STORE MANAGEMENT SYSTEM

Database Management Systems
Mini Project

Satvik Sharma [RA2211029010003]

S Gagan [RA2211029010010]

Anitej Mishra [RA2211029010023]

PROBLEM STATEMENT

Develop a Store Management System to streamline inventory tracking, sales recording, and customer management processes for retail stores.

The system should facilitate efficient management of product stock levels, sales transactions recording, and customer information maintenance.

Key functionalities include inventory updates, sales records, and customer management, with the capability to generate insightful reports to aid decision-making.

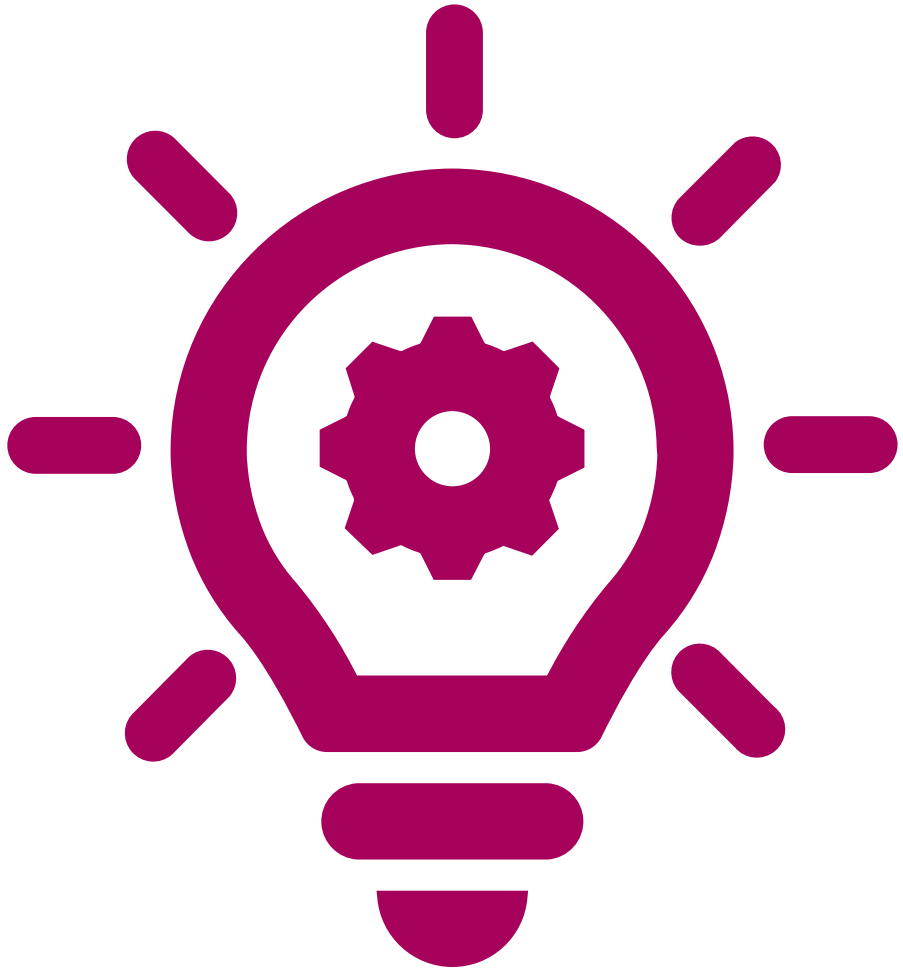
The goal is to enhance operational efficiency, improve customer satisfaction, and optimize inventory management within retail stores.

CONTENTS

DESIGN THINKING APPROACH

- The following are the Design Thinking Tools that have been implemented in our project till now:
- Design the Problem (Empathise)
- Research and Ideation (Define)
- Prototyping (Ideate)
- Implementation

- These are the tools which will be implemented as per the user's usage pattern:
- User Feedback (Prototype)
- Testing
- User Validation
- Documentation
- Reflection and Iteration
- Final Presentation



DESIGN THINKING APPROACH

- DESIGN THE PROBLEM and EMPATHISE

Store owners often resort to pen-and-paper or basic file systems to manage stock, bills, employees, sales, and purchase records, leading to inefficiency and tediousness.

Implementing a robust Store Management System (SMS) is crucial, benefiting the business, store owner, and indirectly, customers in the long run.

- RESEARCH, IDEATION and DEFINE

Businesses utilizing an SMS are notably more efficient and faster compared to those without it. Owners using an SMS experience business growth and customer retention.

All the while those without an SMS observe slowdowns and customer loss. Implementation of an SMS is vital for business sustainability and growth.

