Database Systems CSC 3326 – 01

FINAL PROJECT REPORT

Rita Jalal

Zaynab Aboulkacem

Tanishka Suwalka

Supervised by: Dr. Lamiae Bouanane

FALL 2024

Table of Contents

1. Introduction:

Our project will entail a collaboration with the Ifrane Marche library, which will be the main source of data for our website. This setup helps the library to effectively manage inventory and orders using an advanced digital platform. This website will mainly cater to AUI's students, professors and staff, offering diverse university related necessities.

2. Project Description:

a. Mission:

Our main mission is to create an easy-to-use ordering platform for school supplies, such as books, notebooks, pens and pencils and other necessary tools for both school and university. We plan to digitize the city's library to make it easier for both the customers (AUI and Ifrane community) and the library staff to ensure a seamless experience from browsing products to delivery or pickup.

b. Operations:

The platform will enable users to:

- **Browse** for school/university-related supplies.
- **Place orders** online for books,notebooks, and other stationary.
- **Process and fulfill** orders efficiently.

c. Challenges:

Time Limitations:

- students sometimes find it difficult to find time for excursions to the market for necessities.

Manual Procedures:

- The Ifrane Marche library currently uses a manual method of operations, which can be harder to implement and take longer to fulfill orders.

d. Constraints:

Technical Knowledge:

- To properly adjust to the new digital procedures, library employees might need some training for the new system.

Financial Restrictions:

- This platform's implementation might require major technological investments, which could mean being limited by the amount of money available.

Platform Maintenance:

- To keep the platform operating effectively and securely, regular upgrades and technical assistance will be needed.

Data security:

- protecting user and transaction data is necessary for this implementation

3.Project Objectives:

The primary objective of our platform is to create a seamless online platform where users can:

1. Browse a catalog of books and school-related items.
2. Place orders efficiently and securely.
3. Track order statuses in real time.
4. Manage user accounts, including order history and saved preferences.

In order to obtain this we need to:

- **Provide a Web Platform:** we need a user-friendly online system where users can browse and order their supplies.

- **Restrict Access**: we need to ensure that only verified university members (students, faculty, and staff) can access the platform.

- **Product Catalog**: we need to allow users to view available items, including books, pens, notebooks, and other essentials, with search and filter options.

- **Order Management**: we need to simplify the ordering process, including adding items to a cart, checking out, and tracking orders.

- **Integration with the Library**: we have to enable the library to list its resources and manage its stock more efficiently through the platform.

In short our objectives are to:

- Save time for students and staff by eliminating unnecessary trips to marche.
- Provide easy access to a wide range of university supplies.
- Improve the library's efficiency and accessibility.
- Enhance user convenience through a digital, all-in-one solution.

4. Project planning:

1. Procedures and tasks distribution

In order to meet all the requirements and do a good job. The team gathers to read the guidelines of each deliverable thoroughly. We make sure to share ideas and brainstorm together. Then comes the division of tasks, we try to make it fair for everyone and we all help each other if any of us is stuck in a task. We then review each other's work and give comments and feedback which makes the work better. Needless to mention that communication with the client is crucial because that's how we know what the client wants and what the business rules are to make a good design and ultimately a good database.

Task	Supervisor
Database Design	Tanishka, Rita, Zaynab
Database Creation	Tanishka, Rita, Zaynab
Software Design	Tanishka, Rita, Zaynab
Software Implementation	Tanishka, Rita, Zaynab
Testing	Tanishka, Rita, Zaynab

5.Design:

A. Conceptual Design

a. Requirements' Specification: Main System Processes:

The primary objective of our platform is to create a seamless online platform where users can:

- Browse a catalog of books and school-related items.
- Place orders efficiently and securely.
- Track order statuses in real time.
- Manage user accounts, including order history and saved preferences.

b. Business Rules, Entities, and Relationships:

- **User Access and Registration:**
  - ➔ Any student can create an account on the platform by providing a valid email address, username, and password.
  - ➔ Users must verify their email address through a confirmation link to activate their accounts.
- **Product Catalog Management**
  - ➔ Every item in the catalog must have a unique product ID, name, category, price, and stock quantity.
  - ➔ Only administrators or authorized library staff can add, update, or remove items from the catalog.

