          |     456 | Oak Avenue   | Los Angeles | 1234567891 |
|       3 | Michael Johnson     |     789 | Elm Street   | Chicago     | 1234567892 |
|       4 | Emily Williams      |     101 | Pine Street  | Houston     | 1234567893 |
|       5 | Christopher Brown   |     222 | Maple Avenue | Miami       | 1234567894 |
+---------+---------------------+---------+--------------+-------------+------------+
5 rows in set (0.12 sec)
```

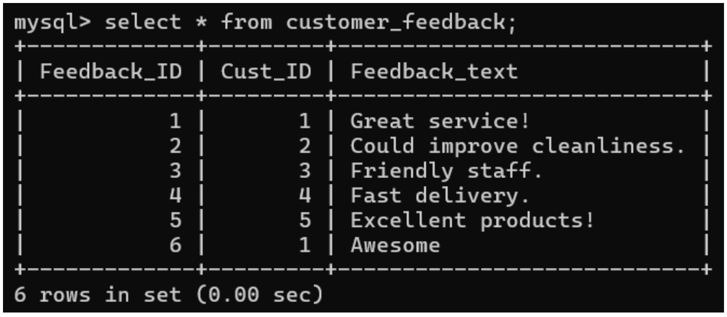
Functional Dependency

- 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

3. Customer_Feedback



```
mysql> select * from customer_feedback;
+-------------+---------+----------------------------+
| Feedback_ID | Cust_ID | Feedback_text              |
+-------------+---------+----------------------------+
|           1 |       1 | Great service!             |
|           2 |       2 | Could improve cleanliness. |
|           3 |       3 | Friendly staff.            |
|           4 |       4 | Fast delivery.             |
|           5 |       5 | Excellent products!        |
|           6 |       1 | Awesome                    |
+-------------+---------+----------------------------+
6 rows in set (0.00 sec)
```

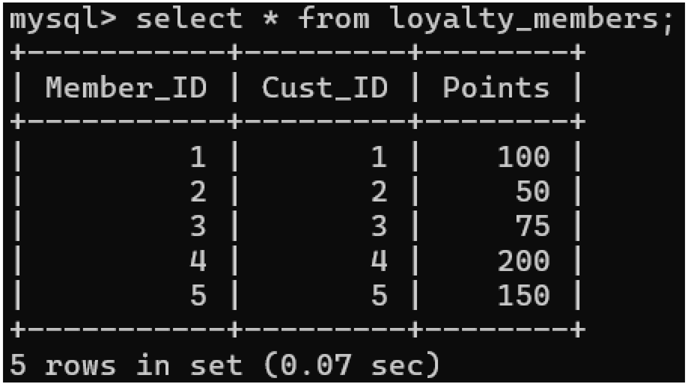
Functional Dependencies:

- 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

4. Loyalty_Members



```
mysql> select * from loyalty_members;
+-----------+---------+--------+
| Member_ID | Cust_ID | Points |
+-----------+---------+--------+
|         1 |       1 |    100 |
|         2 |       2 |     50 |
|         3 |       3 |     75 |
|         4 |       4 |    200 |
|         5 |       5 |    150 |
+-----------+---------+--------+
5 rows in set (0.07 sec)
```

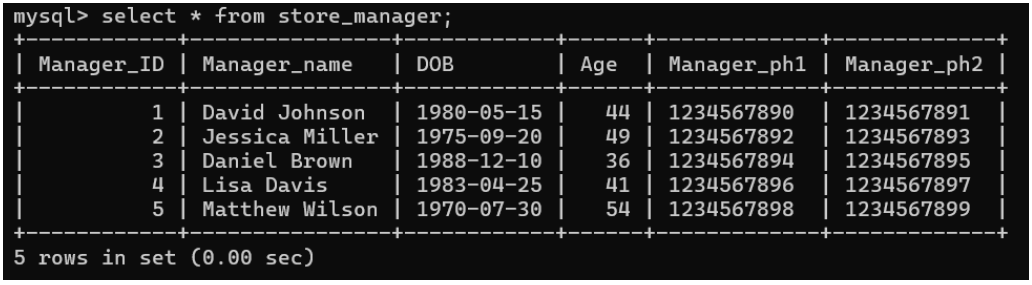
Functional Dependencies:

- 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.

5. Store_Manager



```
mysql> select * from store_manager;
+------------+----------------+------------+------+--------------+-------------+
| Manager_ID | Manager_name   | DOB        | Age  | Manager_ph1  | Manager_ph2 |
+------------+----------------+------------+------+--------------+-------------+
|          1 | David Johnson  | 1980-05-15 |   44 | 1234567890   | 1234567891  |
|          2 | Jessica Miller | 1975-09-20 |   49 | 1234567892   | 1234567893  |
|          3 | Daniel Brown   | 1988-12-10 |   36 | 1234567894   | 1234567895  |
|          4 | Lisa Davis     | 1983-04-25 |   41 | 1234567896   | 1234567897  |
|          5 | Matthew Wilson | 1970-07-30 |   54 | 1234567898   | 1234567899  |
+------------+----------------+------------+------+--------------+-------------+
5 rows in set (0.00 sec)
```

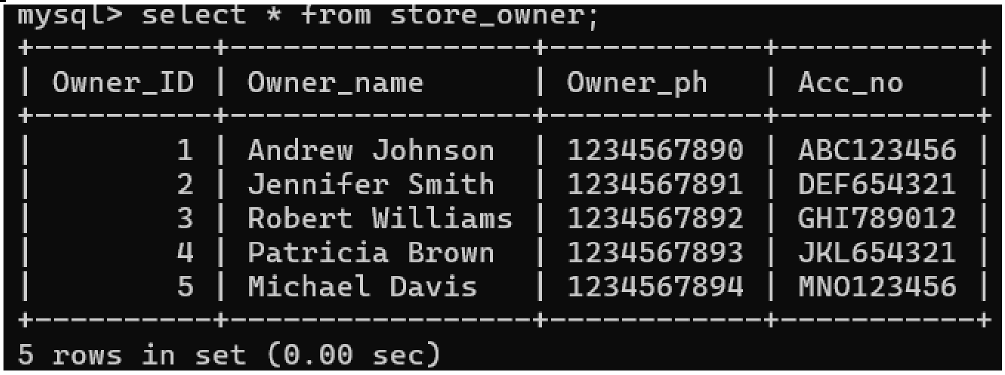
Functional Dependencies:

- 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.

6. Store_Owner



```
mysql> select * from store_owner;
+----------+-----------------+------------+-----------+
| Owner_ID | Owner_name      | Owner_ph   | Acc_no    |
+----------+-----------------+------------+-----------+
|        1 | Andrew Johnson  | 1234567890 | ABC123456 |
|        2 | Jennifer Smith  | 1234567891 | DEF654321 |
|        3 | Robert Williams | 1234567892 | GHI789012 |
|        4 | Patricia Brown  | 1234567893 | JKL654321 |
|        5 | Michael Davis   | 1234567894 | MNO123456 |
+----------+-----------------+------------+-----------+
5 rows in set (0.00 sec)
```

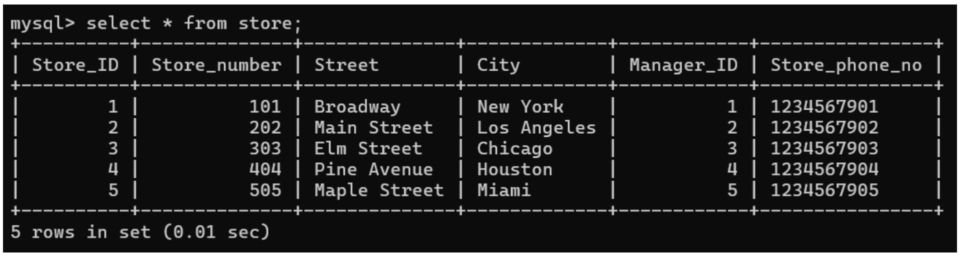
Functional Dependencies:

- 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.

7. Store

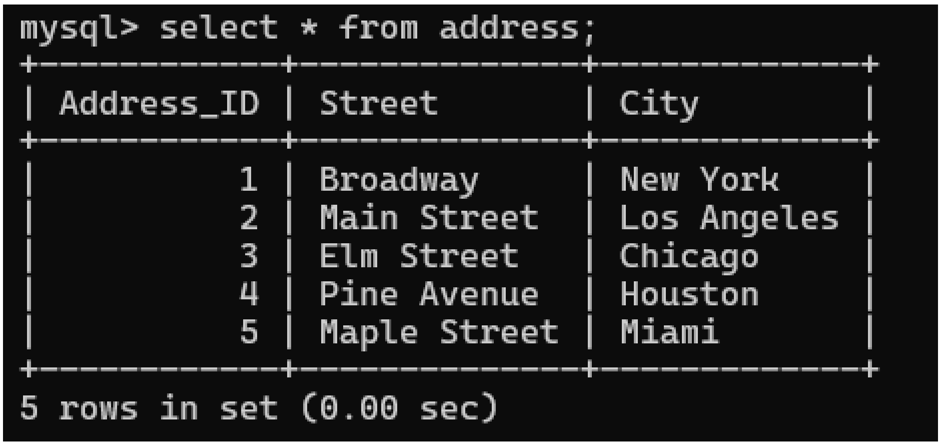