DESIGN THINKING APPROACH

- PROTOTYPING

To develop an SMS for this issue, stakeholders such as the owner/manager, suppliers, customers, and store staff first need to be identified.

Utilizing ER diagrams as prototypes, we can assess the current situation's impact on these stakeholders. Implementing an SMS will positively impact them, paving the way for developing and testing a basic SMS solution.

- IMPLEMENTATION

We must determine the optimal approach for developing the SMS and its associated databases.

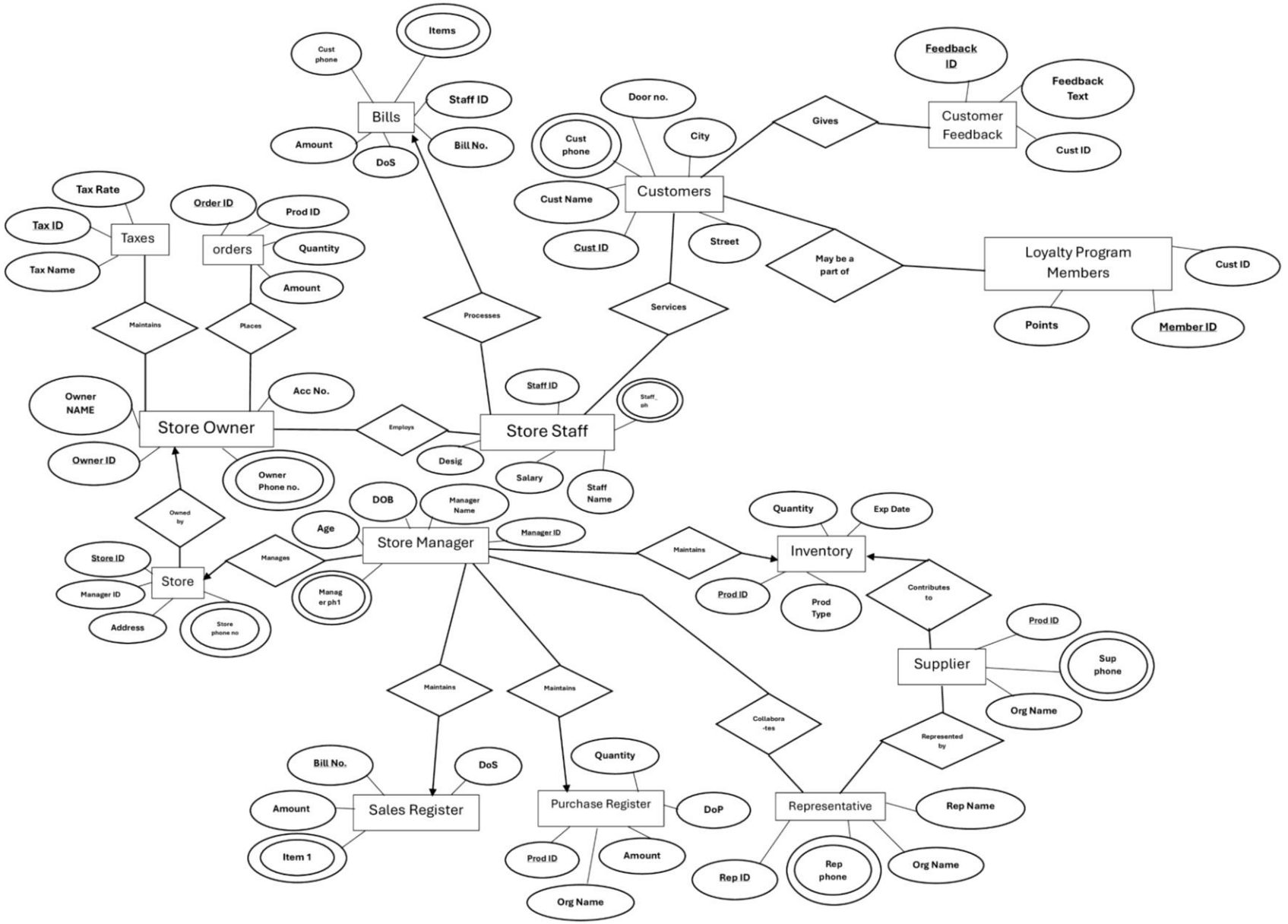
Utilizing ER Diagrams and Database schemas will aid in this process.

For our project, we have selected MySQL as the database management system.