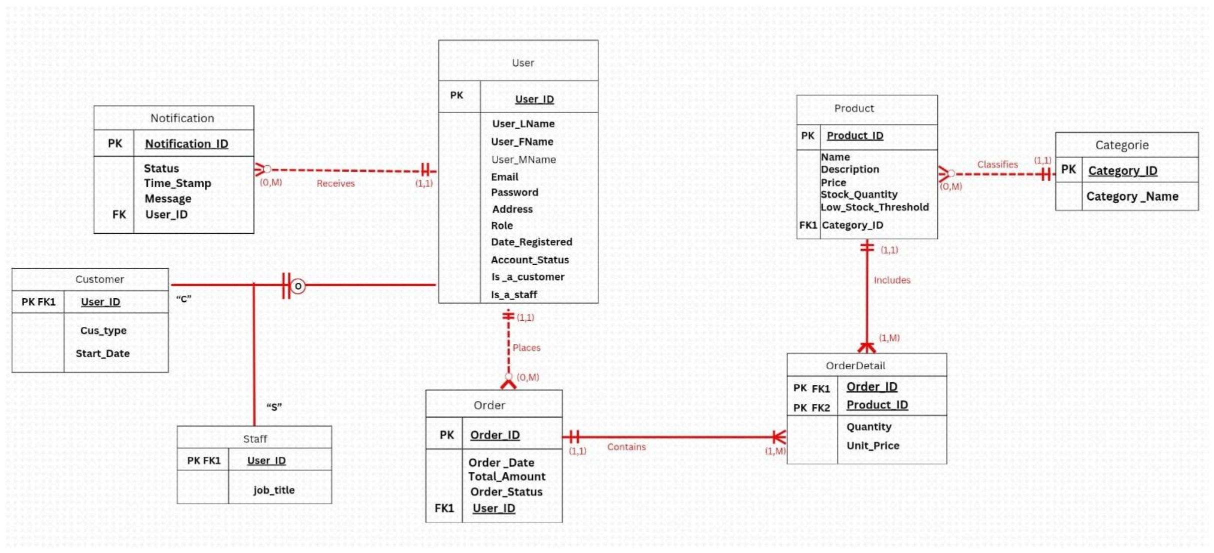
- **Order Placement and Management**
  - ➔ Users can only place an order if all items in the cart are in stock.
  - ➔ Each order must have a unique order ID and include order details such as user ID, items ordered, quantities, total price, and order status.
  - ➔ The system must automatically reduce the stock quantity of ordered items upon successful order placement.

- **Order Status and Tracking**
  - ➔ Orders can have one of the following statuses: "Pending," "Processed," "Shipped," or "Delivered."
  - ➔ Users must be able to view the current status of their orders in real-time.

- **Integration with the Library**
  - ➔ Library resources available for order must be clearly differentiated from other catalog items.
  - ➔ The system must sync with the library's inventory to ensure real-time stock updates.

- **User Account Management**
  - ➔ Each user must have a profile with their name, email, password, and order history.

c. Initial ERD:



d. Normalization :

1.

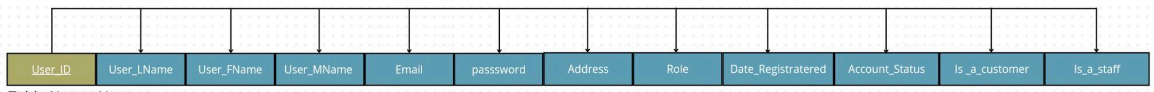


Table Name: User
User (**User_ID** , User_FName, User_LName, Email, Password, Address, Role, Date_Registered, Account_Status, Is_a_customer, Is_a_staff)

2.

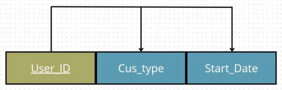


Table Name: customer
Notification (**User_ID**, Cus_type, Start_Date)

3.

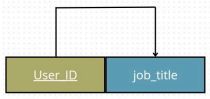


Table Name: Staff
Notification (**User_ID**, job_title)

4.

