DBMS Mini Project Review

Name: Sumithra Suresh (PES2UG23CS625)

Tanish Hegde (PES2UG23CS641)

Semester: 5th Semester

Section: J

1. Title:

   Cake Database Management System

2. Description:

   This project aims to design and implement a database management system for a Cake Store chain.
   The system maintains data related to customers, orders, cakes, outlets, payments, and employees.

   Our objective is to create a fully functional CRUD-based DBMS system supported by:

- A well-structured ER diagram
- A normalized relational schema
- SQL tables with appropriate constraints
- Triggers, procedures, and functions
- A working front-end interface for interacting with the database

  The application enables outlet staff/admin to manage cake inventory, handle customer orders, track payments, monitor sales, and generate analytical insights using MySQL and a Streamlit-based UI.

3. User Requirement Specification:
   Purpose of The Project:
   The purpose of this project is to create an easy-to-use and well-organized database system that supports the everyday operations of a cake store. It helps the store keep track of customers, manage cake inventory, record orders and payments, and handle information from different outlets—all in one place. With a proper database

structure, the store can avoid manual errors, save time, and run its operations more efficiently.

Scope of the Project:
The scope of this project includes designing the database, creating all the required tables with proper relationships, and adding triggers, procedures, and functions that support the cake store's day-to-day workflow. It manages different types of data—customers, cakes, orders, payments, outlets, and even multiple phone numbers and addresses—so the system can reflect real store operations accurately.

On the front-end side, the project provides a Streamlit app that lets users easily view tables, add new records, place or cancel orders, restock cakes, and check simple analytics. The project focuses on the essential features needed for smooth store management while keeping the overall design simple and suitable for an academic mini-project.

Detailed Description:
To support the growing operations of a multi-outlet cake store, this project focuses on designing a structured and reliable Cake Store Database Management System. At the center of the system is the Cake Catalogue, where each cake is uniquely identified and described through its name, category, price, description, and available quantity. This ensures that every outlet can accurately view and manage the cakes it offers.
Each Outlet—identified by an Outlet ID—stores details such as its name, address, and phone numbers. Outlets receive cake catalogues and handle customer orders, making them a key part of the store's workflow. Outlets are staffed by Employees, each identified with an Employee ID and associated with attributes like their name, role, and phone number. An employee works at exactly one outlet, helping maintain clear accountability and easy staff management.
The system also includes Customers, who are uniquely identified using a Customer ID. Each customer's name, email, phone number, and address are stored, allowing the outlet to deliver the order correctly and maintain accurate customer records. Customers interact with the system by placing Orders, which record details such as order date, total amount, and status (e.g., Completed, Pending, or Cancelled). Every order connects three major entities — the customer who placed it, the cake being purchased, and the outlet fulfilling it.
Payments are captured separately through the Payment entity. Each payment includes the amount, method of payment, and payment date, and is associated with a single order. This ensures proper financial tracking and avoids mismatches between completed orders and their payments.