```
mysql> select * from store;
+----------+--------------+--------------+-------------+------------+----------------+
| Store_ID | Store_number | Street       | City        | Manager_ID | Store_phone_no |
+----------+--------------+--------------+-------------+------------+----------------+
|        1 |          101 | Broadway     | New York    |          1 | 1234567901     |
|        2 |          202 | Main Street  | Los Angeles |          2 | 1234567902     |
|        3 |          303 | Elm Street   | Chicago     |          3 | 1234567903     |
|        4 |          404 | Pine Avenue  | Houston     |          4 | 1234567904     |
|        5 |          505 | Maple Street | Miami       |          5 | 1234567905     |
+----------+--------------+--------------+-------------+------------+----------------+
5 rows in set (0.01 sec)
```

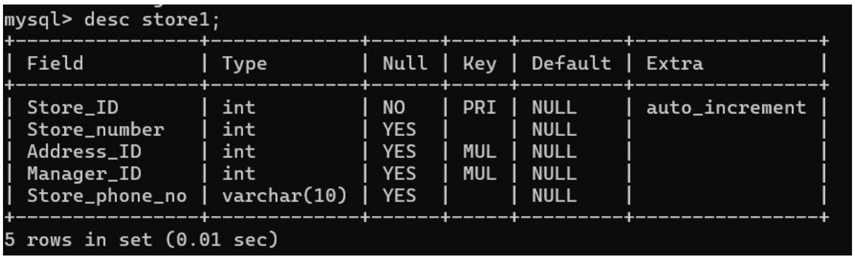
Functional Dependencies:

- 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’.

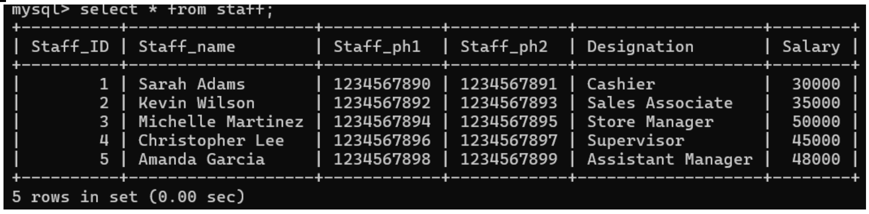


```
mysql> select * from address;
+------------+--------------+-------------+
| Address_ID | Street       | City        |
+------------+--------------+-------------+
|          1 | Broadway     | New York    |
|          2 | Main Street  | Los Angeles |
|          3 | Elm Street   | Chicago     |
|          4 | Pine Avenue  | Houston     |
|          5 | Maple Street | Miami       |
+------------+--------------+-------------+
5 rows in set (0.00 sec)
```



```
mysql> desc store1;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| Store_ID       | int         | NO   | PRI | NULL    | auto_increment |
| Store_number   | int         | YES  |     | NULL    |                |
| Address_ID     | int         | YES  | MUL | NULL    |                |
| Manager_ID     | int         | YES  | MUL | NULL    |                |
| Store_phone_no | varchar(10) | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

8. Store_Staff



```
mysql> select * from staff;
+----------+--------------------+------------+------------+-------------------+--------+
| Staff_ID | Staff_name         | Staff_ph1  | Staff_ph2  | Designation       | Salary |
+----------+--------------------+------------+------------+-------------------+--------+
|        1 | Sarah Adams        | 1234567890 | 1234567891 | Cashier           |  30000 |
|        2 | Kevin Wilson       | 1234567892 | 1234567893 | Sales Associate   |  35000 |
|        3 | Michelle Martinez  | 1234567894 | 1234567895 | Store Manager     |  50000 |
|        4 | Christopher Lee    | 1234567896 | 1234567897 | Supervisor        |  45000 |
|        5 | Amanda Garcia      | 1234567898 | 1234567899 | Assistant Manager |  48000 |
+----------+--------------------+------------+------------+-------------------+--------+
5 rows in set (0.00 sec)
```

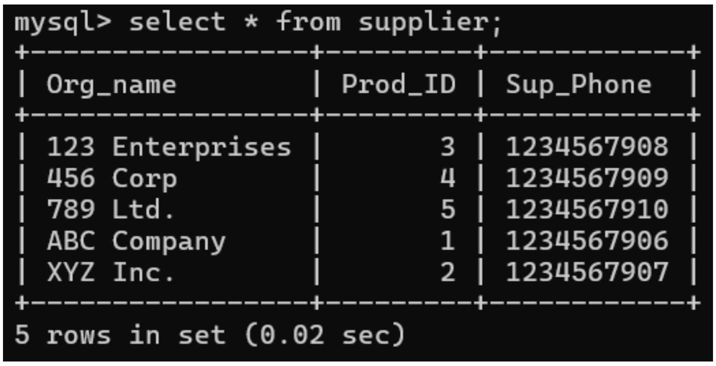
Functional Dependencies:

- 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.

9. Supplier

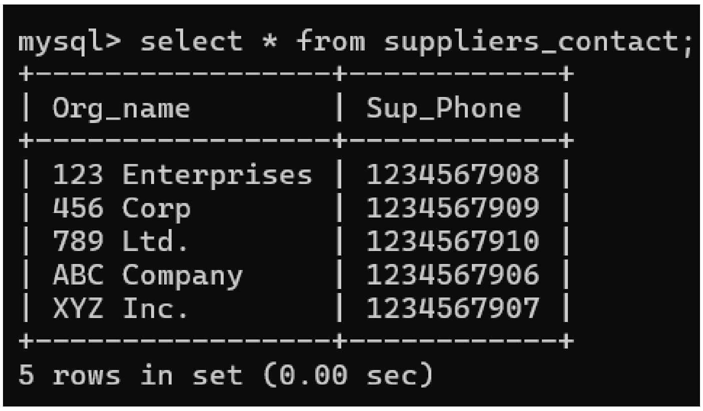


```
mysql> select * from supplier;
+-----------------+---------+------------+
| Org_name        | Prod_ID | Sup_Phone  |
+-----------------+---------+------------+
| 123 Enterprises |       3 | 1234567908 |
| 456 Corp        |       4 | 1234567909 |
| 789 Ltd.        |       5 | 1234567910 |
| ABC Company     |       1 | 1234567906 |
| XYZ Inc.        |       2 | 1234567907 |
+-----------------+---------+------------+
5 rows in set (0.02 sec)
```

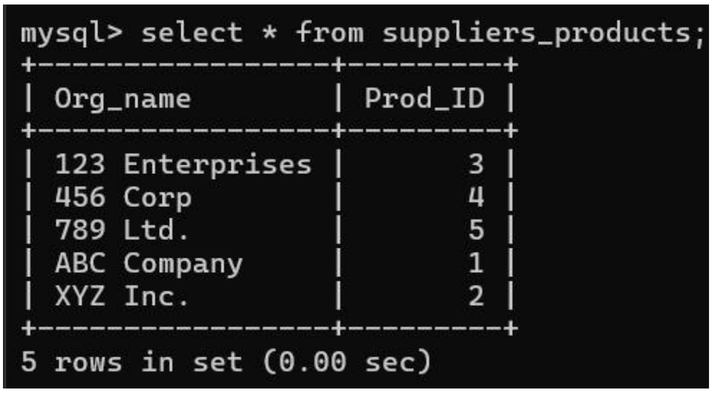
Functional dependencies

- 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.