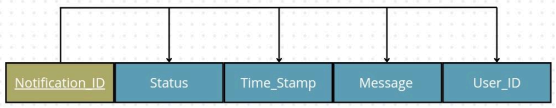


Table Name: Notification
Notification (**Notification_ID**, Status, Time_Stamp, Message , User_ID)

5.

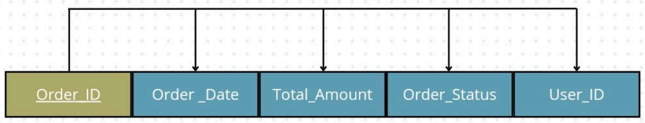


Table Name: Order
Notification (**<u>Order_ID</u>**, Order _Date, Total_Amount, Order_Status, User_ID)

6.

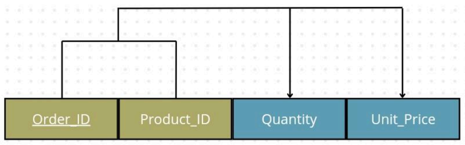


Table Name: OrderDetail
Notification (**<u>Order_ID</u>**, **<u>Product_ID</u>**, Quantity, Unit_Price)

7.

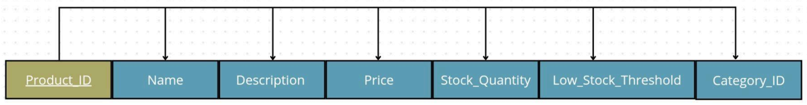


Table Name: Product
Notification (**<u>Product_ID</u>**, Name, Description, Price, Stock_Quantity, Low_Stock_Threshold, Category_ID)

8.

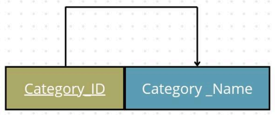
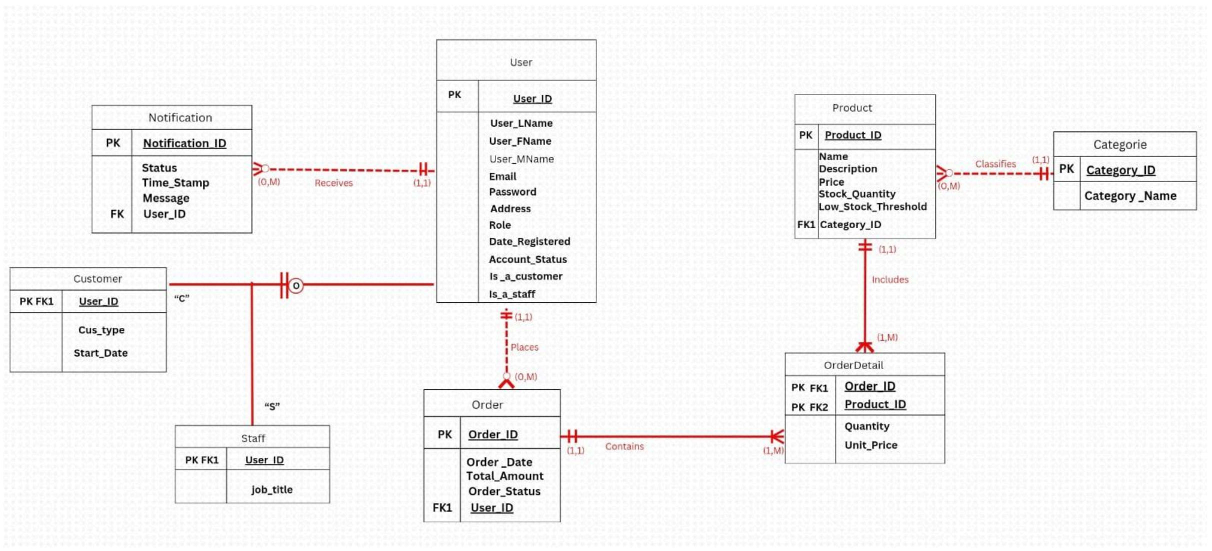


Table Name: Category
Notification (**<u>Category_ID</u>**, Category _Name)

e. Final E-R Model :



B. Logical Design

a. Tables:

1. Table: Users

Description: This table stores information about all users of the system, including both customers and staff members. Each user is uniquely identified by a User_ID.