4. List of Software/Tools/Programming languages used:

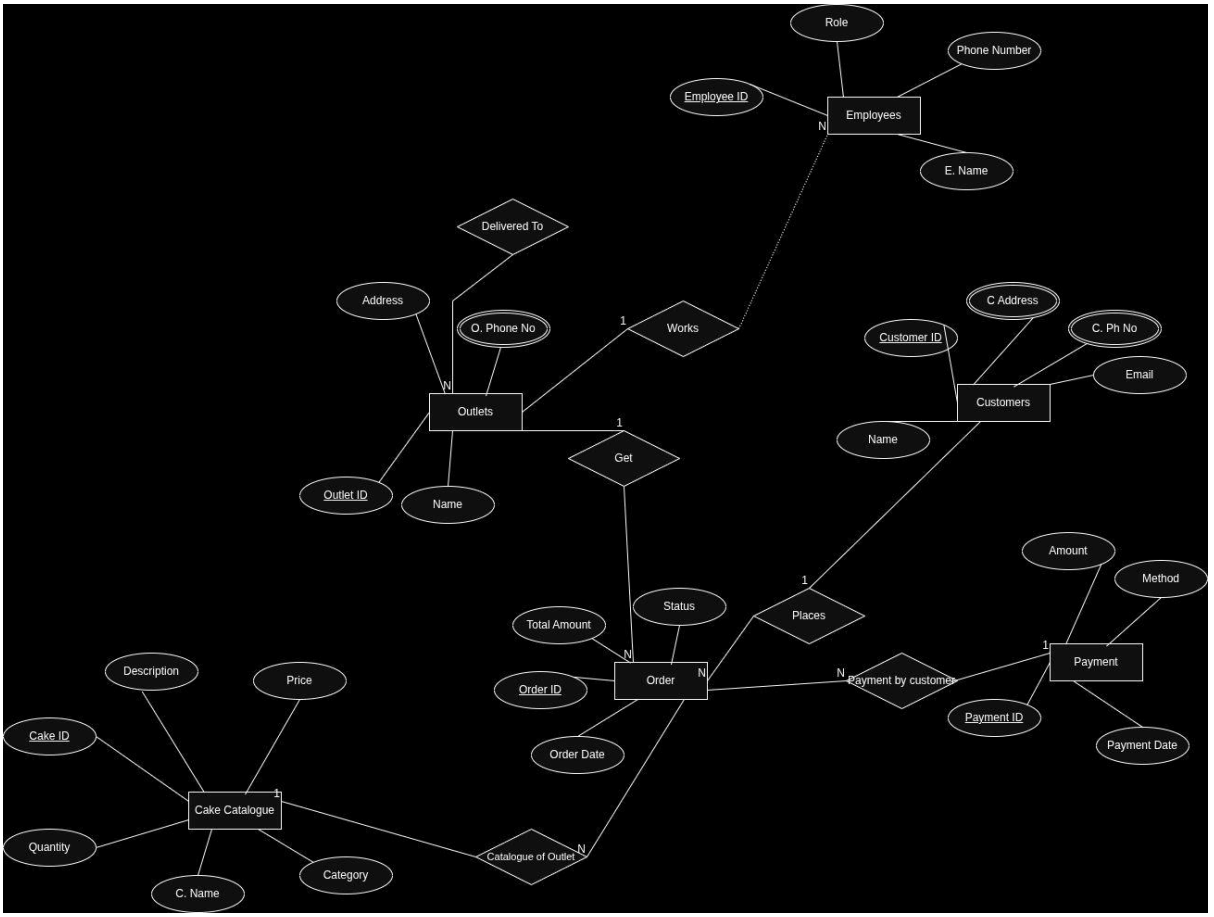
   Streamlist – Python

   MySQL

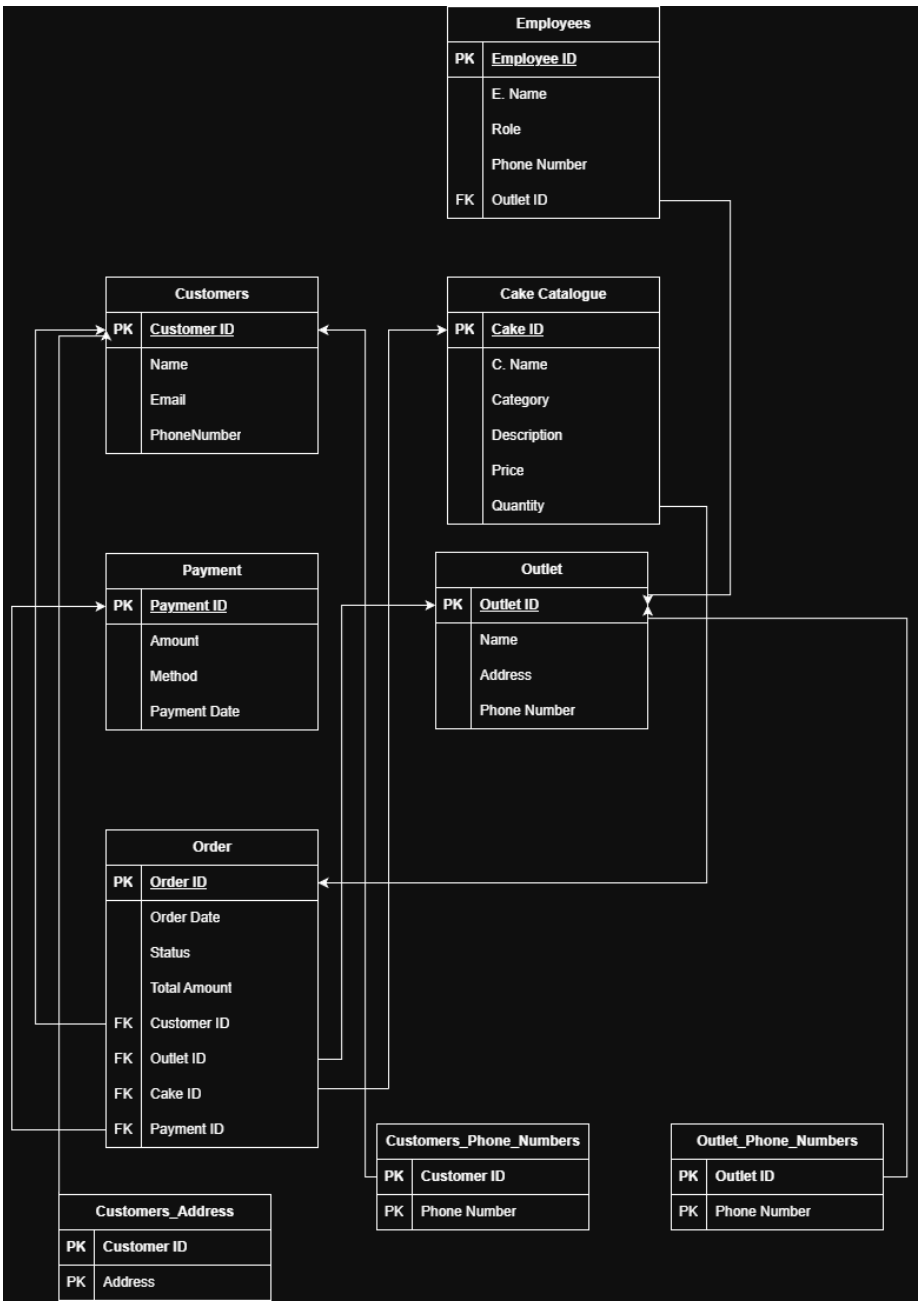
   Pandas

   Draw.io

5. ER Diagram



6. Relational Schema



7. DDL Commands

```sql
CREATE DATABASE IF NOT EXISTS Cake_Store;
USE Cake_Store;

-- ==========================
-- Tables
-- ==========================

CREATE TABLE Payment (
    Payment_ID INT PRIMARY KEY,
    Amount DECIMAL(10,2),
    Method VARCHAR(50),
    Payment_Date DATE
);

CREATE TABLE Outlet (
    Outlet_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Address VARCHAR(200),
    Phone_Number VARCHAR(15)
);
```

```sql
CREATE TABLE Employees (
    Employee_ID INT PRIMARY KEY,
    E_Name VARCHAR(100),
    RoleOfPerson VARCHAR(50),
    Phone_Number VARCHAR(15),
    Outlet_ID INT,
    Production_ID INT,
    FOREIGN KEY (Outlet_ID) REFERENCES Outlet(Outlet_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Customers (
    Customer_ID INT PRIMARY KEY,
    NameOfPerson VARCHAR(100),
    Email VARCHAR(100),
    PhoneNumber VARCHAR(15)
);
```

```sql
CREATE TABLE Cake_Catalogue (
    Cake_ID INT PRIMARY KEY AUTO_INCREMENT,
    C_Name VARCHAR(100),
    C_Category VARCHAR(50),
    C_Description TEXT,
    Price DECIMAL(8,2),
    Quantity INT
);
```

```sql
CREATE TABLE Order_Table (
    Order_ID INT PRIMARY KEY AUTO_INCREMENT,
    Order_Date DATE,
    StatusOrder VARCHAR(20),
    Total_Amount DECIMAL(10,2),
    Customer_ID INT,
    Outlet_ID INT,
    Cake_ID INT,
    Payment_ID INT,
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Outlet_ID) REFERENCES Outlet(Outlet_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
     FOREIGN KEY (Cake_ID) REFERENCES Cake_Catalogue(Cake_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
     FOREIGN KEY (Payment_ID) REFERENCES Payment(Payment_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

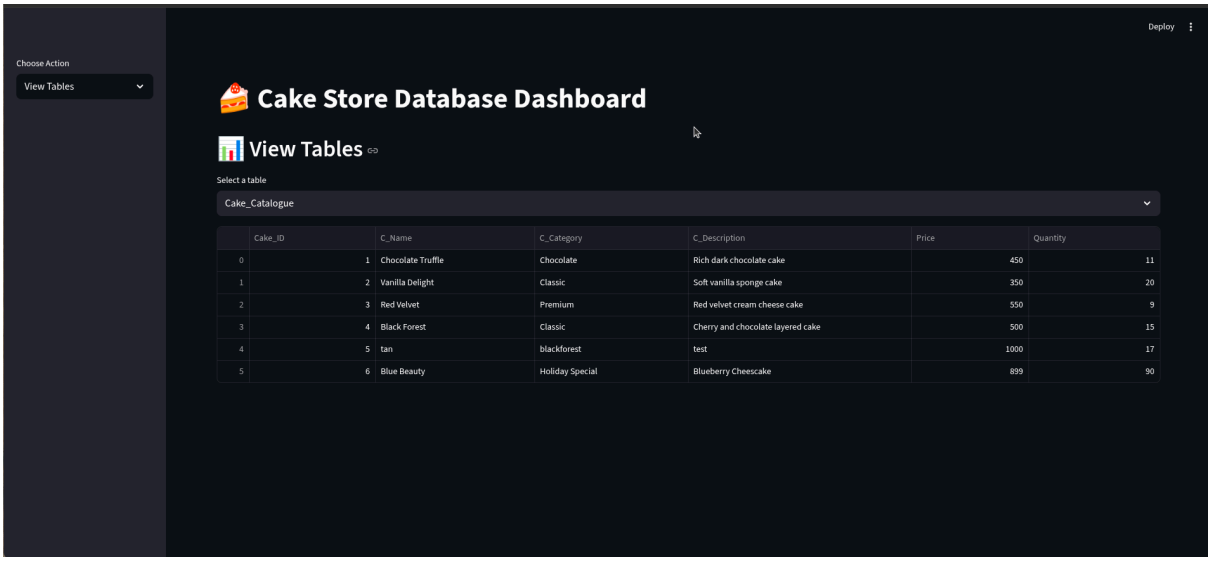
```sql
CREATE TABLE Customers_Address (
    Customer_ID INT,
    Address VARCHAR(200),
    PRIMARY KEY (Customer_ID, Address),
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Customers_Phone_Numbers (
    Customer_ID INT,
    Phone_Number VARCHAR(15),
    PRIMARY KEY (Customer_ID, Phone_Number),
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```sql
CREATE TABLE Outlet_Phone_Numbers (
    Outlet_ID INT,
    Phone_Number VARCHAR(15),
    PRIMARY KEY (Outlet_ID, Phone_Number),
    FOREIGN KEY (Outlet_ID) REFERENCES Outlet(Outlet_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

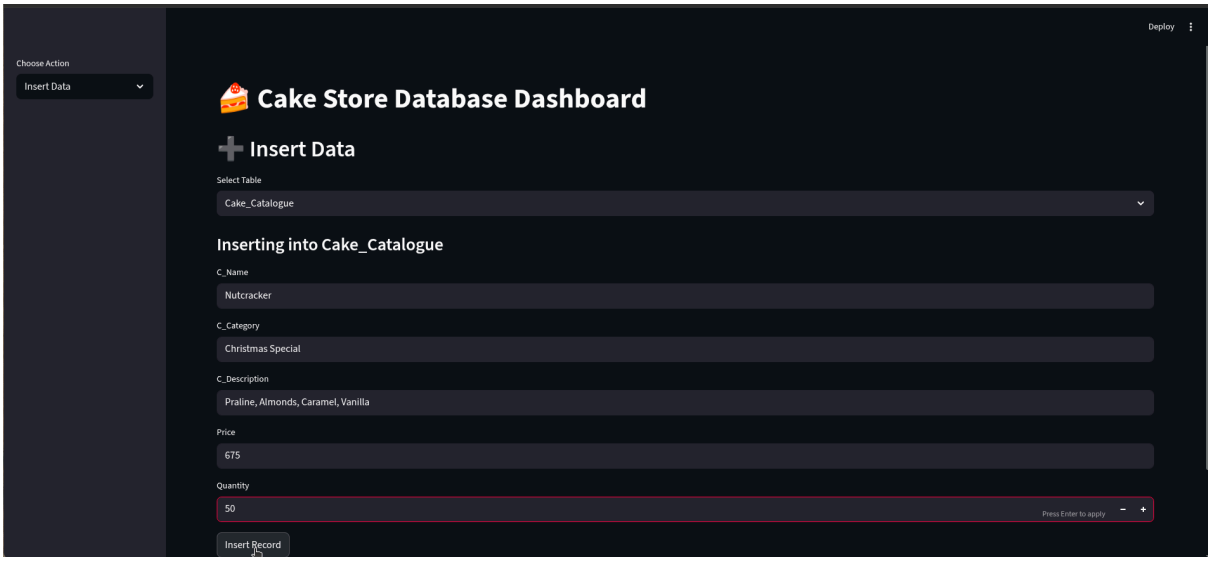
8. CRUD operation Screenshots

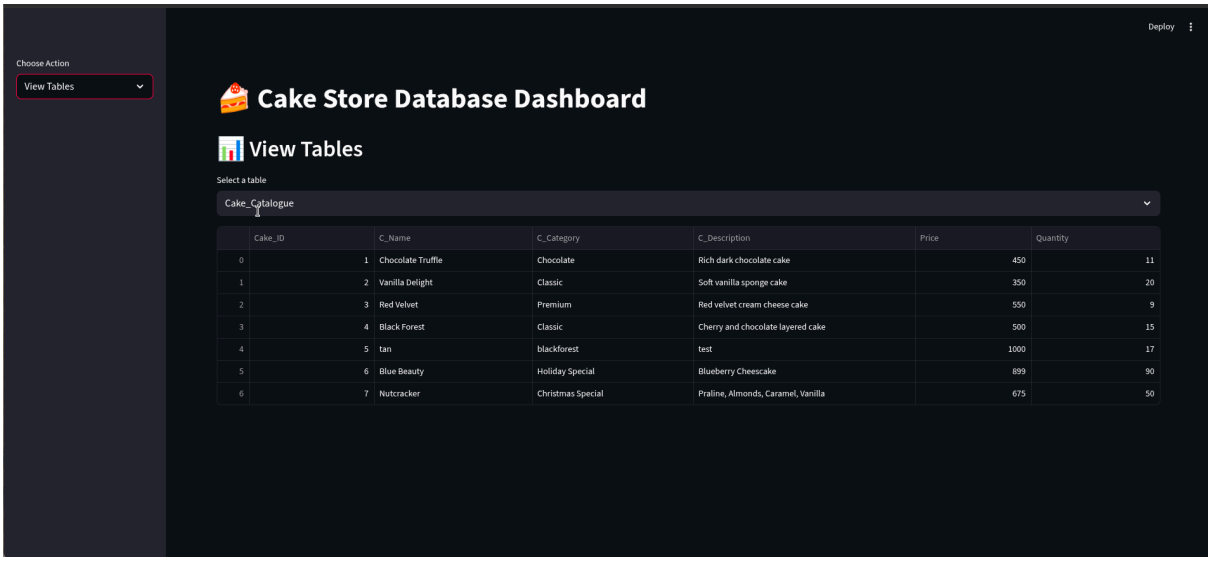
View Tables



	Cake_ID	C_Name	C_Category	C_Description	Price	Quantity
0	1	Chocolate Truffle	Chocolate	Rich dark chocolate cake	450	11
1	2	Vanilla Delight	Classic	Soft vanilla sponge cake	350	20
2	3	Red Velvet	Premium	Red velvet cream cheese cake	550	9
3	4	Black Forest	Classic	Cherry and chocolate layered cake	500	15
4	5	tan	blackforest	test	1000	17
5	6	Blue Beauty	Holiday Special	Blueberry Cheescake	899	90