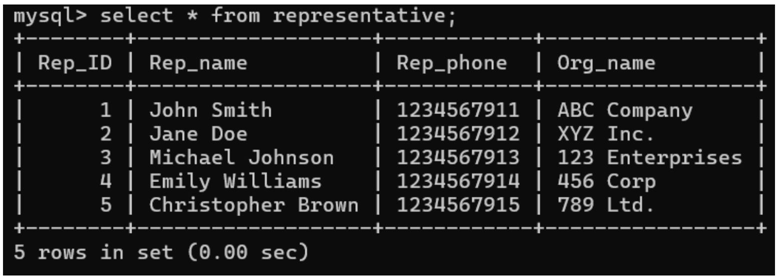


```
mysql> select * from suppliers_contact;
+-----------------+------------+
| Org_name        | Sup_Phone  |
+-----------------+------------+
| 123 Enterprises | 1234567908 |
| 456 Corp        | 1234567909 |
| 789 Ltd.        | 1234567910 |
| ABC Company     | 1234567906 |
| XYZ Inc.        | 1234567907 |
+-----------------+------------+
5 rows in set (0.00 sec)
```



```
mysql> select * from suppliers_products;
+-----------------+---------+
| Org_name        | Prod_ID |
+-----------------+---------+
| 123 Enterprises |       3 |
| 456 Corp        |       4 |
| 789 Ltd.        |       5 |
| ABC Company     |       1 |
| XYZ Inc.        |       2 |
+-----------------+---------+
5 rows in set (0.00 sec)
```

10.Representative



```
mysql> select * from representative;
+--------+-------------------+------------+-----------------+
| Rep_ID | Rep_name          | Rep_phone  | Org_name        |
+--------+-------------------+------------+-----------------+
|      1 | John Smith        | 1234567911 | ABC Company     |
|      2 | Jane Doe          | 1234567912 | XYZ Inc.        |
|      3 | Michael Johnson   | 1234567913 | 123 Enterprises |
|      4 | Emily Williams    | 1234567914 | 456 Corp        |
|      5 | Christopher Brown | 1234567915 | 789 Ltd.        |
+--------+-------------------+------------+-----------------+
5 rows in set (0.00 sec)
```

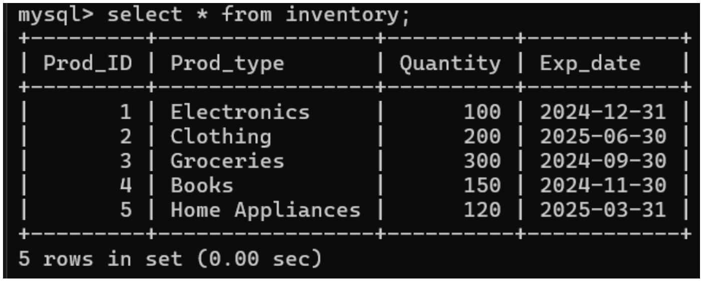
Functional Dependencies:

- 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.

11.Inventory

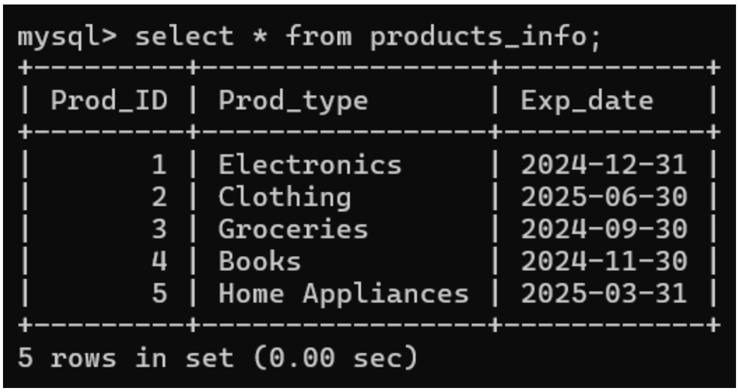


```
mysql> select * from inventory;
+---------+-----------------+----------+------------+
| Prod_ID | Prod_type       | Quantity | Exp_date   |
+---------+-----------------+----------+------------+
|       1 | Electronics     |      100 | 2024-12-31 |
|       2 | Clothing        |      200 | 2025-06-30 |
|       3 | Groceries       |      300 | 2024-09-30 |
|       4 | Books           |      150 | 2024-11-30 |
|       5 | Home Appliances |      120 | 2025-03-31 |
+---------+-----------------+----------+------------+
5 rows in set (0.00 sec)
```

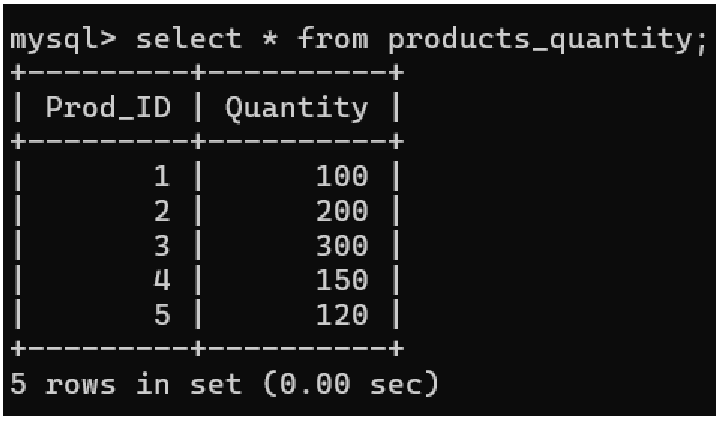
Functional dependencies:

- 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

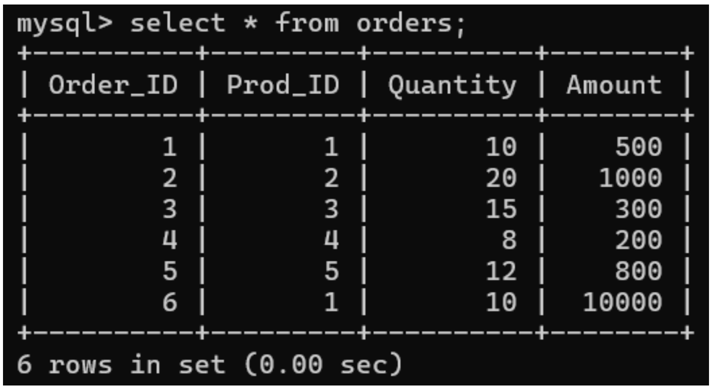


```
mysql> select * from products_info;
+---------+-----------------+------------+
| Prod_ID | Prod_type       | Exp_date   |
+---------+-----------------+------------+
|       1 | Electronics     | 2024-12-31 |
|       2 | Clothing        | 2025-06-30 |
|       3 | Groceries       | 2024-09-30 |
|       4 | Books           | 2024-11-30 |
|       5 | Home Appliances | 2025-03-31 |
+---------+-----------------+------------+
5 rows in set (0.00 sec)
```



```
mysql> select * from products_quantity;
+---------+----------+
| Prod_ID | Quantity |
+---------+----------+
|       1 |      100 |
|       2 |      200 |
|       3 |      300 |
|       4 |      150 |
|       5 |      120 |
+---------+----------+
5 rows in set (0.00 sec)
```

12.Orders



```
mysql> select * from orders;
+----------+---------+----------+--------+
| Order_ID | Prod_ID | Quantity | Amount |
+----------+---------+----------+--------+
|        1 |       1 |       10 |    500 |
|        2 |       2 |       20 |   1000 |
|        3 |       3 |       15 |    300 |
|        4 |       4 |        8 |    200 |
|        5 |       5 |       12 |    800 |
|        6 |       1 |       10 |  10000 |
+----------+---------+----------+--------+
6 rows in set (0.00 sec)
```

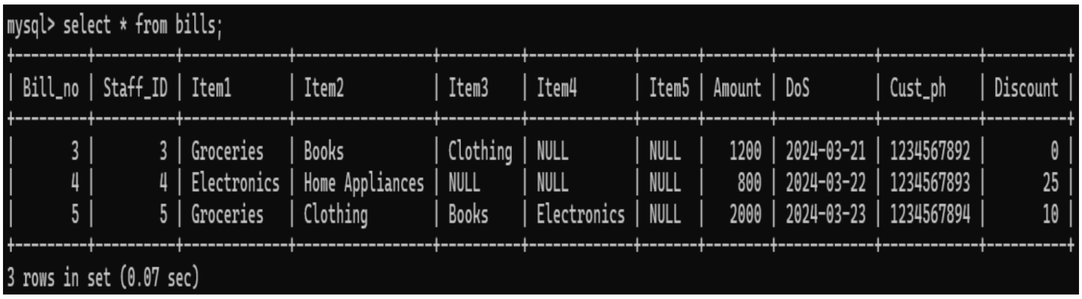
Functional Dependencies:

- 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

13.Bills



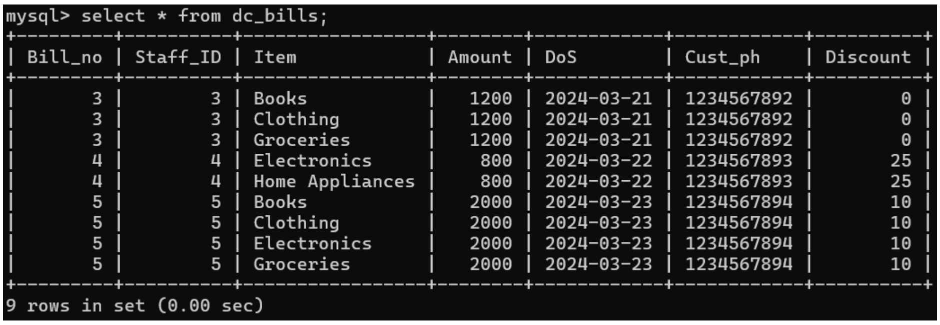
```
mysql> select * from bills;
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
| Bill_no | Staff_ID | Item1       | Item2           | Item3    | Item4       | Item5 | Amount | DoS        | Cust_ph    | Discount |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
|       3 |        3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 | 1234567892 |        0 |
|       4 |        4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 | 1234567893 |       25 |
|       5 |        5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 | 1234567894 |       10 |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
3 rows in set (0.07 sec)
```

Functional Dependency:

- Bill_no → Staff_ID, Item1, Item2, Item3, Item4, Item5, Amount, DoS, Cust_ph, Discount

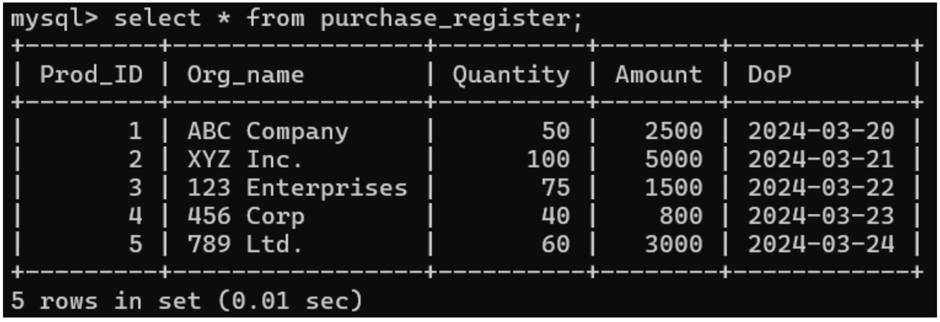
This table has repeating groups.