2. Table: Customers

Description: This table is a subtype of the table S_User. It inherits all attributes of S_User and it contains specific information about customers, including their type (student or staff) and registration date.

3. Table: Staff

Description: This table is also a subtype of the table S_User. It inherits all attributes of S_User and it holds details about staff members and their job titles.

4. Table: Categories

Description: This table manages product categories to organize available products in the shop.

5. Table: Products

Description: This table manages product information available in the shop, including their stock levels and pricing.

6. Table: Orders

Description: This table records all orders placed by customers, including date, total amounts, and order statuses.

7. Table: OrderDetail

Description: This table captures the details of each order, specifying which products are included, their quantities, and unit prices.

8. Table: Notification

Description: This table stores notifications sent to users, such as order updates and account alerts. It includes the message content, status, and timestamp.

b. Attributes Data Types and Constraints:

1. Table: Users

Attributes:

User_ID: INT (Primary Key) - Unique identifier for each user.

User_LName: VARCHAR(100) (NOT NULL) - User's last name.

User_FName: VARCHAR(100) (NOT NULL) - User's first name.

User_MName: VARCHAR(100) - User's middle name (optional).

Email: VARCHAR(255) (UNIQUE, NOT NULL) - User's email address (must be unique).

Password: VARCHAR(255) (NOT NULL) - User's password for authentication.

9City: VARCHAR(100) (NOT NULL) - City where the user resides.

Country: VARCHAR(100) (NOT NULL) - Country where the user resides.

Role: VARCHAR(50) (CHECK: Role IN ('Customer', 'Staff')) - Indicates the role of the user within the system.

Date_Registered: DATE (NOT NULL) - The date when the user registered.

Account_Status: VARCHAR(50) (CHECK: Account_Status IN ('Active', 'Inactive')) - Current status of the user's account.

Is_a_customer: BOOLEAN (NOT NULL, DEFAULT FALSE) - Indicates whether the user is a customer.

Is_a_staff: BOOLEAN (NOT NULL, DEFAULT FALSE) - Indicates whether the user is a staff member.

2. Table: Customers

Attributes:

User_ID: INT (Primary Key, Foreign Key references Users(User_ID)) - Links to the corresponding user in the Users table.

Cus_type: VARCHAR(50) (CHECK: Cus_type IN ('Student', 'Staff')) - Type of customer (e.g., student or staff).

Start_Date: DATE (NOT NULL) - The date the customer relationship started.

3. Table: Staff

Attributes:

User_ID: INT (Primary Key, Foreign Key references Users(User_ID)) - Links to the corresponding user in the Users table.

Job_title: VARCHAR(100) (NOT NULL) - The job title of the staff member.

4. Table: Categories

Attributes:

Category_ID: CHAR(5) (Primary Key) - Unique identifier for each product category.

Category_Name: VARCHAR(100) (NOT NULL) - Name of the category.

5. Table: Products

Attributes:

Product_ID: CHAR(5) (Primary Key) - Unique identifier for each product.

Name: VARCHAR(100) (NOT NULL) - Name of the product.

Description: TEXT - Detailed description of the product.

Price: DECIMAL(10, 2) (NOT NULL) - Price of the product.

Stock_Quantity: INT (NOT NULL, CHECK: Stock_Quantity >= 0) - Number of items currently in stock.

Low_Stock_Threshold: INT (NOT NULL, CHECK: Low_Stock_Threshold>= 0) - Minimum stock level before needing to reorder.

Category_ID: CHAR(5) (NOT NULL, Foreign Key references Categories(Category_ID)) - Links to the category of the product.

6. Table: Orders

Attributes:

Order_ID: CHAR(5) (Primary Key) - Unique identifier for each order.

Order_Date: DATE (NOT NULL) - The date on which the order was placed.

Total_Amount: DECIMAL(10, 2) (NOT NULL) - Total cost of the order.

Order_Status: VARCHAR(50) (CHECK: Order_Status IN ('Pending', 'Processed', 'Shipped', 'Delivered')) - Current status of the order.