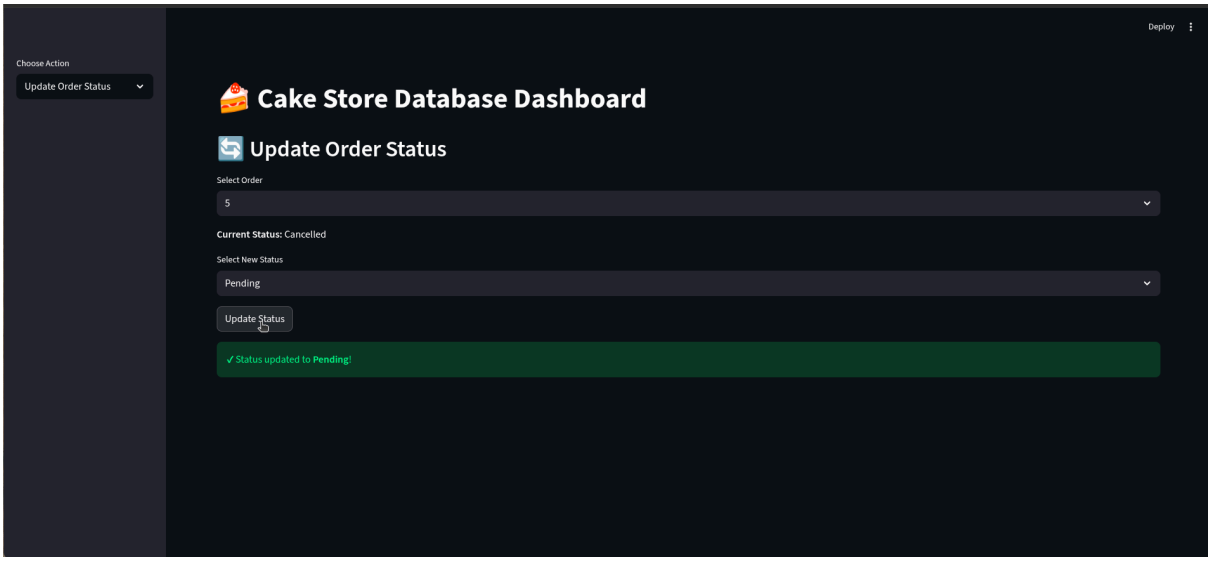
Insert Data

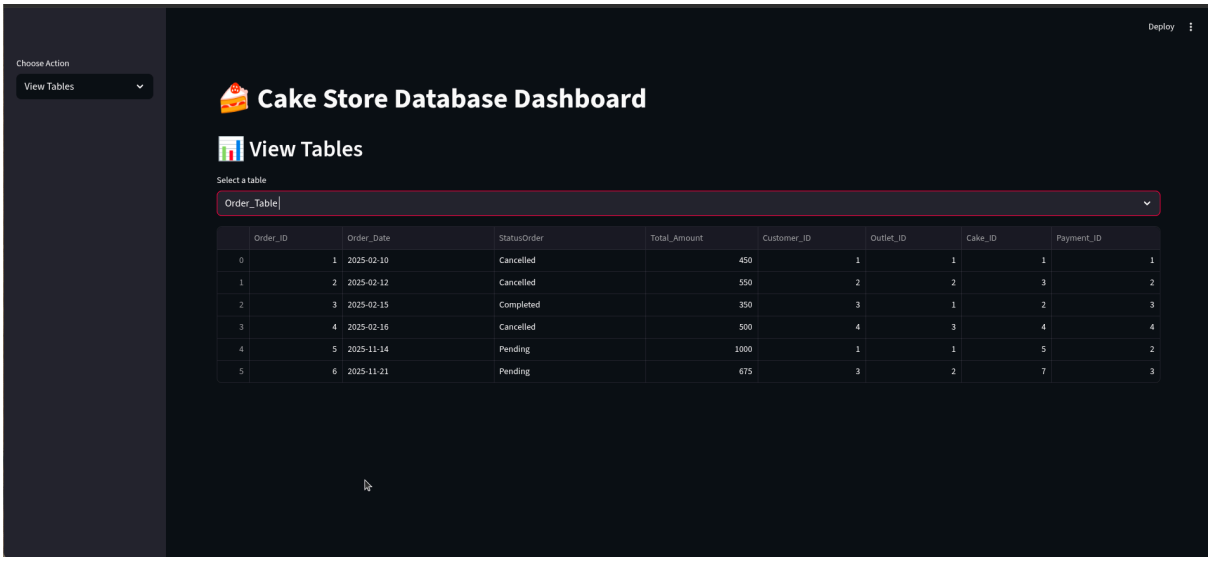




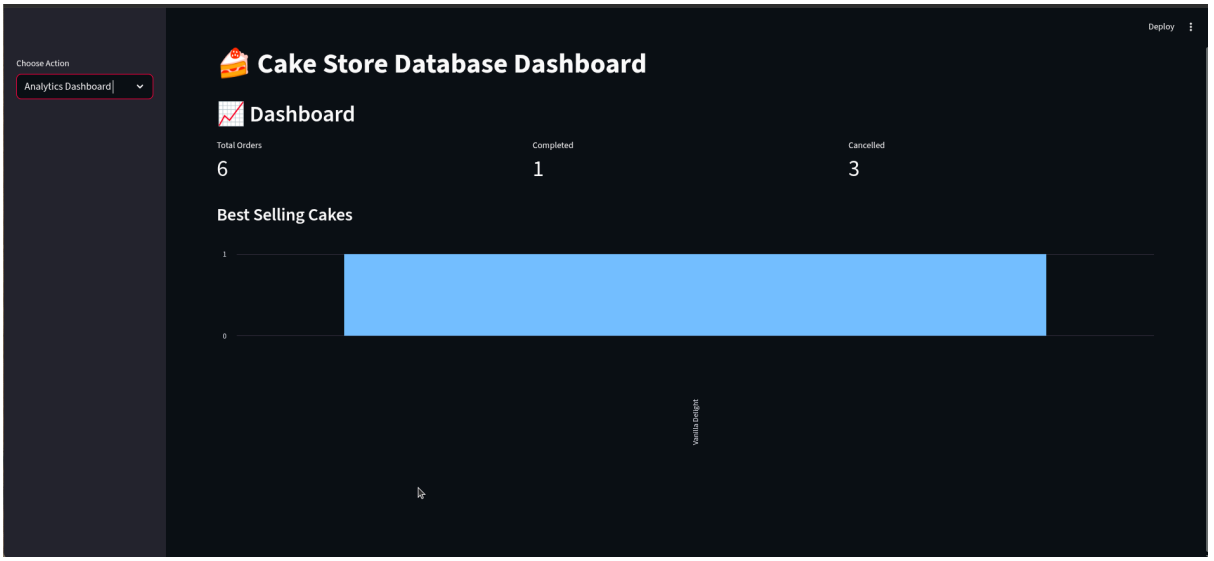
	Cake_ID	C_Name	C_Category	C_Description	Price	Quantity
0	1	Chocolate Truffle	Chocolate	Rich dark chocolate cake	450	11
1	2	Vanilla Delight	Classic	Soft vanilla sponge cake	350	20
2	3	Red Velvet	Premium	Red velvet cream cheese cake	550	9
3	4	Black Forest	Classic	Cherry and chocolate layered cake	500	15
4	5	tan	blackforest	test	1000	17
5	6	Blue Beauty	Holiday Special	Blueberry Cheescake	899	90
6	7	Nutcracker	Christmas Special	Praline, Almonds, Caramel, Vanilla	675	50