This can be normalized by using 1NF.



```
mysql> select * from dc_bills;
+---------+----------+-----------------+--------+------------+------------+----------+
| Bill_no | Staff_ID | Item            | Amount | DoS        | Cust_ph    | Discount |
+---------+----------+-----------------+--------+------------+------------+----------+
|       3 |        3 | Books           |   1200 | 2024-03-21 | 1234567892 |        0 |
|       3 |        3 | Clothing        |   1200 | 2024-03-21 | 1234567892 |        0 |
|       3 |        3 | Groceries       |   1200 | 2024-03-21 | 1234567892 |        0 |
|       4 |        4 | Electronics     |    800 | 2024-03-22 | 1234567893 |       25 |
|       4 |        4 | Home Appliances |    800 | 2024-03-22 | 1234567893 |       25 |
|       5 |        5 | Books           |   2000 | 2024-03-23 | 1234567894 |       10 |
|       5 |        5 | Clothing        |   2000 | 2024-03-23 | 1234567894 |       10 |
|       5 |        5 | Electronics     |   2000 | 2024-03-23 | 1234567894 |       10 |
|       5 |        5 | Groceries       |   2000 | 2024-03-23 | 1234567894 |       10 |
+---------+----------+-----------------+--------+------------+------------+----------+
9 rows in set (0.00 sec)
```

14.Purchase_Register

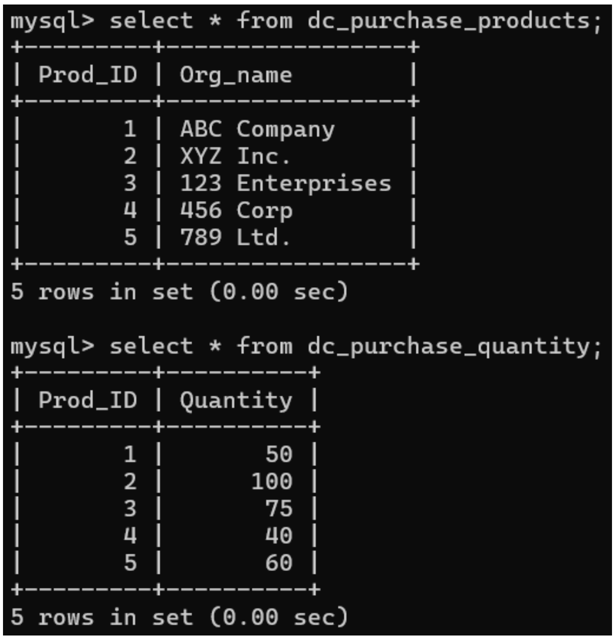


```
mysql> select * from purchase_register;
+---------+-----------------+----------+--------+------------+
| Prod_ID | Org_name        | Quantity | Amount | DoP        |
+---------+-----------------+----------+--------+------------+
|       1 | ABC Company     |       50 |   2500 | 2024-03-20 |
|       2 | XYZ Inc.        |      100 |   5000 | 2024-03-21 |
|       3 | 123 Enterprises |       75 |   1500 | 2024-03-22 |
|       4 | 456 Corp        |       40 |    800 | 2024-03-23 |
|       5 | 789 Ltd.        |       60 |   3000 | 2024-03-24 |
+---------+-----------------+----------+--------+------------+
5 rows in set (0.01 sec)
```

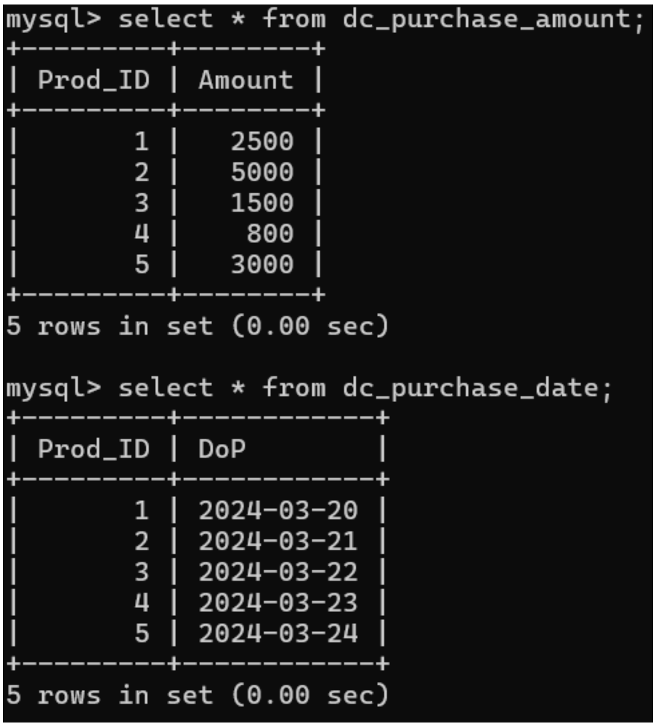
Functional Dependencies:

- 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.

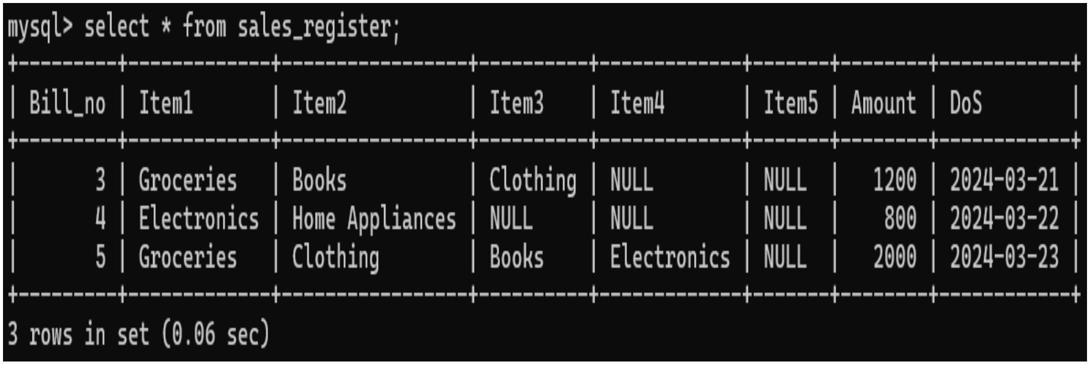


```
mysql> select * from dc_purchase_products;
+---------+-----------------+
| Prod_ID | Org_name        |
+---------+-----------------+
|       1 | ABC Company     |
|       2 | XYZ Inc.        |
|       3 | 123 Enterprises |
|       4 | 456 Corp        |
|       5 | 789 Ltd.        |
+---------+-----------------+
5 rows in set (0.00 sec)

mysql> select * from dc_purchase_quantity;
+---------+----------+
| Prod_ID | Quantity |
+---------+----------+
|       1 |       50 |
|       2 |      100 |
|       3 |       75 |
|       4 |       40 |
|       5 |       60 |
+---------+----------+
5 rows in set (0.00 sec)
```



```
mysql> select * from dc_purchase_amount;
+---------+--------+
| Prod_ID | Amount |
+---------+--------+
|       1 |   2500 |
|       2 |   5000 |
|       3 |   1500 |
|       4 |    800 |
|       5 |   3000 |
+---------+--------+
5 rows in set (0.00 sec)

mysql> select * from dc_purchase_date;
+---------+------------+
| Prod_ID | DoP        |
+---------+------------+
|       1 | 2024-03-20 |
|       2 | 2024-03-21 |
|       3 | 2024-03-22 |
|       4 | 2024-03-23 |
|       5 | 2024-03-24 |
+---------+------------+
5 rows in set (0.00 sec)
```