User_ID: INT (NOT NULL, Foreign Key references Users(User_ID)) - Links to the user who placed the order.

7. Table: OrderDetail

Attributes:

Order_ID: CHAR(5) (Foreign Key references Orders(Order_ID)) - Links to the relevant order in the Orders table.

Product_ID: CHAR(5) (Foreign Key references Products(Product_ID)) - Links to the relevant product being ordered.

Quantity: INT (NOT NULL, CHECK: Quantity > 0) - The number of units for each product in the order.

Unit_Price: DECIMAL(10, 2) (NOT NULL) - The price per unit at the time of the order.

PRIMARY KEY (Order_ID, Product_ID) - Composite primary key ensuring that each product appears once per order.

8. Table: Notification

Attributes:

Notification_ID: INT (Primary Key) - Unique identifier for each notification.

Status: VARCHAR(50) (CHECK: Status IN ('Unread', 'Read')) - Indicates whether the notification has been read or not.

Time_Stamp: TIMESTAMP (DEFAULT CURRENT_TIMESTAMP) - The time the notification was created.

Message: TEXT (NOT NULL) - The content of the notification message.

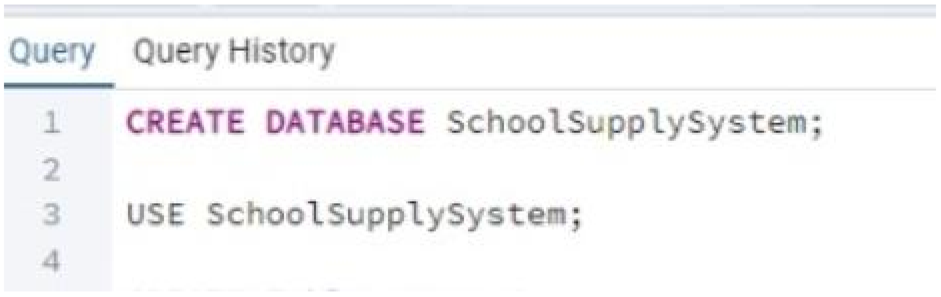
User_ID: INT (NOT NULL, Foreign Key references Users(User_ID)) - Links to the user who receives the notification.

6.Implementation:

A. Tables Creation and Population:

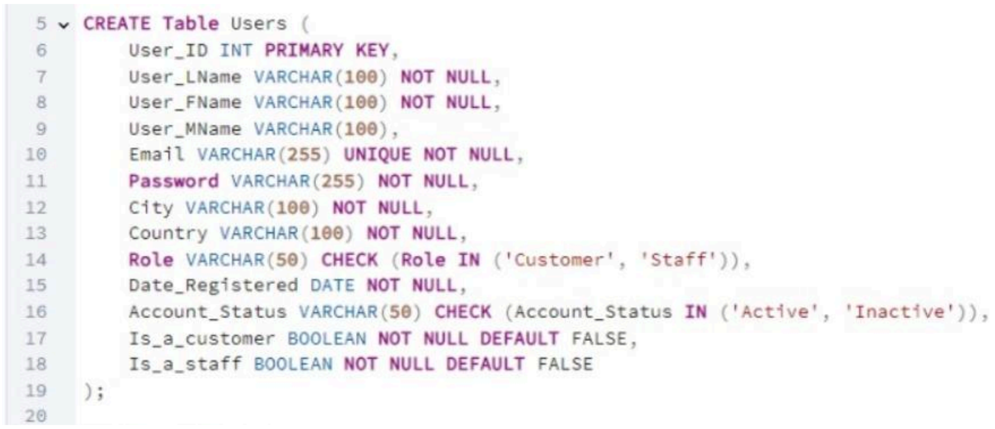
We created all the tables based on the business rules and all the constraints mentioned in the previous part. As for the population, we used random but relevant data.