Update Order Status

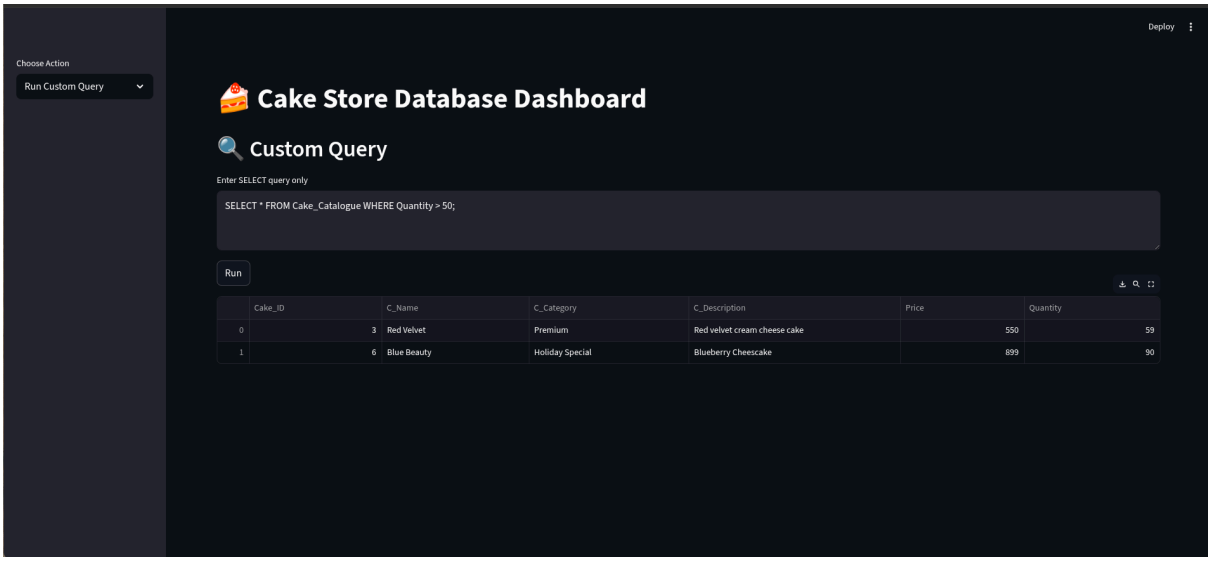




Analytics Dashboard

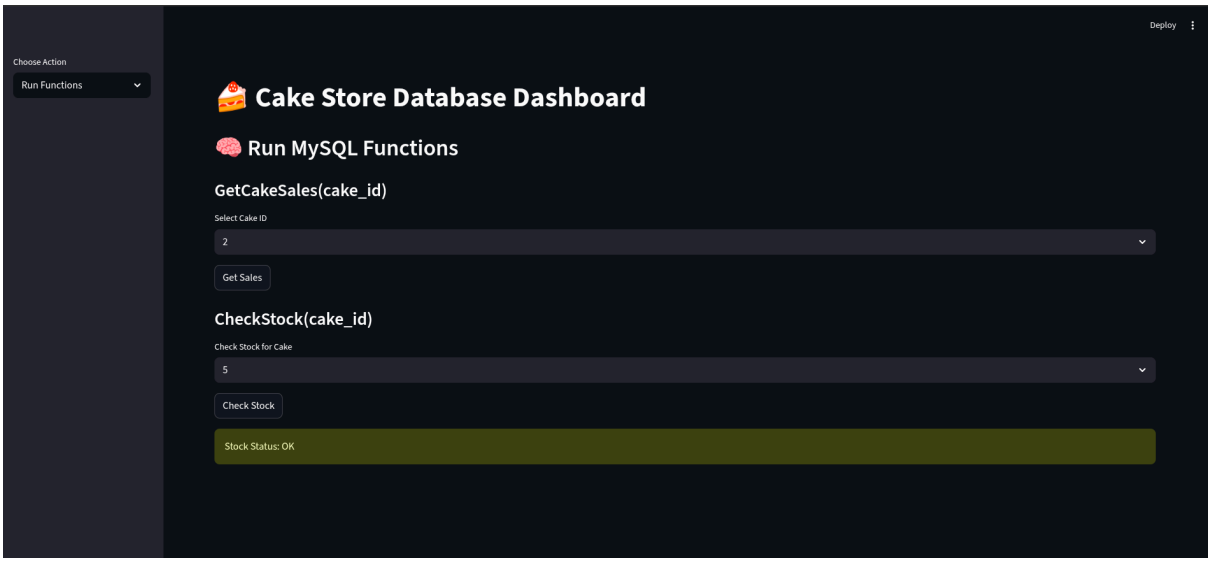


Custom Query

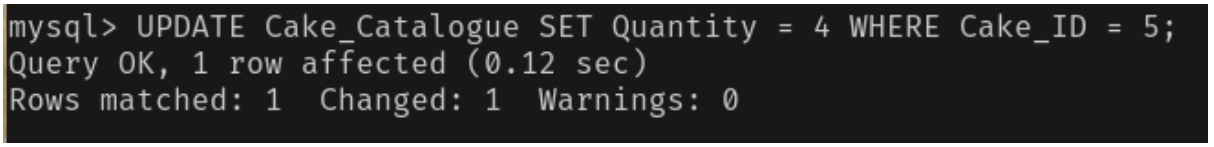


9. List of functionalities/features of the application and its associated screenshots using front end

Functions



If Stock is less than 5

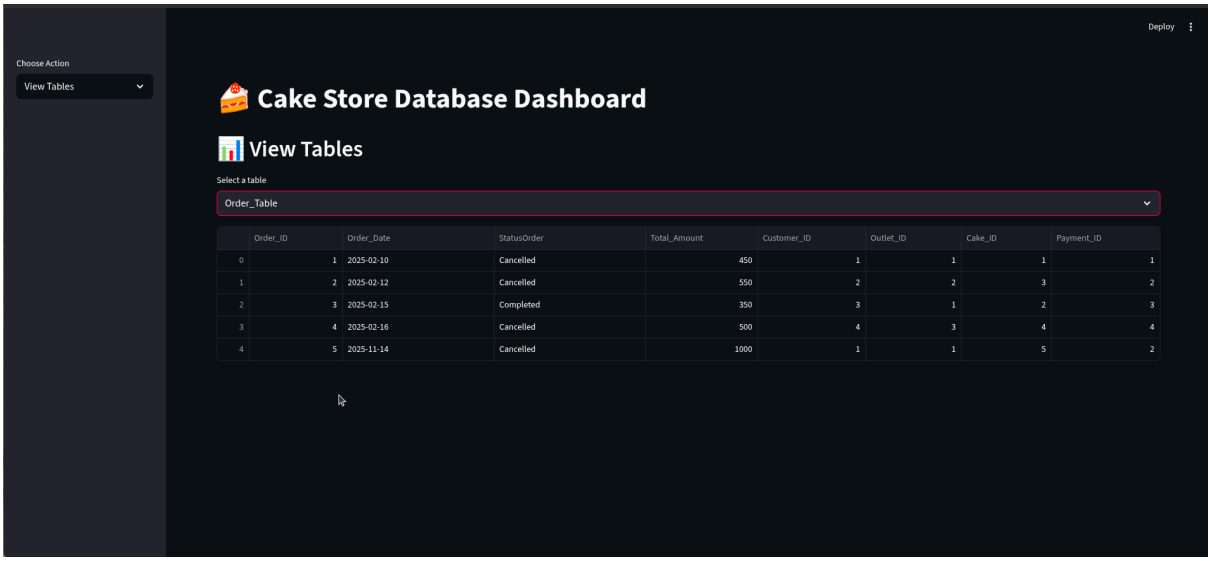


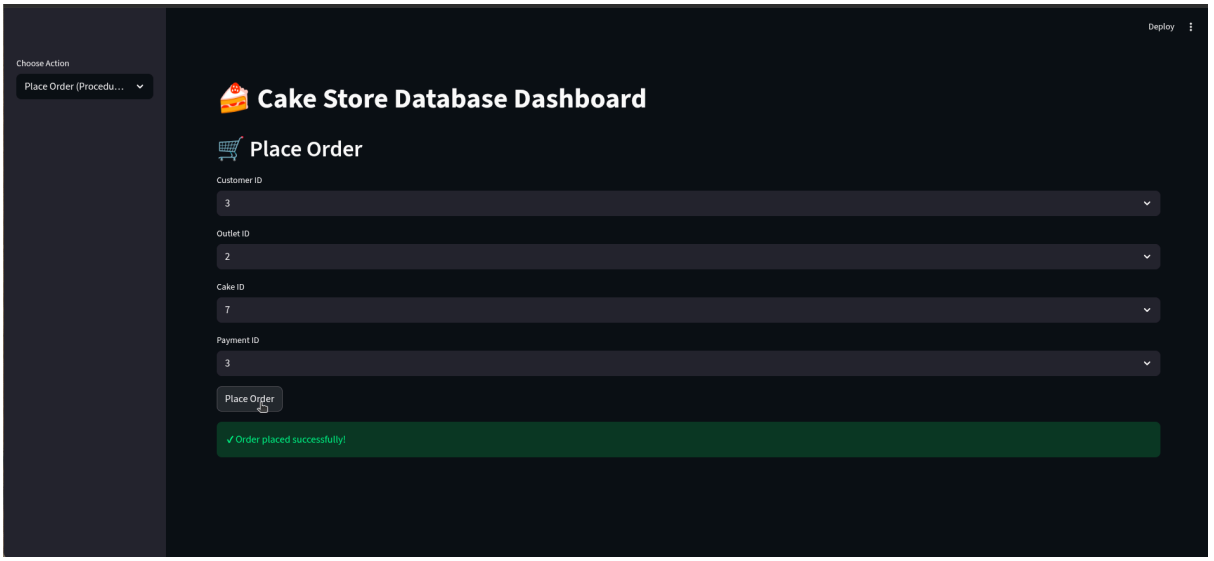
If Stock is 0 or negative

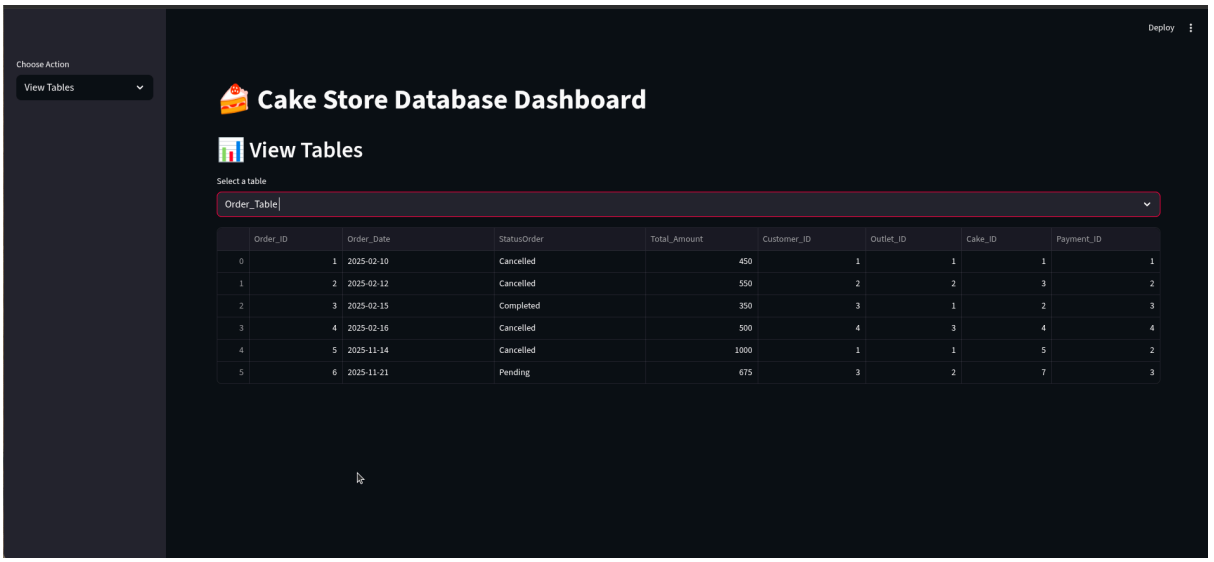
```
mysql> UPDATE Cake_Catalogue SET Quantity = 0 WHERE Cake_ID = 5;
Query OK, 1 row affected (0.14 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Procedures

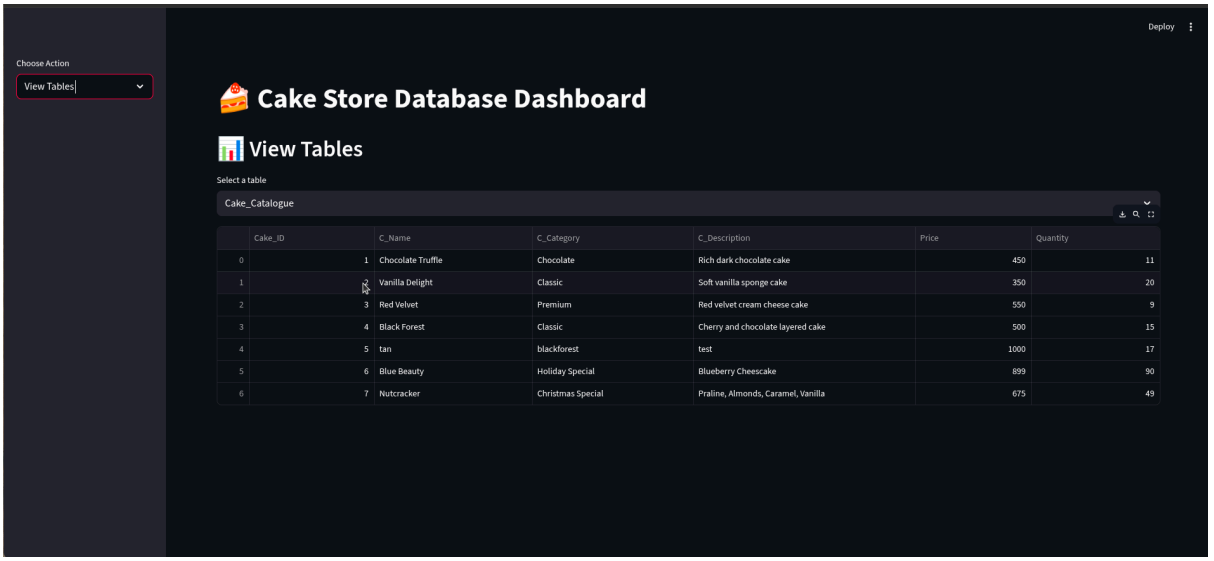
Place Order

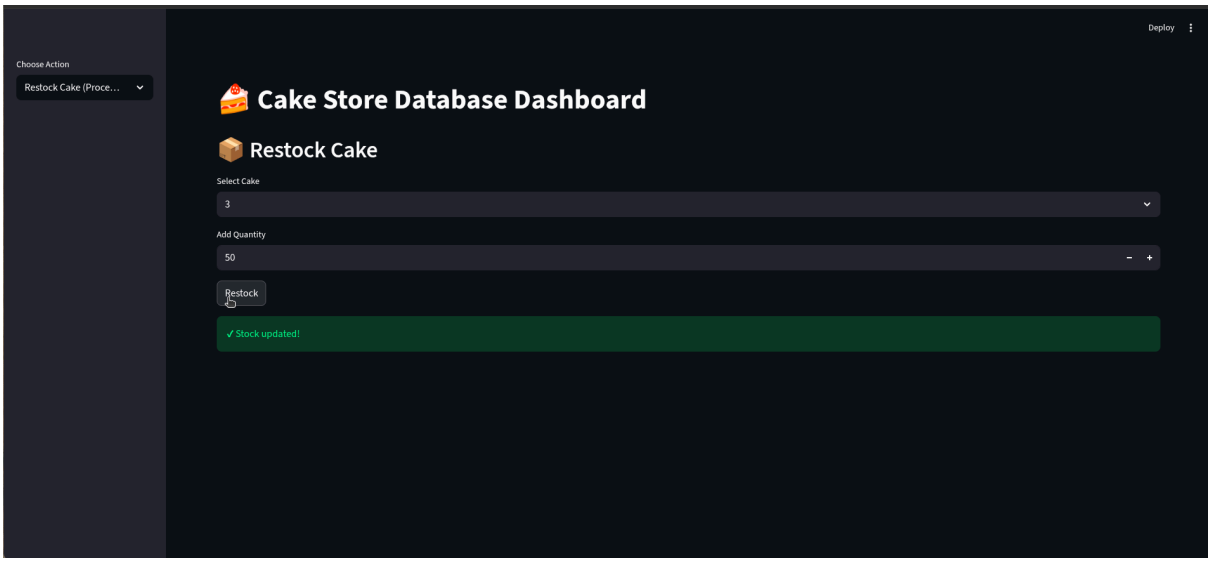


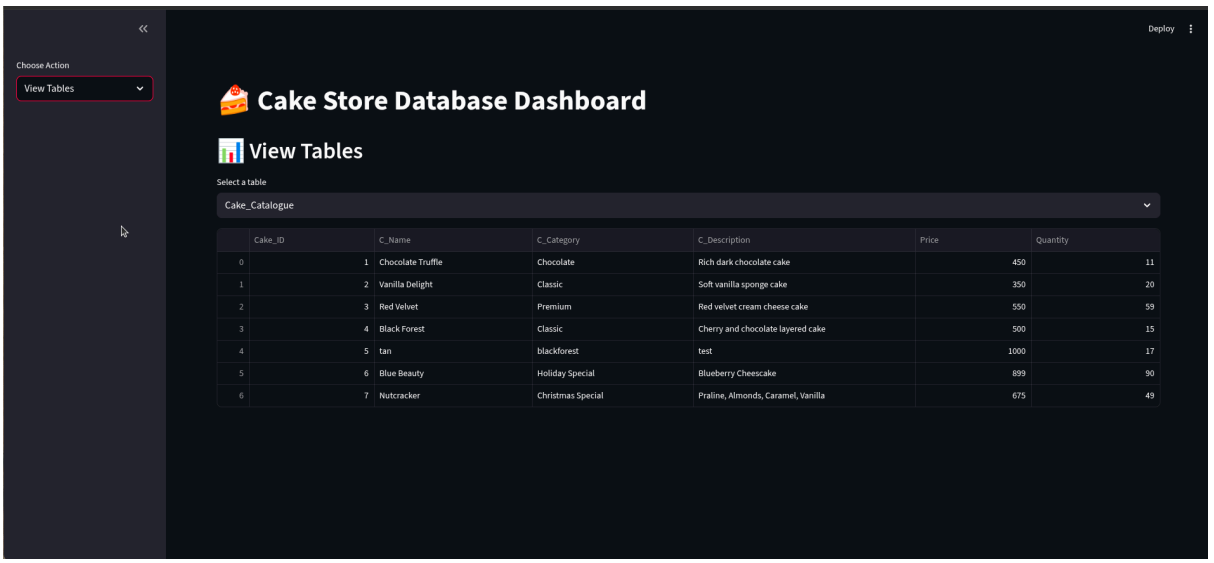

Deploy
Choose Action
Place Order (Procedu...
Cake Store Database Dashboard
Place Order
Customer ID
3
Outlet ID
2
Cake ID
7
Payment ID
3
Place Order
✓ Order placed successfully!


Deploy
Choose Action
View Tables
Cake Store Database Dashboard
View Tables
Select a table
Order_Table
Order_ID Order_Date StatusOrder Total_Amount Customer_ID Outlet_ID Cake_ID Payment_ID
0 1 2025-02-10 Cancelled 450 1 1 1 1
1 2 2025-02-12 Cancelled 550 2 2 3 2
2 3 2025-02-15 Completed 350 3 1 2 3
3 4 2025-02-16 Cancelled 500 4 3 4 4
4 5 2025-11-14 Cancelled 1000 1 1 5 2
5 6 2025-11-21 Pending 675 3 2 7 3

Restock Cake