15.Sales_Register

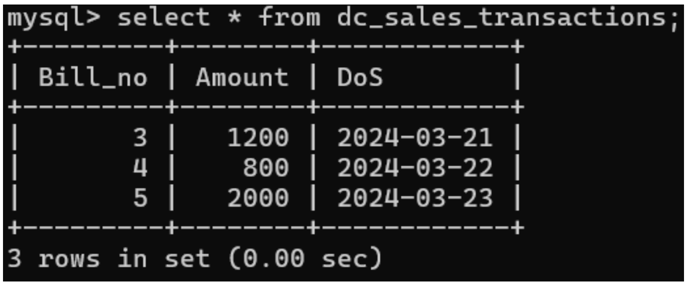


```
mysql> select * from sales_register;
+---------+-------------+-----------------+----------+-------------+-------+--------+------------+
| Bill_no | Item1       | Item2           | Item3    | Item4       | Item5 | Amount | DoS        |
+---------+-------------+-----------------+----------+-------------+-------+--------+------------+
|       3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 |
|       4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 |
|       5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 |
+---------+-------------+-----------------+----------+-------------+-------+--------+------------+
3 rows in set (0.06 sec)
```

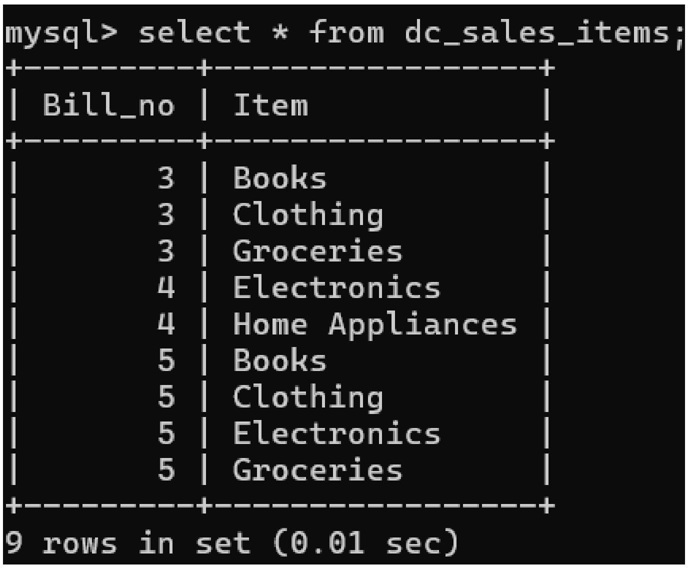
Functional Dependencies:

- Bill_no → Item1, Item2, Item3, Item4, Item5, Amount, DoS

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



```
mysql> select * from dc_sales_transactions;
+---------+--------+------------+
| Bill_no | Amount | DoS        |
+---------+--------+------------+
|       3 |   1200 | 2024-03-21 |
|       4 |    800 | 2024-03-22 |
|       5 |   2000 | 2024-03-23 |
+---------+--------+------------+
3 rows in set (0.00 sec)
```



```
mysql> select * from dc_sales_items;
+---------+-----------------+
| Bill_no | Item            |
+---------+-----------------+
|       3 | Books           |
|       3 | Clothing        |
|       3 | Groceries       |
|       4 | Electronics     |
|       4 | Home Appliances |
|       5 | Books           |
|       5 | Clothing        |
|       5 | Electronics     |
|       5 | Groceries       |
+---------+-----------------+
9 rows in set (0.01 sec)
```

Chapter 9

CONCURRENCY CONTROL

Transactions in a database are of two types mainly:

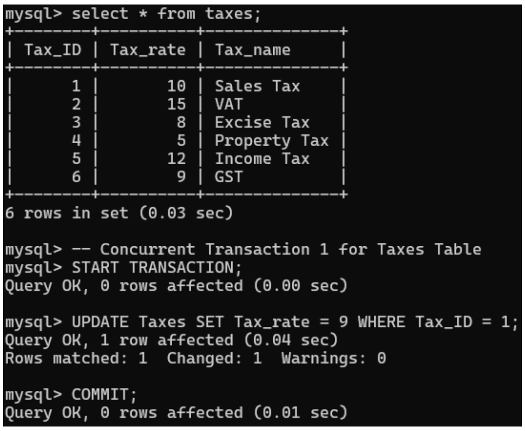
- **Concurrent Transactions:** In a concurrent transaction schedule, multiple transactions can execute simultaneously. This allows for better utilization of system resources and can improve overall system throughput. However, concurrency introduces the possibility of interference between transactions, 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.

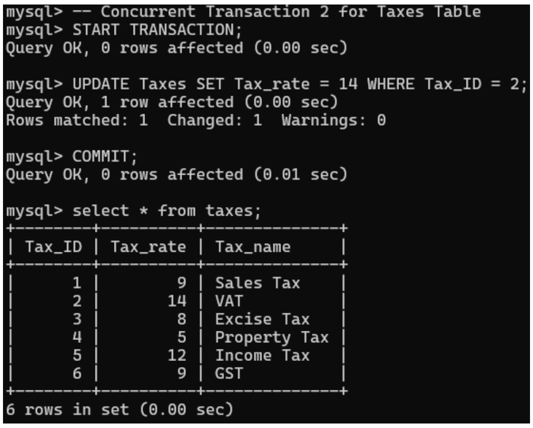
Some transactions in SMS are given hereafter:

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

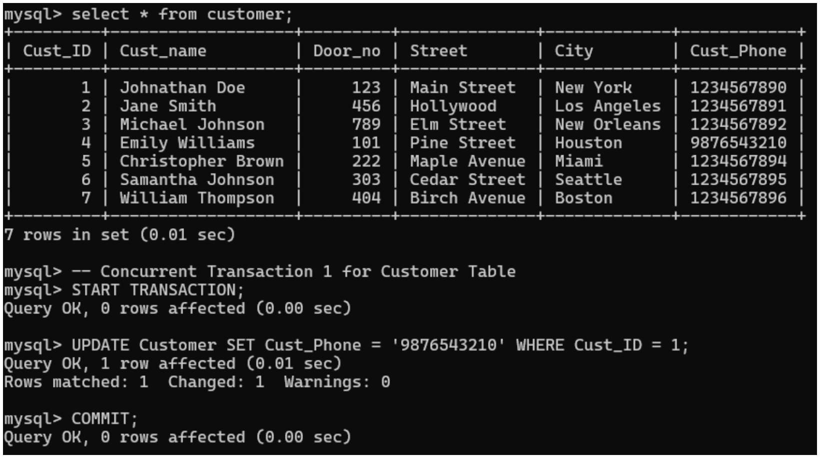


```
mysql> select * from taxes;
+--------+----------+--------------+
| Tax_ID | Tax_rate | Tax_name     |
+--------+----------+--------------+
|      1 |       10 | Sales Tax    |
|      2 |       15 | VAT          |
|      3 |        8 | Excise Tax   |
|      4 |        5 | Property Tax |
|      5 |       12 | Income Tax   |
|      6 |        9 | GST          |
+--------+----------+--------------+
6 rows in set (0.03 sec)

mysql> -- Concurrent Transaction 1 for Taxes Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Taxes SET Tax_rate = 9 WHERE Tax_ID = 1;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

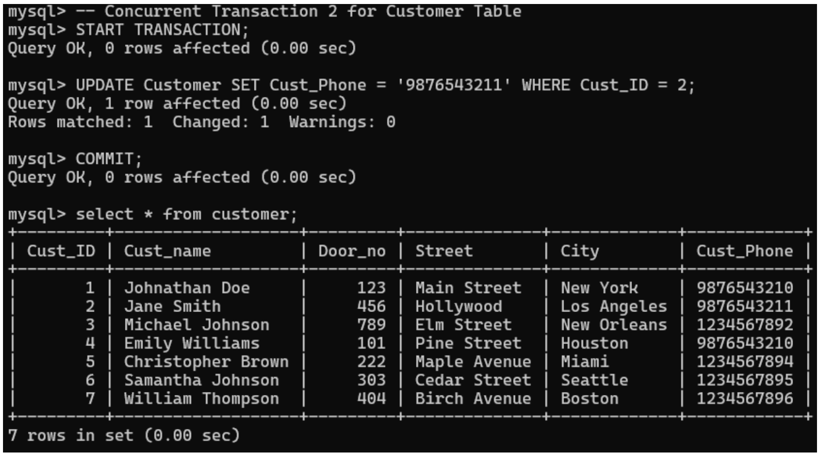