- Database School supply system



```
CREATE DATABASE SchoolSupplySystem;

USE SchoolSupplySystem;

```

- Table Users



```
CREATE Table Users (
    User_ID INT PRIMARY KEY,
    User_LName VARCHAR(100) NOT NULL,
    User_FName VARCHAR(100) NOT NULL,
    User_MName VARCHAR(100),
    Email VARCHAR(255) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,
    City VARCHAR(100) NOT NULL,
    Country VARCHAR(100) NOT NULL,
    Role VARCHAR(50) CHECK (Role IN ('Customer', 'Staff')),
    Date_Registered DATE NOT NULL,
    Account_Status VARCHAR(50) CHECK (Account_Status IN ('Active', 'Inactive')),
    Is_a_customer BOOLEAN NOT NULL DEFAULT FALSE,
    Is_a_staff BOOLEAN NOT NULL DEFAULT FALSE
);
```

```
INSERT INTO Users (User_ID, User_LName, User_FName, User_MName, Email, Password,
                   Home_Address, City, Country, Role, Date_Registered, Account_Status,
                   Is_a_customer, Is_a_staff)

VALUES

    (101, 'Zaynab', 'Aboulkacem', NULL, 'ab.zaynab@example.com', 'pw123','Ifrane', 'Morocco', 'Staff',
                   '2024-01-01', 'Active', FALSE, TRUE),

    (102, 'Tanishka', 'Suwalka', NULL, 'su.tanishka@example.com', 'pw456','Ifrane', 'Morocco', 'Staff',
        '2024-01-02', 'Active', FALSE, TRUE),

    (103, 'Rita', 'Jalal', NULL, 'ja.rita@example.com', 'pw789', 'Ifrane', 'Morocco', 'Staff', '2023-01-01',
                   'Active', FALSE, TRUE),

    (104, 'Anass', 'Karim', NULL, 'an.karim@example.com', 'sepass','Ifrane', 'Morocco', 'Customer',
                   '2024-02-01', 'Active', TRUE, FALSE),

    (105, 'Dounia', 'Balloumi', NULL, 'ba.dounia@example.com', 'mypass', 'Ifrane', 'Morocco',
                   'Customer', '2024-02-02', 'Active', TRUE, FALSE),

    (106, 'Youssef', 'Ait Haddou', NULL, 'yo.ait@example.com', 'passw0rd','Ifrane', 'Morocco',
                   'Customer', '2024-02-03', 'Active', TRUE, FALSE),

    (107, 'Rayane', 'Mestari', 'Ahmed', 'me.rayane@example.com', 'wordpass','Ifrane', 'Morocco',
                   'Customer', '2024-02-04', 'Active', TRUE, FALSE),

    (108, 'Nour', 'El Idrissi', 'Fatima', 'no.elidrissi@example.com', 'pa55word', 'Ifrane', 'Morocco',
                   'Customer', '2024-02-05', 'Active', TRUE, FALSE),

    (109, 'Oumaima', 'Dehmane', NULL, 'de.oumaima@example.com', 'pass','Ifrane', 'Morocco',
                   'Customer', '2024-02-06', 'Inactive', TRUE, FALSE),
```

(110, 'Imane', 'Oussalim', 'Samira', 'im.oussa@example.com', 'securepass','Ifrane', 'Morocco',
'Customer', '2024-02-07', 'Active', TRUE, FALSE),

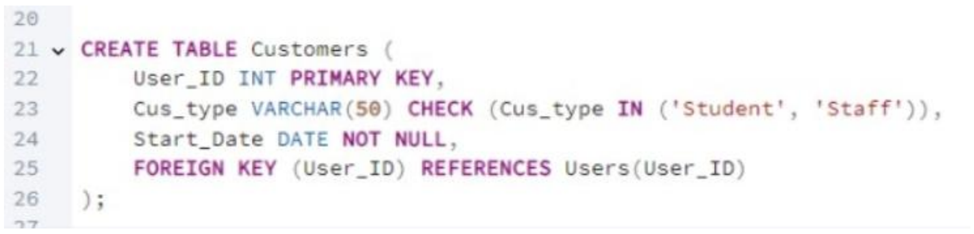
(111, 'Chaymae', 'Belabbar', NULL, 'be.chay@example.com', 'passxyz', 'Ifrane', 'Morocco',
'Customer', '2024-02-08', 'Inactive', TRUE, FALSE),