ENTITIES

STORE	STORE MANAGER	STORE OWNER	STORE STAFF	INVENTORY
SUPPLIER	REPRESENTATIVE	TAXES	ORDERS	BILLS
CUSTOMER	CUSTOMER FEEDBACK	LOYALTY PROGRAM MEMBERS	PURCHASE REGISTER	SALES REGISTER

ENTITY-RELATIONSHIP MODEL



Example: Loyalty Members

Creating the Table…

```
CREATE TABLE Loyalty_members (
        Member_ID INT PRIMARY KEY,
        Cust_ID INT,
        Points INT,
        FOREIGN KEY (Cust_ID)
        REFERENCES Customer(Cust_ID)
);
```

Adding Values to it…

```
INSERT INTO Loyalty_members
        (Member_ID,Cust_ID, Points) VALUES
  (1, 1, 100), (2, 2, 50),
  (3, 3, 75),
  (4, 4, 200),
  (5, 5, 150);
```

CREATING TABLES AND ADDING VALUES

Result

After Creating the Table…

```
mysql> desc loyalty_members;
+-----------+------+------+-----+---------+-------+
| Field     | Type | Null | Key | Default | Extra |
+-----------+------+------+-----+---------+-------+
| Member_ID | int  | NO   | PRI | NULL    |       |
| Cust_ID   | int  | YES  | MUL | NULL    |       |
| Points    | int  | YES  |     | NULL    |       |
+-----------+------+------+-----+---------+-------+
3 rows in set (0.08 sec)
```

After Adding Values to it…

```
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)
```

CREATING TABLES AND ADDING VALUES

PL/SQL

Calculate Total Sales of a Customer

```
SQL> CREATE OR REPLACE PROCEDURE CalculateTotalBillAmount(cust_id IN INT)
  2  IS
  3      total_amount NUMBER;
  4  BEGIN
  5      SELECT SUM(b.Amount) INTO total_amount
  6      FROM Bills b
  7      INNER JOIN Customer c ON b.Cust_ph = c.Cust_Phone
  8      WHERE c.Cust_ID = cust_id;
  9
 10      DBMS_OUTPUT.PUT_LINE('Total Bill Amount: ' || total_amount);
 11  END;
 12  /

Procedure created.

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

PL/SQL procedure successfully completed.

SQL>
```

VIEWS

Loyalty Member

```
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>
```

COMPLEX QUERIES

TRIGGERS

Capturing and Displaying Customer Feedback

```
SQL> -- Recreate the trigger to log customer feedback
SQL> CREATE OR REPLACE TRIGGER Log_Customer_Feedback
  2  AFTER INSERT ON Customer_feedback
  3  FOR EACH ROW
  4  BEGIN
  5      INSERT INTO Customer_feedback (Feedback_ID, Cust_ID, Feedback_text)
  6      VALUES (:NEW.Feedback_ID, :NEW.Cust_ID, :NEW.Feedback_text);
  7  END;
  8  /

Trigger created.

SQL>
SQL> -- Recreate the view to display feedback log
SQL> CREATE OR REPLACE VIEW Customer_Feedback_Log AS
  2  SELECT Feedback_ID, Cust_ID, Feedback_text, SYSDATE AS Feedback_date
  3  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>
```

CURSORS

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
```

COMPLEX QUERIES

NORMALIZATION

- Example: Taxes
- Functional Dependencies:
  - Tax_id → Tax_rate
  - Tax_id → Tax_name
- There is a transitive dependency in Tax_id → Tax_name:
  - Tax_id is not super key
  - Tax_name is not prime
- We can apply 3NF and decompose the above table into 'Taxes_names' and 'Taxes_rates'.

```
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)
```

TAXES_NAMES

```
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)
```

TAXES_RATES

```
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)
```

DECOMPOSED & NORMALIZED TABLES

CONCURRENCY CONTROL: CONCURRENT TRANSACTIONS

- 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)
```