```
mysql> -- Concurrent Transaction 2 for Taxes Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Taxes SET Tax_rate = 14 WHERE Tax_ID = 2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from taxes;
+--------+----------+--------------+
| Tax_ID | Tax_rate | Tax_name     |
+--------+----------+--------------+
|      1 |        9 | Sales Tax    |
|      2 |       14 | VAT          |
|      3 |        8 | Excise Tax   |
|      4 |        5 | Property Tax |
|      5 |       12 | Income Tax   |
|      6 |        9 | GST          |
+--------+----------+--------------+
6 rows in set (0.00 sec)
```

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

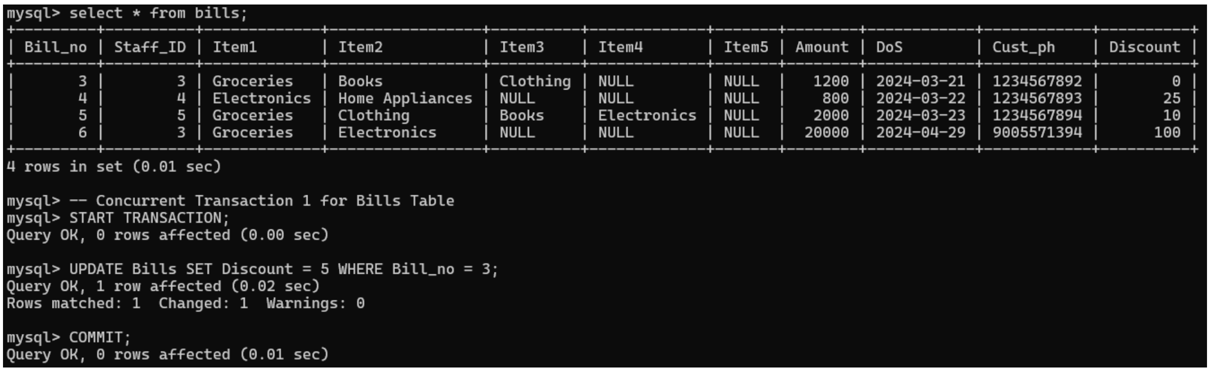


```
mysql> select * from customer;
+---------+--------------------+---------+--------------+-------------+------------+
| Cust_ID | Cust_name          | Door_no | Street       | City        | Cust_Phone |
+---------+--------------------+---------+--------------+-------------+------------+
|       1 | Johnathan Doe      |     123 | Main Street  | New York    | 1234567890 |
|       2 | Jane Smith         |     456 | Hollywood    | Los Angeles | 1234567891 |
|       3 | Michael Johnson    |     789 | Elm Street   | New Orleans | 1234567892 |
|       4 | Emily Williams     |     101 | Pine Street  | Houston     | 9876543210 |
|       5 | Christopher Brown  |     222 | Maple Avenue | Miami       | 1234567894 |
|       6 | Samantha Johnson   |     303 | Cedar Street | Seattle     | 1234567895 |
|       7 | William Thompson   |     404 | Birch Avenue | Boston      | 1234567896 |
+---------+--------------------+---------+--------------+-------------+------------+
7 rows in set (0.01 sec)

mysql> -- Concurrent Transaction 1 for Customer Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Customer SET Cust_Phone = '9876543210' WHERE Cust_ID = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

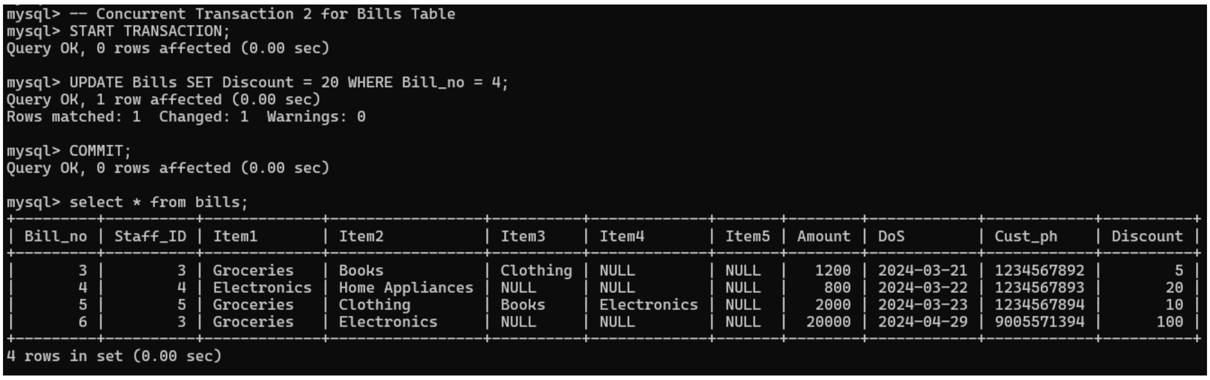


```
mysql> -- Concurrent Transaction 2 for Customer Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Customer SET Cust_Phone = '9876543211' WHERE Cust_ID = 2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customer;
+---------+--------------------+---------+--------------+-------------+------------+
| Cust_ID | Cust_name          | Door_no | Street       | City        | Cust_Phone |
+---------+--------------------+---------+--------------+-------------+------------+
|       1 | Johnathan Doe      |     123 | Main Street  | New York    | 9876543210 |
|       2 | Jane Smith         |     456 | Hollywood    | Los Angeles | 9876543211 |
|       3 | Michael Johnson    |     789 | Elm Street   | New Orleans | 1234567892 |
|       4 | Emily Williams     |     101 | Pine Street  | Houston     | 9876543210 |
|       5 | Christopher Brown  |     222 | Maple Avenue | Miami       | 1234567894 |
|       6 | Samantha Johnson   |     303 | Cedar Street | Seattle     | 1234567895 |
|       7 | William Thompson   |     404 | Birch Avenue | Boston      | 1234567896 |
+---------+--------------------+---------+--------------+-------------+------------+
7 rows in set (0.00 sec)
```

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



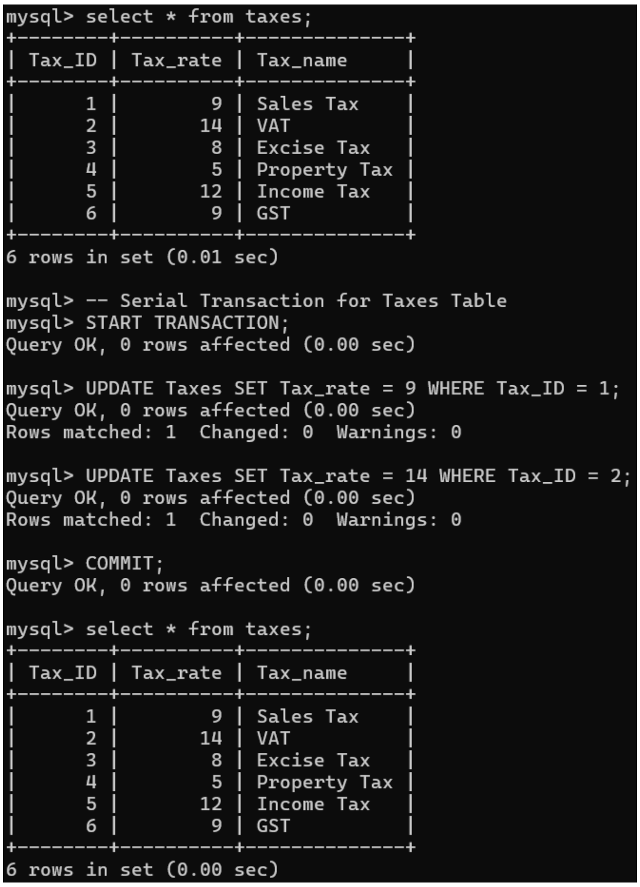
```
mysql> select * from bills;
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
| Bill_no | Staff_ID | Item1       | Item2           | Item3    | Item4       | Item5 | Amount | DoS        | Cust_ph    | Discount |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
|       3 |        3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 | 1234567892 |        0 |
|       4 |        4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 | 1234567893 |       25 |
|       5 |        5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 | 1234567894 |       10 |
|       6 |        3 | Groceries   | Electronics     | NULL     | NULL        | NULL  |  20000 | 2024-04-29 | 9005571394 |      100 |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
4 rows in set (0.01 sec)

mysql> -- Concurrent Transaction 1 for Bills Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Bills SET Discount = 5 WHERE Bill_no = 3;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```