(112, 'Fatima', 'Ben Salah', 'Khadija', 'fa.bensalah@example.com', 'staffpass2024','Ifrane', 'Morocco',
'Staff', '2024-03-01', 'Active', FALSE, TRUE),

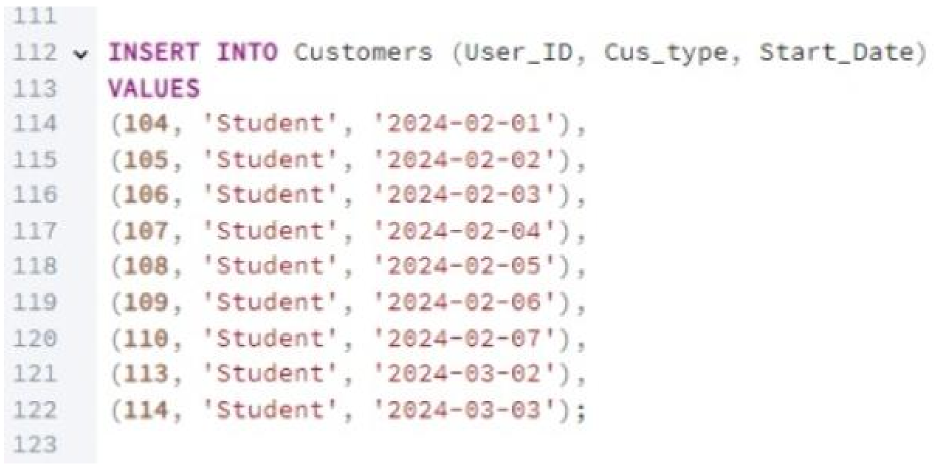
(113, 'Omar', 'El Amrani', NULL, 'om.elamrani@example.com', 'pass321', 'Ifrane', 'Morocco',
'Customer', '2024-03-02', 'Active', TRUE, FALSE),

(114, 'Lamia', 'Alaoui', NULL, 'la.alaoui@example.com', 'mypass55', 'Ifrane', 'Morocco', 'Customer',
'2024-03-03', 'Active', TRUE, FALSE);

- Table Customers

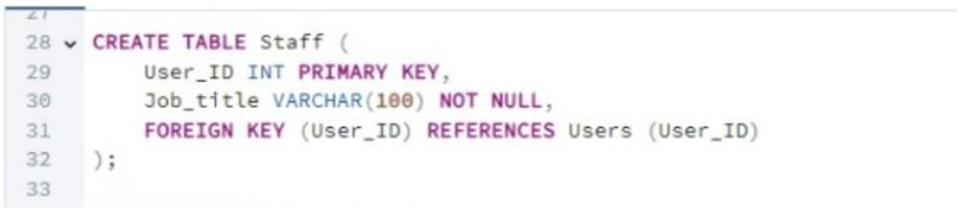


```
CREATE TABLE Customers (
    User_ID INT PRIMARY KEY,
    Cus_type VARCHAR(50) CHECK (Cus_type IN ('Student', 'Staff')),
    Start_Date DATE NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);
```

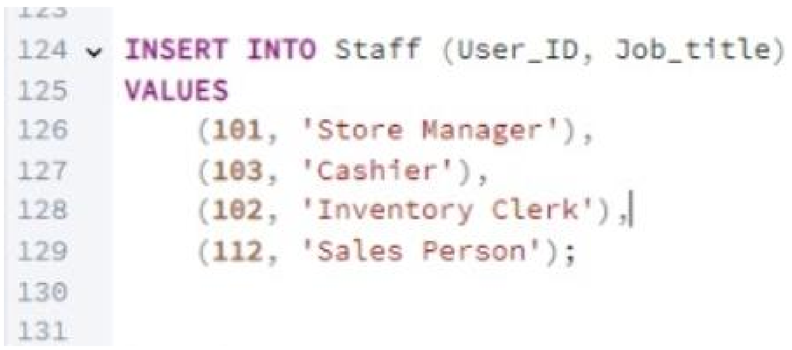


```
INSERT INTO Customers (User_ID, Cus_type, Start_Date)
VALUES
(104, 'Student', '2024-02-01'),
(105, 'Student', '2024-02-02'),
(106, 'Student', '2024-02-03'),
(107, 'Student', '2024-02-04'),
(108, 'Student', '2024-02-05'),
(109, 'Student', '2024-02-06'),
(110, 'Student', '2024-02-07'),
(113, 'Student', '2024-03-02'),
(114, 'Student', '2024-03-03');
```