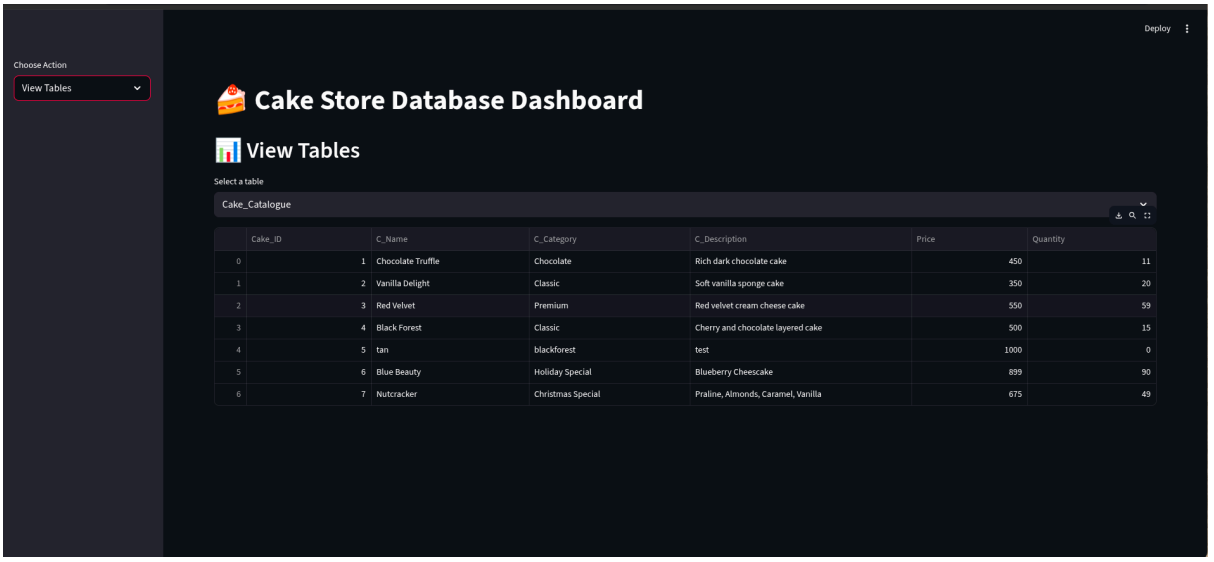


Triggers

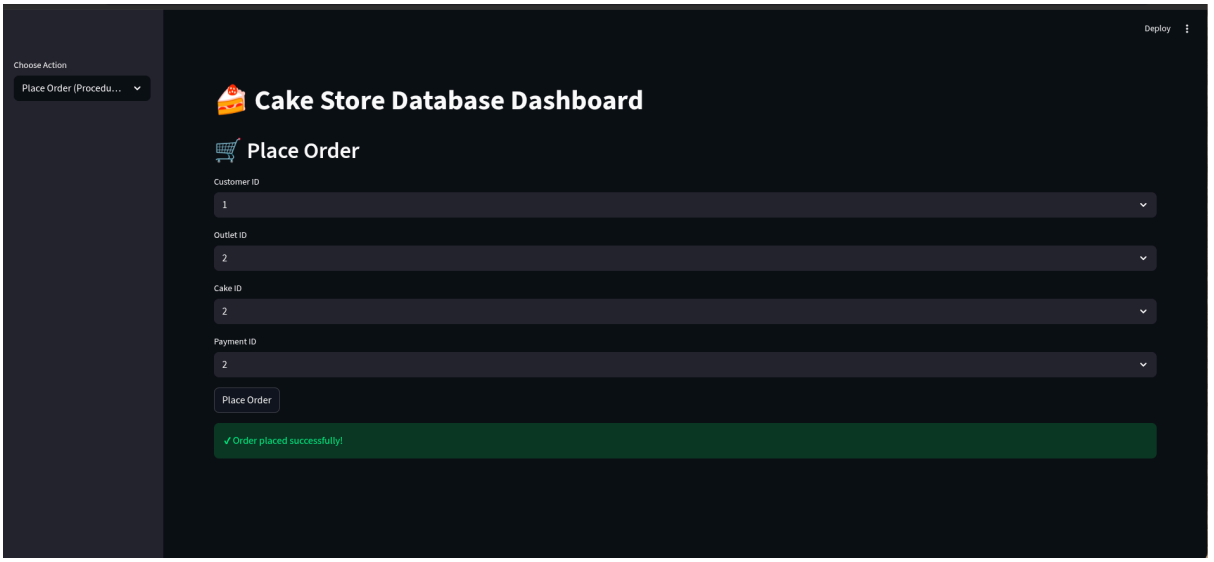
Trigger when negative stock

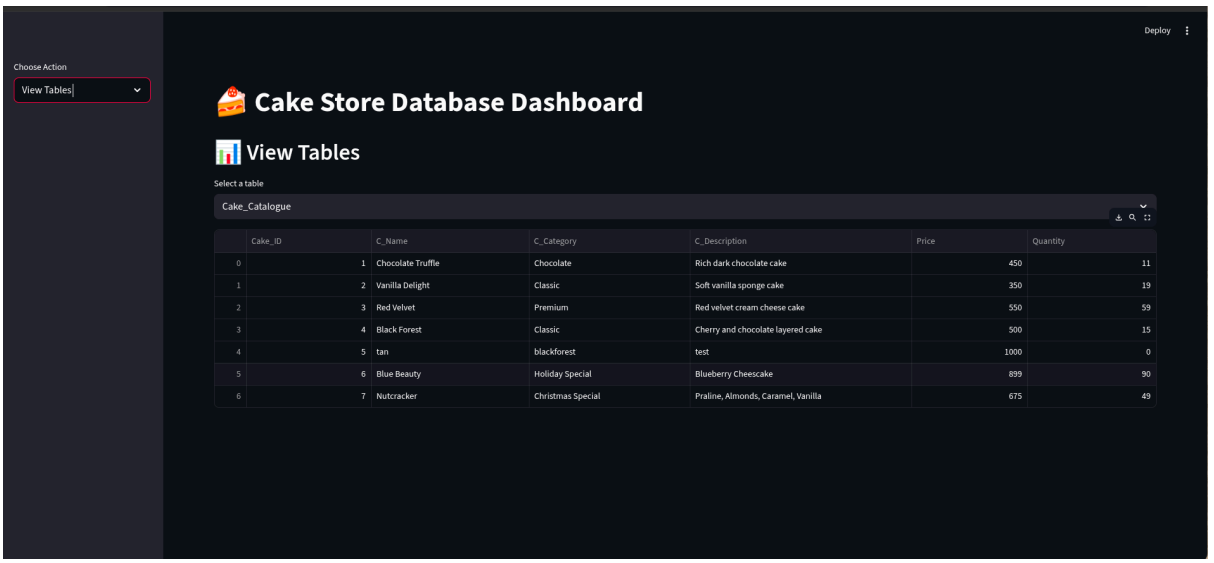
```
mysql> UPDATE Cake_Catalogue SET Quantity = -1 WHERE Cake_ID = 5;
ERROR 1644 (45000): Cake stock cannot be negative!
```

Stock reduce on order



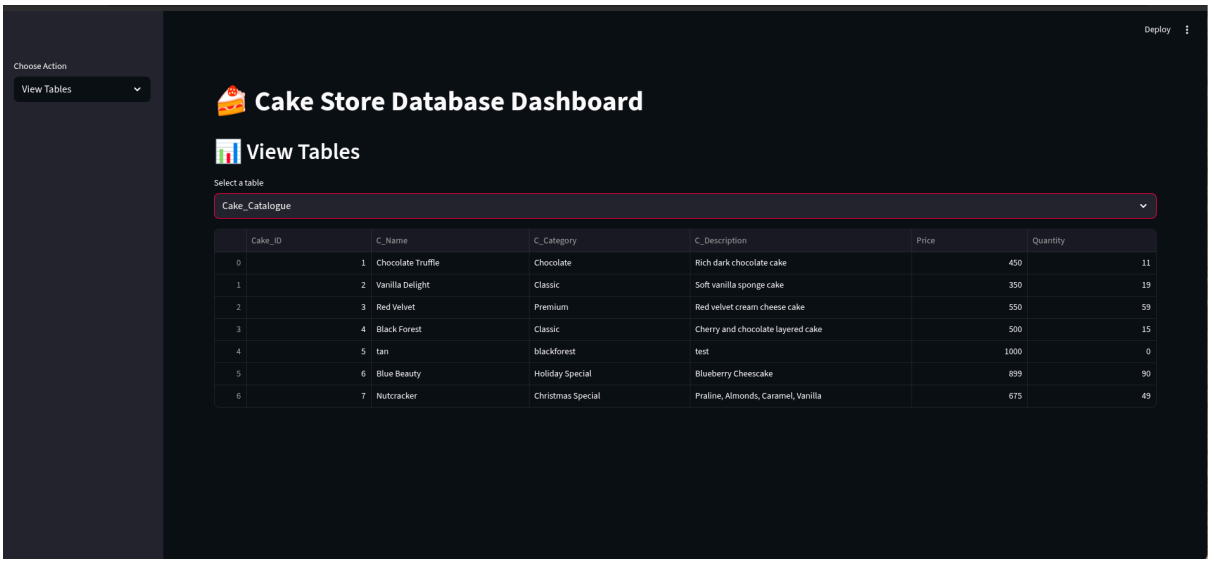
	Cake_ID	C_Name	C_Category	C_Description	Price	Quantity
0	1	Chocolate Truffle	Chocolate	Rich dark chocolate cake	450	11
1	2	Vanilla Delight	Classic	Soft vanilla sponge cake	350	20
2	3	Red Velvet	Premium	Red velvet cream cheese cake	550	59
3	4	Black Forest	Classic	Cherry and chocolate layered cake	500	15
4	5	tan	blackforest	test	1000	0
5	6	Blue Beauty	Holiday Special	Blueberry Cheescake	899	90
6	7	Nutcracker	Christmas Special	Praline, Almonds, Caramel, Vanilla	675	49



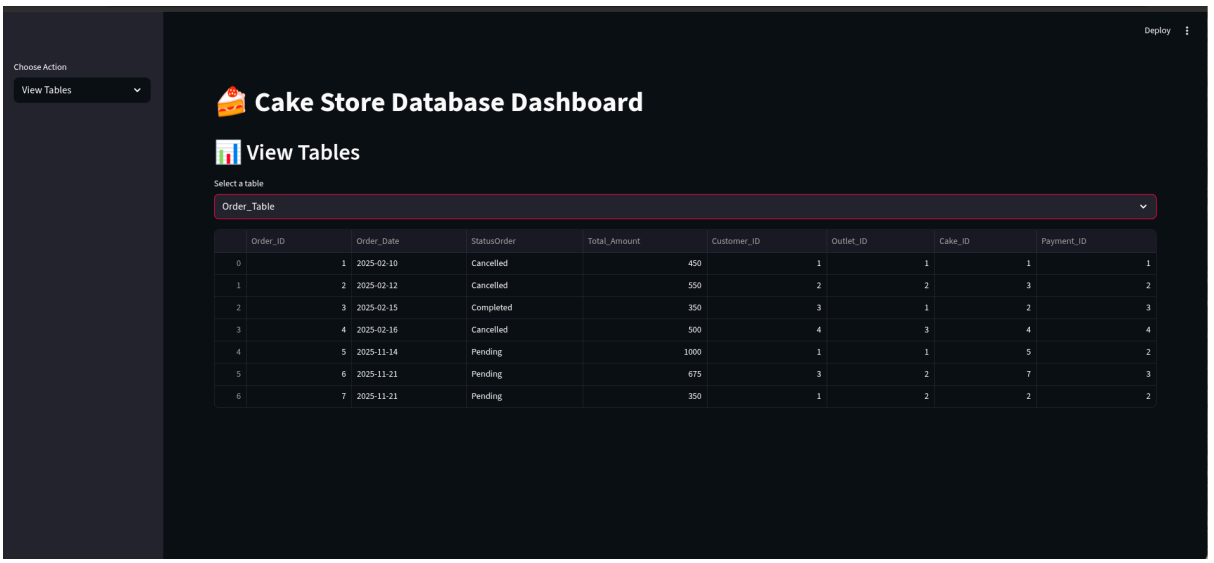


	Cake_ID	C_Name	C_Category	C_Description	Price	Quantity
0	1	Chocolate Truffle	Chocolate	Rich dark chocolate cake	450	11
1	2	Vanilla Delight	Classic	Soft vanilla sponge cake	350	19
2	3	Red Velvet	Premium	Red velvet cream cheese cake	550	59
3	4	Black Forest	Classic	Cherry and chocolate layered cake	500	15
4	5	tan	blackforest	test	1000	0
5	6	Blue Beauty	Holiday Special	Blueberry Cheescake	899	90
6	7	Nutcracker	Christmas Special	Praline, Almonds, Caramel, Vanilla	675	49

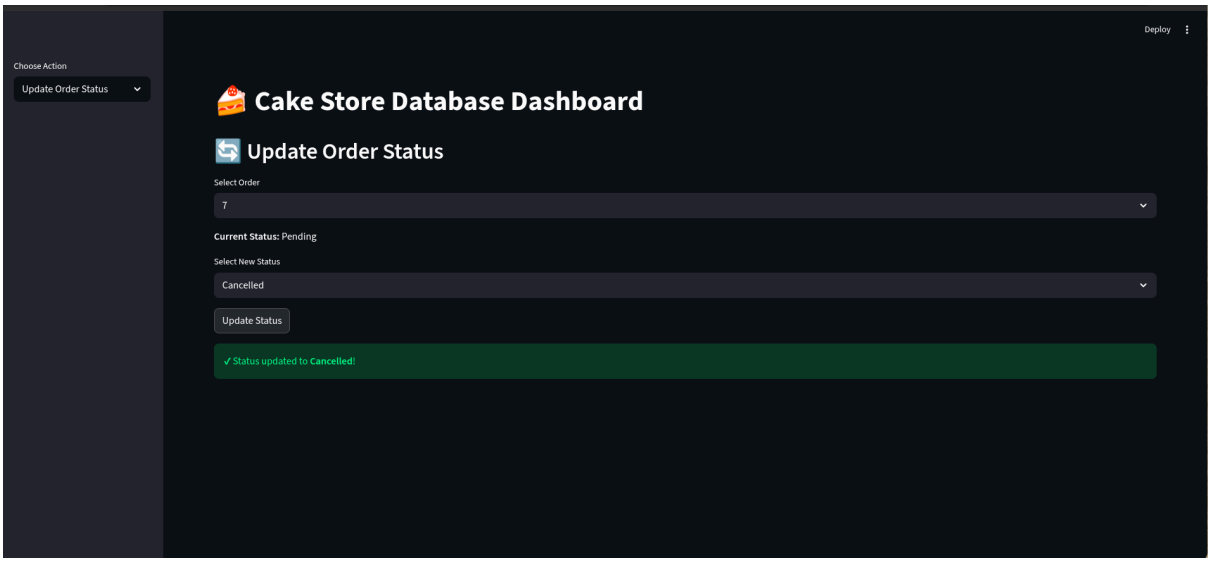
Stock increase on order cancellation