```
mysql> -- Concurrent Transaction 2 for Bills Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Bills SET Discount = 20 WHERE Bill_no = 4;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from bills;
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
| Bill_no | Staff_ID | Item1       | Item2           | Item3    | Item4       | Item5 | Amount | DoS        | Cust_ph    | Discount |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
|       3 |        3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 | 1234567892 |        5 |
|       4 |        4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 | 1234567893 |       20 |
|       5 |        5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 | 1234567894 |       10 |
|       6 |        3 | Groceries   | Electronics     | NULL     | NULL        | NULL  |  20000 | 2024-04-29 | 9005571394 |      100 |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
4 rows in set (0.00 sec)
```

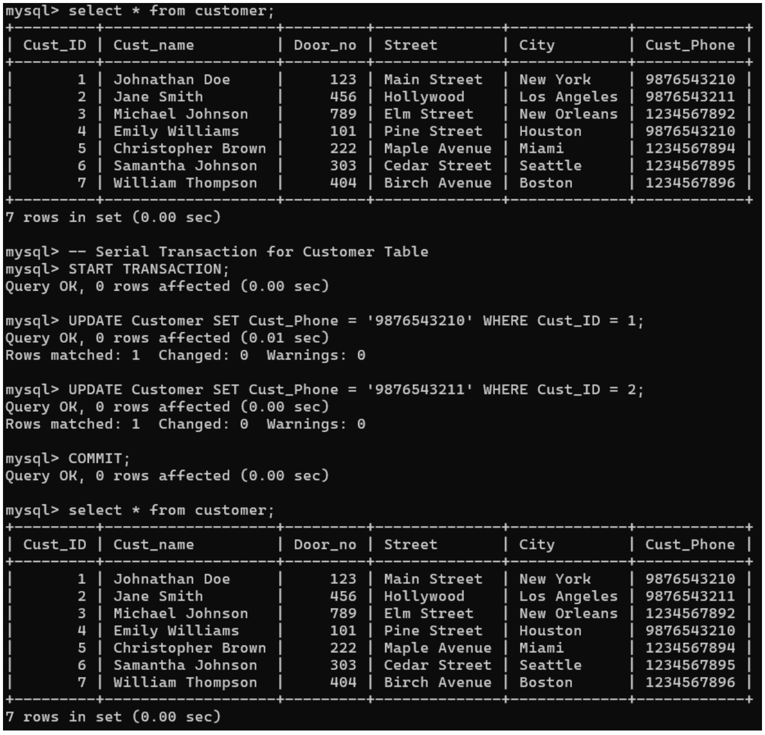
SERIAL TRANSACTIONS

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



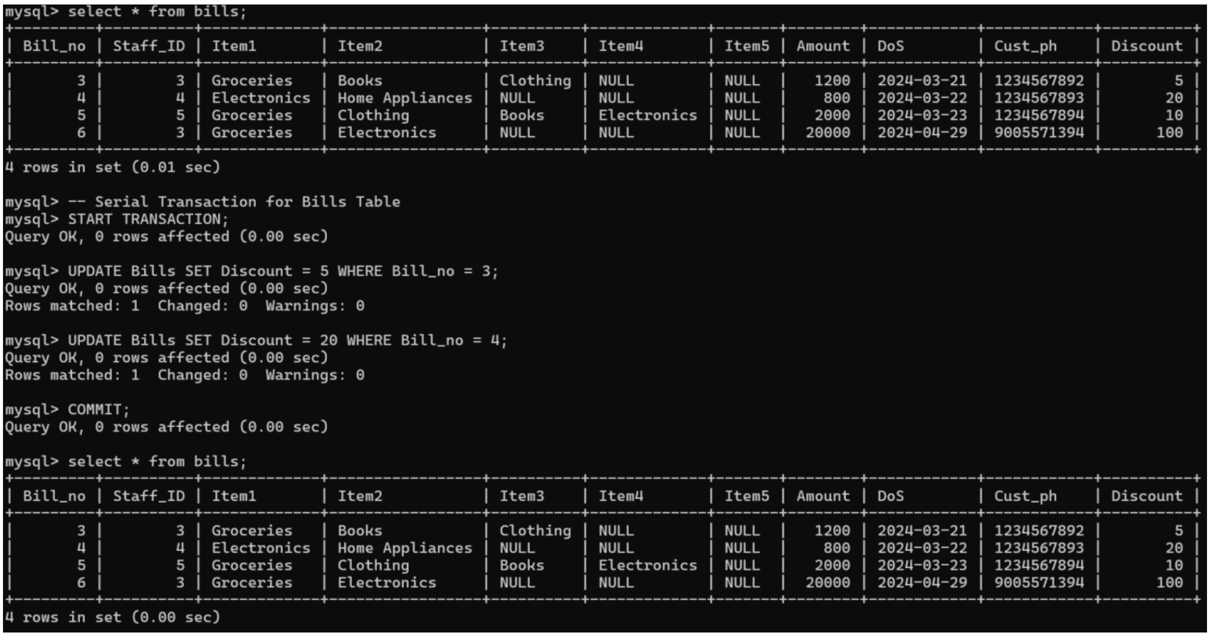
```
mysql> select * from taxes;
+--------+----------+--------------+
| Tax_ID | Tax_rate | Tax_name     |
+--------+----------+--------------+
|      1 |        9 | Sales Tax    |
|      2 |       14 | VAT          |
|      3 |        8 | Excise Tax   |
|      4 |        5 | Property Tax |
|      5 |       12 | Income Tax   |
|      6 |        9 | GST          |
+--------+----------+--------------+
6 rows in set (0.01 sec)

mysql> -- Serial Transaction for Taxes Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Taxes SET Tax_rate = 9 WHERE Tax_ID = 1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> UPDATE Taxes SET Tax_rate = 14 WHERE Tax_ID = 2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from taxes;
+--------+----------+--------------+
| Tax_ID | Tax_rate | Tax_name     |
+--------+----------+--------------+
|      1 |        9 | Sales Tax    |
|      2 |       14 | VAT          |
|      3 |        8 | Excise Tax   |
|      4 |        5 | Property Tax |
|      5 |       12 | Income Tax   |
|      6 |        9 | GST          |
+--------+----------+--------------+
6 rows in set (0.00 sec)
```

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



```
mysql> select * from customer;
+---------+-------------------+---------+--------------+-------------+------------+
| Cust_ID | Cust_name         | Door_no | Street       | City        | Cust_Phone |
+---------+-------------------+---------+--------------+-------------+------------+
|       1 | Johnathan Doe     |     123 | Main Street  | New York    | 9876543210 |
|       2 | Jane Smith        |     456 | Hollywood    | Los Angeles | 9876543211 |
|       3 | Michael Johnson   |     789 | Elm Street   | New Orleans | 1234567892 |
|       4 | Emily Williams    |     101 | Pine Street  | Houston     | 9876543210 |
|       5 | Christopher Brown |     222 | Maple Avenue | Miami       | 1234567894 |
|       6 | Samantha Johnson  |     303 | Cedar Street | Seattle     | 1234567895 |
|       7 | William Thompson  |     404 | Birch Avenue | Boston      | 1234567896 |
+---------+-------------------+---------+--------------+-------------+------------+
7 rows in set (0.00 sec)

mysql> -- Serial Transaction for Customer Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Customer SET Cust_Phone = '9876543210' WHERE Cust_ID = 1;
Query OK, 0 rows affected (0.01 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> UPDATE Customer SET Cust_Phone = '9876543211' WHERE Cust_ID = 2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customer;
+---------+-------------------+---------+--------------+-------------+------------+
| Cust_ID | Cust_name         | Door_no | Street       | City        | Cust_Phone |
+---------+-------------------+---------+--------------+-------------+------------+
|       1 | Johnathan Doe     |     123 | Main Street  | New York    | 9876543210 |
|       2 | Jane Smith        |     456 | Hollywood    | Los Angeles | 9876543211 |
|       3 | Michael Johnson   |     789 | Elm Street   | New Orleans | 1234567892 |
|       4 | Emily Williams    |     101 | Pine Street  | Houston     | 9876543210 |
|       5 | Christopher Brown |     222 | Maple Avenue | Miami       | 1234567894 |
|       6 | Samantha Johnson  |     303 | Cedar Street | Seattle     | 1234567895 |
|       7 | William Thompson  |     404 | Birch Avenue | Boston      | 1234567896 |
+---------+-------------------+---------+--------------+-------------+------------+
7 rows in set (0.00 sec)
```

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



```
mysql> select * from bills;
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
| Bill_no | Staff_ID | Item1       | Item2           | Item3    | Item4       | Item5 | Amount | DoS        | Cust_ph    | Discount |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
|       3 |        3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 | 1234567892 |        5 |
|       4 |        4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 | 1234567893 |       20 |
|       5 |        5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 | 1234567894 |       10 |
|       6 |        3 | Groceries   | Electronics     | NULL     | NULL        | NULL  |  20000 | 2024-04-29 | 9005571394 |      100 |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
4 rows in set (0.01 sec)