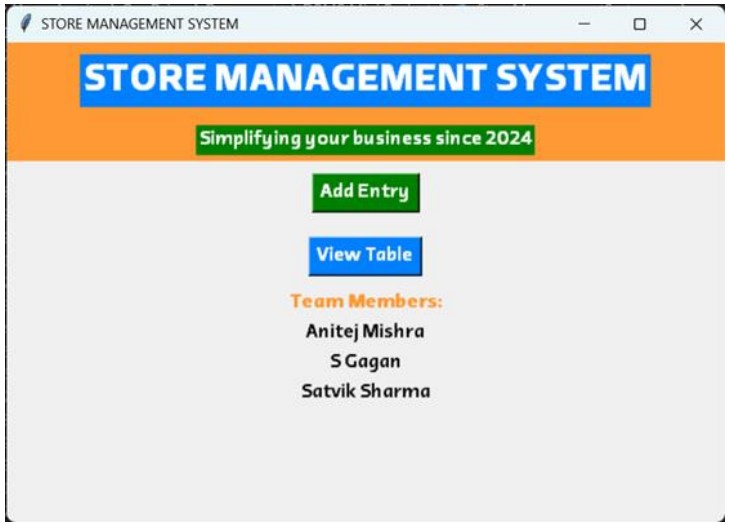
CONCURRENCY CONTROL: SERIAL TRANSACTIONS

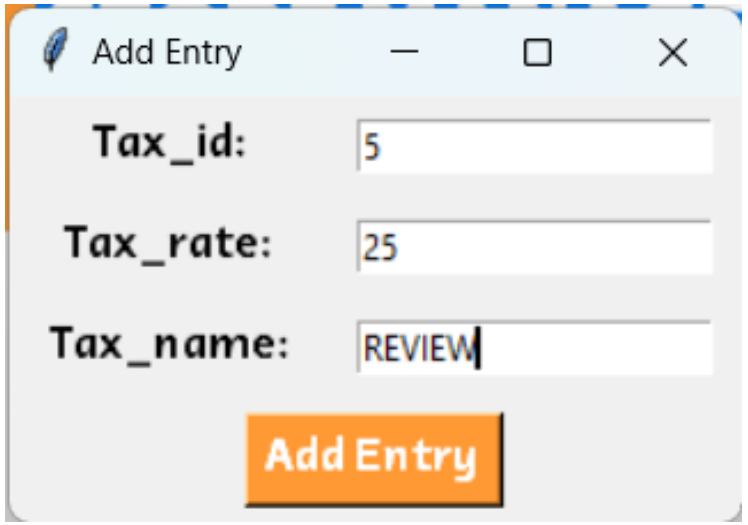
- Update the discount for Bill 3 from 0 to 5, and update the discount for Bill 4 from 25 to 20 in “bills” table

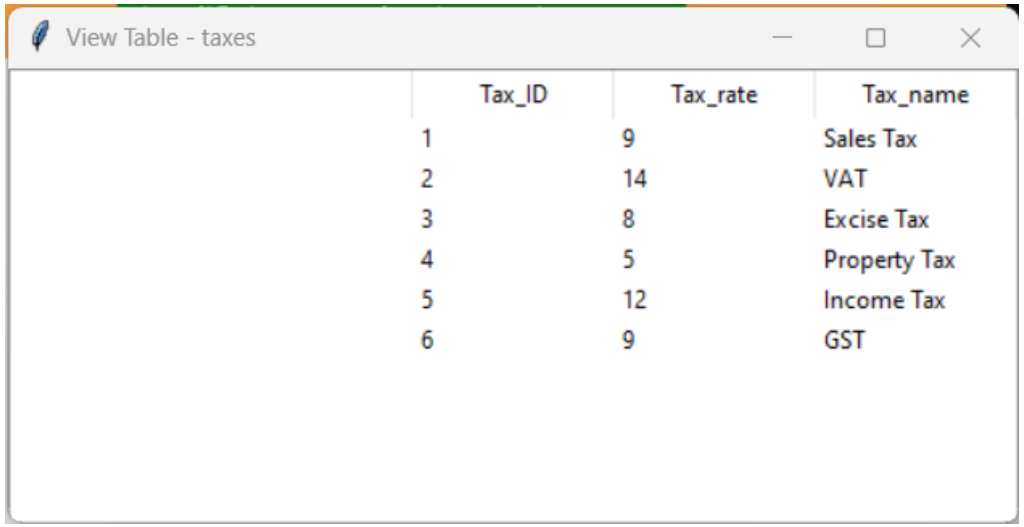
```
mysql> select * from bills;
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+--------
| Bill_no | Staff_ID | Item1       | Item2           | Item3    | Item4       | Item5 | Amount | DoS        | Cust_ph
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+--------
|       3 |        3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 | 123456
|       4 |        4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 | 123456
|       5 |        5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 | 123456
|       6 |        3 | Groceries   | Electronics     | NULL     | NULL        | NULL  |  20000 | 2024-04-29 | 900557
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+--------
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
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+--------
|       3 |        3 | Groceries   | Books           | Clothing | NULL        | NULL  |   1200 | 2024-03-21 | 123456
|       4 |        4 | Electronics | Home Appliances | NULL     | NULL        | NULL  |    800 | 2024-03-22 | 123456
|       5 |        5 | Groceries   | Clothing        | Books    | Electronics | NULL  |   2000 | 2024-03-23 | 123456
|       6 |        3 | Groceries   | Electronics     | NULL     | NULL        | NULL  |  20000 | 2024-04-29 | 900557
+---------+----------+-------------+-----------------+----------+-------------+-------+--------+------------+--------
4 rows in set (0.00 sec)
```

USING PYTHON AND TKINTER

FRONT END AND API







	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

THANK YOU!

By Anitej Mishra, S Gagan and Satvik Sharma