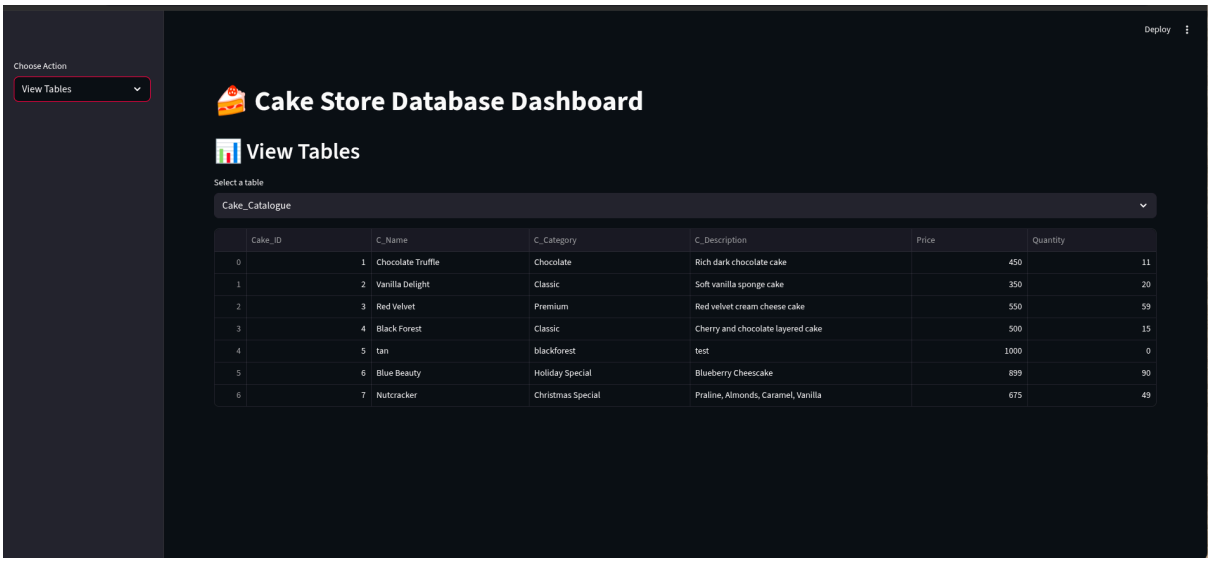


	Cake_ID	C_Name	C_Category	C_Description	Price	Quantity
0	1	Chocolate Truffle	Chocolate	Rich dark chocolate cake	450	11
1	2	Vanilla Delight	Classic	Soft vanilla sponge cake	350	19
2	3	Red Velvet	Premium	Red velvet cream cheese cake	550	59
3	4	Black Forest	Classic	Cherry and chocolate layered cake	500	15
4	5	tan	blackforest	test	1000	0
5	6	Blue Beauty	Holiday Special	Blueberry Cheescake	899	90
6	7	Nutcracker	Christmas Special	Praline, Almonds, Caramel, Vanilla	675	49



	Order_ID	Order_Date	StatusOrder	Total_Amount	Customer_ID	Outlet_ID	Cake_ID	Payment_ID
0	1	2025-02-10	Cancelled	450	1	1	1	1
1	2	2025-02-12	Cancelled	550	2	2	3	2
2	3	2025-02-15	Completed	350	3	1	2	3
3	4	2025-02-16	Cancelled	500	4	3	4	4
4	5	2025-11-14	Pending	1000	1	1	5	2
5	6	2025-11-21	Pending	675	3	2	7	3
6	7	2025-11-21	Pending	350	1	2	2	2





	Cake_ID	C_Name	C_Category	C_Description	Price	Quantity
0	1	Chocolate Truffle	Chocolate	Rich dark chocolate cake	450	11
1	2	Vanilla Delight	Classic	Soft vanilla sponge cake	350	20
2	3	Red Velvet	Premium	Red velvet cream cheese cake	550	59
3	4	Black Forest	Classic	Cherry and chocolate layered cake	500	15
4	5	tan	blackforest	test	1000	0
5	6	Blue Beauty	Holiday Special	Blueberry Cheescake	899	90
6	7	Nutcracker	Christmas Special	Praline, Almonds, Caramel, Vanilla	675	49

10. Triggers, Procedures/Functions, Nested query, Join, Aggregate queries