mysql> -- Serial Transaction for Bills Table
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Bills SET Discount = 5 WHERE Bill_no = 3;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> UPDATE Bills SET Discount = 20 WHERE Bill_no = 4;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from bills;
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
| Bill_no | Staff_ID | Item1       | Item2           | Item3    | Item4       | Item5 | Amount | DoS        | Cust_ph    | Discount |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
|       3 |        3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 | 1234567892 |        5 |
|       4 |        4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 | 1234567893 |       20 |
|       5 |        5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 | 1234567894 |       10 |
|       6 |        3 | Groceries   | Electronics     | NULL     | NULL        | NULL  |  20000 | 2024-04-29 | 9005571394 |      100 |
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+------------+----------+
4 rows in set (0.00 sec)
```

Chapter 10

API USING PYTHON

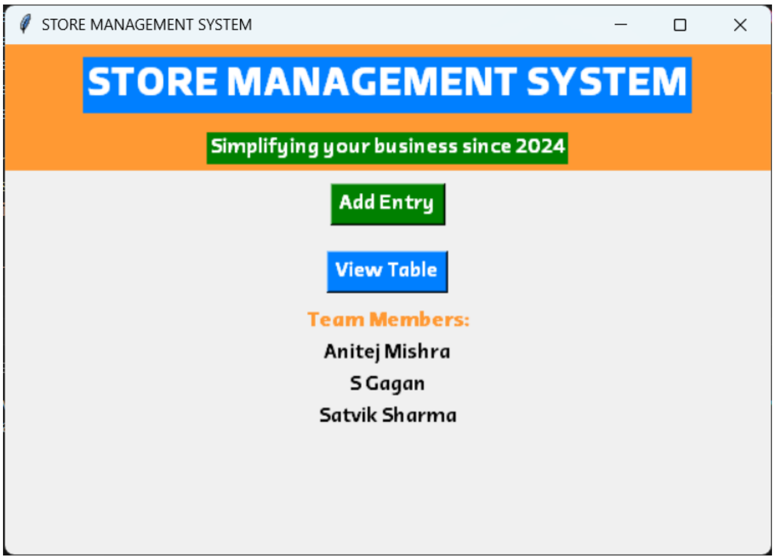
To make our SMS project easy to use, even for those who don’t have much knowledge about using computers, we have created a Python application, which connects to the SMS database.

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



STORE MANAGEMENT SYSTEM

STORE MANAGEMENT SYSTEM

Simplifying your business since 2024

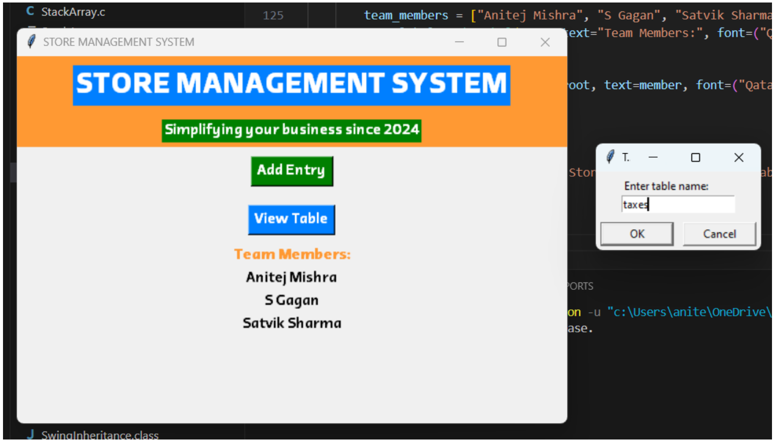
Team Members:

Anitej Mishra

S Gagan

Satvik Sharma

- Add Entry
  - Entering the table name…



```
team_members = ["Anitej Mishra", "S Gagan", "Satvik Sharma
ext="Team Members:", font=("Q
oot, text=member, font=("Qata
Stor
on -u "c:\Users\anite\OneDrive\
ase.
```

STORE MANAGEMENT SYSTEM

STORE MANAGEMENT SYSTEM

Simplifying your business since 2024

Team Members:

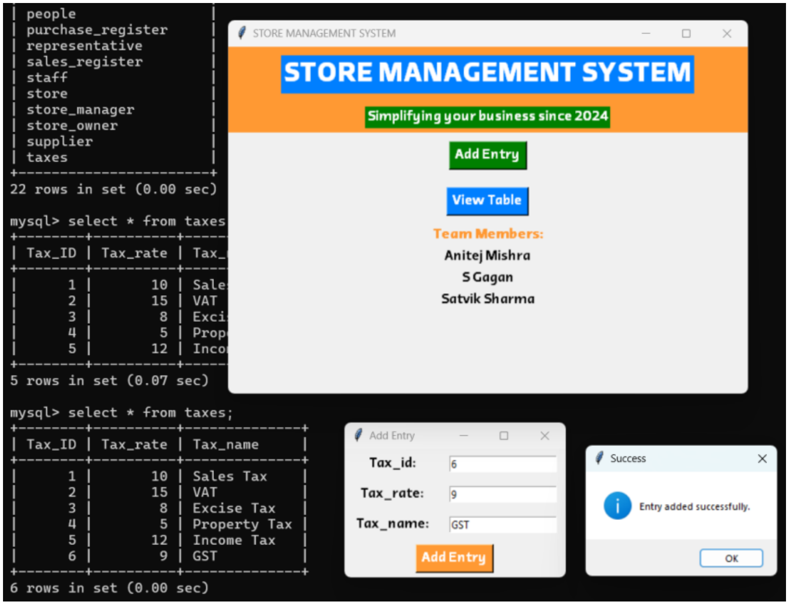
Anitej Mishra

S Gagan

Satvik Sharma

Enter table name: taxes

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

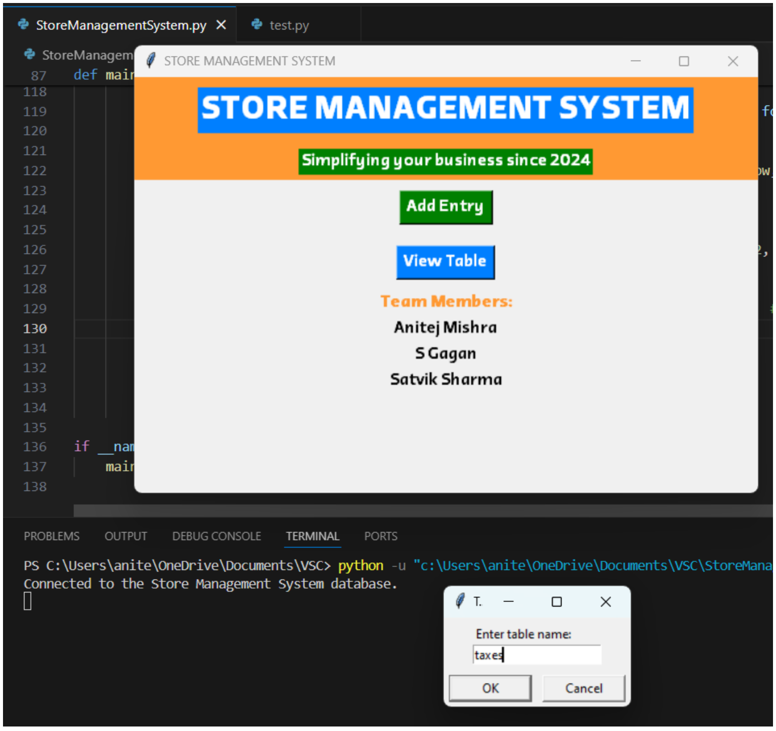


```
| people               |
| purchase_register    |
| representative       |
| sales_register       |
| staff                |
| store                |
| store_manager        |
| store_owner          |
| supplier             |
| taxes                |
+----------------------+
22 rows in set (0.00 sec)

mysql> select * from taxes
+--------+----------+------
| Tax_ID | Tax_rate | Tax_
+--------+----------+------
|      1 |       10 | Sale
|      2 |       15 | VAT
|      3 |        8 | Exci
|      4 |        5 | Prop
|      5 |       12 | Inco
+--------+----------+------
5 rows in set (0.07 sec)

mysql> select * from taxes;
+--------+----------+--------------+
| Tax_ID | Tax_rate | Tax_name     |
+--------+----------+--------------+
|      1 |       10 | Sales Tax    |
|      2 |       15 | VAT          |
|      3 |        8 | Excise Tax   |
|      4 |        5 | Property Tax |
|      5 |       12 | Income Tax   |
|      6 |        9 | GST          |
+--------+----------+--------------+
6 rows in set (0.00 sec)
```

Tax_id: 6
Tax_rate: 9
Tax_name: GST

Entry added successfully.

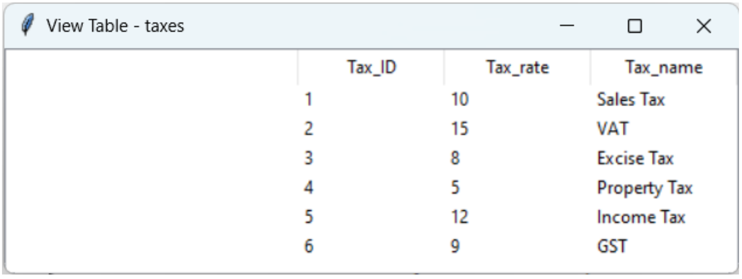
- View Table
  - Entering the table name…



```
PS C:\Users\anite\OneDrive\Documents\VSC> python -u "c:\Users\anite\OneDrive\Documents\VSC\StoreMana
Connected to the Store Management System database.
```

Enter table name: taxes

  - The output…



View Table - taxes

Tax_ID	Tax_rate	Tax_name
1	10	Sales Tax
2	15	VAT
3	8	Excise Tax
4	5	Property Tax
5	12	Income Tax
6	9	GST

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/

2. **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

3. **Retail Store Management System**
   By Srikant Surendra Rout
   Indira Gandhi National Open University
   Published in 2011

4. **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