```
-- ========================================
-- TRIGGERS
-- ========================================

-- 1. Prevent Negative Stock
DELIMITER //

CREATE TRIGGER trg_prevent_negative_stock
BEFORE UPDATE ON Cake_Catalogue
FOR EACH ROW
BEGIN
    IF NEW.Quantity < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cake stock cannot be negative!';
    END IF;
END;
//

DELIMITER ;


-- 2. Auto Reduce Cake Stock on Order Insert
DELIMITER //

CREATE TRIGGER trg_reduce_cake_stock
AFTER INSERT ON Order_Table
FOR EACH ROW
BEGIN
    UPDATE Cake_Catalogue
    SET Quantity = Quantity - 1
    WHERE Cake_ID = NEW.Cake_ID;
END;
//

DELIMITER ;
```

```
206
207 CREATE TRIGGER trg_calculate_order_total
208 BEFORE INSERT ON Order_Table
209 FOR EACH ROW
210 BEGIN
211     IF NEW.Total_Amount IS NULL OR NEW.Total_Amount = 0 THEN
212         SET NEW.Total_Amount = (
213             SELECT Price FROM Cake_Catalogue WHERE Cake_ID = NEW.Cake_ID
214         );
215     END IF;
216 END;
217 //
218
219 DELIMITER ;
220
221
222 -- 4. Validate Payment Exists
223 DELIMITER //
224
225 CREATE TRIGGER trg_validate_payment
226 BEFORE INSERT ON Order_Table
227 FOR EACH ROW
228 BEGIN
229     IF (SELECT COUNT(*) FROM Payment WHERE Payment_ID = NEW.Payment_ID) = 0 THEN
230         SIGNAL SQLSTATE '45000'
231         SET MESSAGE_TEXT = 'Invalid Payment_ID! Payment record does not exist.';
232     END IF;
233 END;
234 //
235
236 DELIMITER ;
237
238 -- 5. Auto Increase Cake Stock on Order Cancellation
239 DELIMITER //
240
241 CREATE TRIGGER trg_increase_cake_stock_on_cancel
242 AFTER UPDATE ON Order_Table
243 FOR EACH ROW
244 BEGIN
245     -- Only trigger when status changes from NOT Cancelled → Cancelled
246     IF OLD.StatusOrder <> 'Cancelled' AND NEW.StatusOrder = 'Cancelled' THEN
```

```
    IF OLD.StatusOrder <> 'Cancelled' AND NEW.StatusOrder = 'Cancelled' THEN
        UPDATE Cake_Catalogue
        SET Quantity = Quantity + 1
        WHERE Cake_ID = NEW.Cake_ID;
    END IF;
END;
//

DELIMITER ;


-- ========================================
-- PROCEDURES
-- ========================================

DELIMITER //

-- 1. Procedure: Place an Order
CREATE PROCEDURE Place_Order (
    IN p_customer INT,
    IN p_outlet INT,
    IN p_cake INT,
    IN p_payment INT
)
BEGIN
    INSERT INTO Order_Table (
        Order_Date,
        StatusOrder,
        Customer_ID,
        Outlet_ID,
        Cake_ID,
        Payment_ID
    )
    VALUES (
        CURDATE(),
        'Pending',
        p_customer,
        p_outlet,
        p_cake,
        p_payment
    );
```

```
290
291 -- 2. Procedure: Restock Cakes
292 CREATE PROCEDURE Restock_Cake (
293     IN p_cake_id INT,
294     IN p_add_quantity INT
295 )
296 BEGIN
297     UPDATE Cake_Catalogue
298     SET Quantity = Quantity + p_add_quantity
299     WHERE Cake_ID = p_cake_id;
300 END;
301 //
302
303 DELIMITER ;
304
305 -- ========================================
306 -- FUNCTIONS
307 -- ========================================
308
309 DELIMITER //
310
311 -- 1. Function: Get Total Sales of a Cake
312 CREATE FUNCTION GetCakeSales(cake INT)
313 RETURNS DECIMAL(10,2)
314 DETERMINISTIC
315 BEGIN
316     RETURN (
317         SELECT SUM(Total_Amount)
318         FROM Order_Table
319         WHERE Cake_ID = cake AND StatusOrder = 'Completed'
320     );
321 END;
322 //
323
324
325 -- 2. Function: Check Stock Health
326 CREATE FUNCTION CheckStock(cake INT)
327 RETURNS VARCHAR(20)
328 DETERMINISTIC
329 BEGIN
330     DECLARE qty INT;
```

```
325 -- 2. Function: Check Stock Health
326 CREATE FUNCTION CheckStock(cake INT)
327 RETURNS VARCHAR(20)
328 DETERMINISTIC
329 BEGIN
330     DECLARE qty INT;
331
332     SELECT Quantity INTO qty
333     FROM Cake_Catalogue
334     WHERE Cake_ID = cake;
335
336     IF qty <= 0 THEN
337         RETURN 'OUT OF STOCK';
338     ELSEIF qty < 5 THEN
339         RETURN 'LOW STOCK';
340     ELSE
341         RETURN 'OK';
342     END IF;
343 END;
344 //
345
346 DELIMITER ;
```

11. Code snippets for invoking the Procedures/Functions/Trigger

```
# ------------------------------------------------
# UTILITIES
# ------------------------------------------------
def fetch_df(query):
    conn = get_connection()
    df = pd.read_sql(query, conn)
    conn.close()
    return df

def get_list(table, col):
    df = fetch_df(f"SELECT {col} FROM {table}")
    return df[col].tolist()


def execute_sql(query, params=None):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute(query, params if params else [])
    conn.commit()
    conn.close()

def get_value(table, select_col, where_col, where_val):
    conn = get_connection()
    cursor = conn.cursor()
    query = f"SELECT {select_col} FROM {table} WHERE {where_col} = %s"
    cursor.execute(query, (where_val,))
    result = cursor.fetchone()
    conn.close()
    return result[0] if result else None
```

```
# ------------------------------------------------
# VIEW TABLES
# ------------------------------------------------
if menu == "View Tables":
    st.header("📊 View Tables")

    tables = fetch_df("SHOW TABLES")
    table = st.selectbox("Select a table", tables.iloc[:, 0])

    if table:
        df = fetch_df(f"SELECT * FROM {table}")
        st.dataframe(df, use_container_width=True)
```

```python
# INSERT DATA (generic)
# ------------------------------------------------
elif menu == "Insert Data":
    st.header("➕ Insert Data")

    tables = fetch_df("SHOW TABLES")
    table = st.selectbox("Select Table", tables.iloc[:, 0])

    if table:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute(f"DESCRIBE {table}")
        cols = cursor.fetchall()
        conn.close()

        auto_inc = ["Order_ID", "Cake_ID"]

        form = {}
        st.subheader(f"Inserting into {table}")

        for col in cols:
            name, dtype = col[0], col[1]

            if name in auto_inc:
                continue

            if "int" in dtype:
                form[name] = st.number_input(name, step=1)
            elif "date" in dtype:
                form[name] = st.date_input(name)
            else:
                form[name] = st.text_input(name)

        if st.button("Insert Record"):
            col_names = ", ".join(form.keys())
            placeholders = ", ".join(["%s"] * len(form))
            values = list(form.values())
            query = f"INSERT INTO {table} ({col_names}) VALUES ({placeholders})"

            try:
                execute_sql(query, values)
```

```
            col_names = ", ".join(form.keys())
            placeholders = ", ".join(["%s"] * len(form))
            values = list(form.values())
            query = f"INSERT INTO {table} ({col_names}) VALUES ({placeholders})"

            try:
                execute_sql(query, values)
                st.success("✔ Record inserted!")
            except Exception as e:
                st.error(f"❌ {e}")


# ------------------------------------------------
# PLACE ORDER — CALL PROCEDURE
# ------------------------------------------------
elif menu == "Place Order (Procedure)":
    st.header("🛒 Place Order")

    customer = st.selectbox("Customer ID", get_list("Customers", "Customer_ID"))
    outlet = st.selectbox("Outlet ID", get_list("Outlet", "Outlet_ID"))
    cake = st.selectbox("Cake ID", get_list("Cake_Catalogue", "Cake_ID"))
    payment = st.selectbox("Payment ID", get_list("Payment", "Payment_ID"))

    if st.button("Place Order"):
        try:
            execute_sql("CALL Place_Order(%s,%s,%s,%s)", (customer, outlet, cake, payment
))
            st.success("✔ Order placed successfully!")
        except Exception as e:
           st.error(f"❌ {e}")
```

```python
# RESTOCK CAKE – CALL PROCEDURE
# ------------------------------------------------
elif menu == "Restock Cake (Procedure)":
    st.header("📦 Restock Cake")

    cake_id = st.selectbox("Select Cake", get_list("Cake_Catalogue", "Cake_ID"))
    qty = st.number_input("Add Quantity", step=1)

    if st.button("Restock"):
        execute_sql("CALL Restock_Cake(%s, %s)", (cake_id, qty))
        st.success("✔ Stock updated!")


# ------------------------------------------------
# RUN FUNCTIONS
# ------------------------------------------------
elif menu == "Run Functions":
    st.header("🧠 Run MySQL Functions")

    st.subheader("GetCakeSales(cake_id)")
    cake = st.selectbox("Select Cake ID", get_list("Cake_Catalogue", "Cake_ID"))
    if st.button("Get Sales"):
        df = fetch_df(f"SELECT GetCakeSales({cake}) AS TotalSales")
        st.info(f"Total Sales for Cake {cake}: ₹{df['TotalSales'][0]}")

    st.subheader("CheckStock(cake_id)")
    cake2 = st.selectbox("Check Stock for Cake", get_list("Cake_Catalogue", "Cake_ID"))
    if st.button("Check Stock"):
        df = fetch_df(f"SELECT CheckStock({cake2}) AS StockStatus")
        st.warning(f"Stock Status: {df['StockStatus'][0]}")
```

```python
# ----------------------------------------------------
# UPDATE ORDER STATUS
# ----------------------------------------------------
elif menu == "Update Order Status":
    st.header("🔄 Update Order Status")

    # Get existing order IDs
    orders = get_list("Order_Table", "Order_ID")
    order_id = st.selectbox("Select Order", orders)

    # Get current status to show in UI (optional)
    current_status = get_value("Order_Table", "StatusOrder", "Order_ID", order_id)
    st.write(f"**Current Status:** {current_status}")

    # Choose new status
    new_status = st.selectbox(
        "Select New Status",
        ["Pending", "Completed", "Cancelled"],
        index=["Pending", "Completed", "Cancelled"].index(current_status)
        if current_status in ["Pending", "Completed", "Cancelled"]
        else 0
    )

    # Update DB
    if st.button("Update Status"):
        execute_sql(
            "UPDATE Order_Table SET StatusOrder=%s WHERE Order_ID=%s",
            (new_status, order_id)
        )
        st.success(f"✔ Status updated to **{new_status}**!")

# ----------------------------------------------------
# ANALYTICS DASHBOARD
# ----------------------------------------------------
elif menu == "Analytics Dashboard":
    st.header("📈 Dashboard")

    col1, col2, col3 = st.columns(3)

    col1.metric("Total Orders", fetch_df("SELECT COUNT(*) AS c FROM Order_Table")["c"][0]
)
```

```python
# ---------------------------------------------------
# ANALYTICS DASHBOARD
# ---------------------------------------------------
elif menu == "Analytics Dashboard":
    st.header("📈 Dashboard")

    col1, col2, col3 = st.columns(3)

    col1.metric("Total Orders", fetch_df("SELECT COUNT(*) AS c FROM Order_Table")["c"][0]
)
    col2.metric("Completed", fetch_df("SELECT COUNT(*) AS c FROM Order_Table WHERE Status
Order='Completed'")["c"][0])
    col3.metric("Cancelled", fetch_df("SELECT COUNT(*) AS c FROM Order_Table WHERE Status
Order='Cancelled'")["c"][0])

    st.subheader("Best Selling Cakes")
    df_best = fetch_df("""
        SELECT C.C_Name, COUNT(*) AS Sold
        FROM Order_Table O
        JOIN Cake_Catalogue C ON O.Cake_ID=C.Cake_ID
        WHERE StatusOrder='Completed'
        GROUP BY O.Cake_ID
        ORDER BY Sold DESC
    """)
    st.bar_chart(df_best.set_index("C_Name"))


# ---------------------------------------------------
# RUN SELECT QUERIES
# ---------------------------------------------------
elif menu == "Run Custom Query":
    st.header("🔍 Custom Query")

    sql = st.text_area("Enter SELECT query only")

    if st.button("Run"):
        if not sql.strip().lower().startswith("select"):
            st.error("Only SELECT queries are allowed.")
        else:
            st.dataframe(fetch_df(sql), use_container_width=True)
```

12. SQL queries(Create, Insert, Triggers, Procedures/Functions, Nested query, Join, Aggregate queries) used in the project in the form of .sql file

.sql file code

```sql
CREATE DATABASE IF NOT EXISTS Cake_Store;
USE Cake_Store;

-- ==========================
-- Tables
-- ==========================

CREATE TABLE Payment (
    Payment_ID INT PRIMARY KEY,
    Amount DECIMAL(10,2),
    Method VARCHAR(50),
    Payment_Date DATE
);

CREATE TABLE Outlet (
    Outlet_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Address VARCHAR(200),
    Phone_Number VARCHAR(15)
);

CREATE TABLE Employees (
    Employee_ID INT PRIMARY KEY,
    E_Name VARCHAR(100),
    RoleOfPerson VARCHAR(50),
    Phone_Number VARCHAR(15),
    Outlet_ID INT,
    Production_ID INT,
    FOREIGN KEY (Outlet_ID) REFERENCES Outlet(Outlet_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Customers (
    Customer_ID INT PRIMARY KEY,
    NameOfPerson VARCHAR(100),
    Email VARCHAR(100),
    PhoneNumber VARCHAR(15)
);

CREATE TABLE Cake_Catalogue (
```

```sql

CREATE TABLE Cake_Catalogue (
    Cake_ID INT PRIMARY KEY AUTO_INCREMENT,
    C_Name VARCHAR(100),
    C_Category VARCHAR(50),
    C_Description TEXT,
    Price DECIMAL(8,2),
    Quantity INT
);

CREATE TABLE Order_Table (
    Order_ID INT PRIMARY KEY AUTO_INCREMENT,
    Order_Date DATE,
    StatusOrder VARCHAR(20),
    Total_Amount DECIMAL(10,2),
    Customer_ID INT,
    Outlet_ID INT,
    Cake_ID INT,
    Payment_ID INT,
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Outlet_ID) REFERENCES Outlet(Outlet_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
     FOREIGN KEY (Cake_ID) REFERENCES Cake_Catalogue(Cake_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
     FOREIGN KEY (Payment_ID) REFERENCES Payment(Payment_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Customers_Address (
    Customer_ID INT,
    Address VARCHAR(200),
    PRIMARY KEY (Customer_ID, Address),
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

```sql

CREATE TABLE Customers_Phone_Numbers (
    Customer_ID INT,
    Phone_Number VARCHAR(15),
    PRIMARY KEY (Customer_ID, Phone_Number),
    FOREIGN KEY (Customer_ID) REFERENCES Customers(Customer_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Outlet_Phone_Numbers (
    Outlet_ID INT,
    Phone_Number VARCHAR(15),
    PRIMARY KEY (Outlet_ID, Phone_Number),
    FOREIGN KEY (Outlet_ID) REFERENCES Outlet(Outlet_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);


-- =========================
-- Inserts
-- =========================

-- Payment
INSERT INTO Payment (Payment_ID, Amount, Method, Payment_Date) VALUES
(1, 450.00, 'Credit Card', '2025-01-12'),
(2, 720.50, 'UPI', '2025-01-15'),
(3, 299.99, 'Cash', '2025-02-01'),
(4, 1500.00, 'Debit Card', '2025-02-05');

-- Outlet
INSERT INTO Outlet (Outlet_ID, Name, Address, Phone_Number) VALUES
(1, 'Sweet Haven', 'MG Road, Bengaluru', '9876543210'),
(2, 'Cake World', 'Indiranagar, Bengaluru', '9988776655'),
(3, 'Bakers Hub', 'Koramangala, Bengaluru', '9123456780');

-- Employees
INSERT INTO Employees (Employee_ID, E_Name, RoleOfPerson, Phone_Number, Outlet_ID, Production_ID) VALUES
(1, 'Ramesh Kumar', 'Manager', '9123012301', 1, NULL),
```

```sql
INSERT INTO Employees (Employee_ID, E_Name, RoleOfPerson, Phone_Number, Outlet_ID, Production_ID) VALUES
(1, 'Ramesh Kumar', 'Manager', '9123012301', 1, NULL),
(2, 'Anita Sharma', 'Chef', '9898989898', 1, NULL),
(3, 'John Mathew', 'Cashier', '9000090000', 2, NULL),
(4, 'Sara Ali', 'Baker', '8080808080', 3, NULL);

-- Customers
INSERT INTO Customers (Customer_ID, NameOfPerson, Email, PhoneNumber) VALUES
(1, 'Aarav Gupta', 'aarav@example.com', '8881112222'),
(2, 'Riya Sharma', 'riya@example.com', '9991112222'),
(3, 'Karan Verma', 'karan@example.com', '7771112222'),
(4, 'Meera Rao', 'meera@example.com', '9662223333');

-- Cake Catalogue
INSERT INTO Cake_Catalogue (C_Name, C_Category, C_Description, Price, Quantity) VALUES
('Chocolate Truffle', 'Chocolate', 'Rich dark chocolate cake', 450.00, 10),
('Vanilla Delight', 'Classic', 'Soft vanilla sponge cake', 350.00, 20),
('Red Velvet', 'Premium', 'Red velvet cream cheese cake', 550.00, 8),
('Black Forest', 'Classic', 'Cherry and chocolate layered cake', 500.00, 15);

-- Orders
INSERT INTO Order_Table (Order_Date, StatusOrder, Total_Amount, Customer_ID, Outlet_ID, Cake_ID, Payment_ID) VALUES
('2025-02-10', 'Completed', 450.00, 1, 1, 1, 1),
('2025-02-12', 'Pending', 550.00, 2, 2, 3, 2),
('2025-02-15', 'Completed', 350.00, 3, 1, 2, 3),
('2025-02-16', 'Cancelled', 500.00, 4, 3, 4, 4);

-- Customer Addresses
INSERT INTO Customers_Address (Customer_ID, Address) VALUES
(1, 'HSR Layout, Bengaluru'),
(2, 'Whitefield, Bengaluru'),
(3, 'JP Nagar, Bengaluru'),
(4, 'BTM Layout, Bengaluru');

-- Customer Phone Numbers
INSERT INTO Customers_Phone_Numbers (Customer_ID, Phone_Number) VALUES
(1, '8881112222'),
(1, '8881113333'),
(2, '9991112222'),
```

```
(1, '8881113333'),
(2, '9991112222'),
(3, '7771112222');

-- Outlet Phone Numbers
INSERT INTO Outlet_Phone_Numbers (Outlet_ID, Phone_Number) VALUES
(1, '9876543210'),
(1, '9765432109'),
(2, '9988776655'),
(3, '9123456780');

-- ========================================
-- TRIGGERS
-- ========================================

-- 1. Prevent Negative Stock
DELIMITER //

CREATE TRIGGER trg_prevent_negative_stock
BEFORE UPDATE ON Cake_Catalogue
FOR EACH ROW
BEGIN
    IF NEW.Quantity < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cake stock cannot be negative!';
    END IF;
END;
//

DELIMITER ;

-- 2. Auto Reduce Cake Stock on Order Insert
DELIMITER //

CREATE TRIGGER trg_reduce_cake_stock
AFTER INSERT ON Order_Table
FOR EACH ROW
BEGIN
    UPDATE Cake_Catalogue
    SET Quantity = Quantity - 1
```

```
    UPDATE Cake_Catalogue
    SET Quantity = Quantity - 1
    WHERE Cake_ID = NEW.Cake_ID;
END;
//

DELIMITER ;


-- 3. Auto Calculate Order Total
DELIMITER //

CREATE TRIGGER trg_calculate_order_total
BEFORE INSERT ON Order_Table
FOR EACH ROW
BEGIN
    IF NEW.Total_Amount IS NULL OR NEW.Total_Amount = 0 THEN
        SET NEW.Total_Amount = (
            SELECT Price FROM Cake_Catalogue WHERE Cake_ID = NEW.Cake_ID
        );
    END IF;
END;
//

DELIMITER ;


-- 4. Validate Payment Exists
DELIMITER //

CREATE TRIGGER trg_validate_payment
BEFORE INSERT ON Order_Table
FOR EACH ROW
BEGIN
    IF (SELECT COUNT(*) FROM Payment WHERE Payment_ID = NEW.Payment_ID) = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid Payment_ID! Payment record does not exist.';
    END IF;
END;
//

```

```
DELIMITER ;

-- 5. Auto Increase Cake Stock on Order Cancellation
DELIMITER //

CREATE TRIGGER trg_increase_cake_stock_on_cancel
AFTER UPDATE ON Order_Table
FOR EACH ROW
BEGIN
    -- Only trigger when status changes from NOT Cancelled → Cancelled
    IF OLD.StatusOrder <> 'Cancelled' AND NEW.StatusOrder = 'Cancelled' THEN
        UPDATE Cake_Catalogue
        SET Quantity = Quantity + 1
        WHERE Cake_ID = NEW.Cake_ID;
    END IF;
END;
//

DELIMITER ;


-- ========================================
-- PROCEDURES
-- ========================================

DELIMITER //

-- 1. Procedure: Place an Order
CREATE PROCEDURE Place_Order (
    IN p_customer INT,
    IN p_outlet INT,
    IN p_cake INT,
    IN p_payment INT
)
BEGIN
    INSERT INTO Order_Table (
        Order_Date,
        StatusOrder,
        Customer_ID,
        Outlet_ID,
        Cake_ID,
```

```
        Payment_ID
    )
    VALUES (
        CURDATE(),
        'Pending',
        p_customer,
        p_outlet,
        p_cake,
        p_payment
    );
END;
//


-- 2. Procedure: Restock Cakes
CREATE PROCEDURE Restock_Cake (
    IN p_cake_id INT,
    IN p_add_quantity INT
)
BEGIN
    UPDATE Cake_Catalogue
    SET Quantity = Quantity + p_add_quantity
    WHERE Cake_ID = p_cake_id;
END;
//

DELIMITER ;

-- ========================================
-- FUNCTIONS
-- ========================================

DELIMITER //

-- 1. Function: Get Total Sales of a Cake
CREATE FUNCTION GetCakeSales(cake INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    RETURN (
        SELECT SUM(Total_Amount)
```

```sql
-- FUNCTIONS
-- ========================================

DELIMITER //

-- 1. Function: Get Total Sales of a Cake
CREATE FUNCTION GetCakeSales(cake INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    RETURN (
        SELECT SUM(Total_Amount)
        FROM Order_Table
        WHERE Cake_ID = cake AND StatusOrder = 'Completed'
    );
END;
//


-- 2. Function: Check Stock Health
CREATE FUNCTION CheckStock(cake INT)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE qty INT;

    SELECT Quantity INTO qty
    FROM Cake_Catalogue
    WHERE Cake_ID = cake;

    IF qty <= 0 THEN
        RETURN 'OUT OF STOCK';
    ELSEIF qty < 5 THEN
        RETURN 'LOW STOCK';
    ELSE
        RETURN 'OK';
    END IF;
END;
//

DELIMITER ;
```

13. Github repo link

    https://github.com/tanishhegde/CakeStore-Database-Management-System