- Table Staff

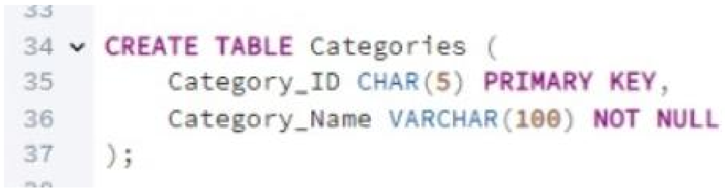


```sql
CREATE TABLE Staff (
    User_ID INT PRIMARY KEY,
    Job_title VARCHAR(100) NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES Users (User_ID)
);
```

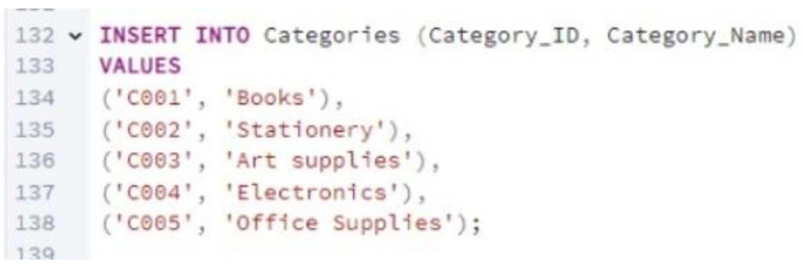


```sql
INSERT INTO Staff (User_ID, Job_title)
VALUES
    (101, 'Store Manager'),
    (103, 'Cashier'),
    (102, 'Inventory Clerk'),
    (112, 'Sales Person');
```

- Table Categories

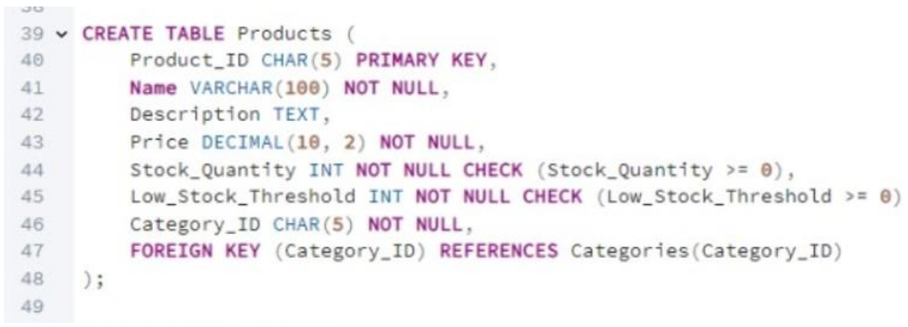


```sql
CREATE TABLE Categories (
    Category_ID CHAR(5) PRIMARY KEY,
    Category_Name VARCHAR(100) NOT NULL
);
```

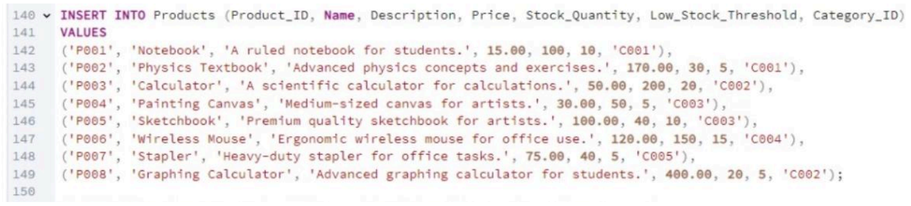


```sql
INSERT INTO Categories (Category_ID, Category_Name)
VALUES
('C001', 'Books'),
('C002', 'Stationery'),
('C003', 'Art supplies'),
('C004', 'Electronics'),
('C005', 'Office Supplies');
```

- Table Products

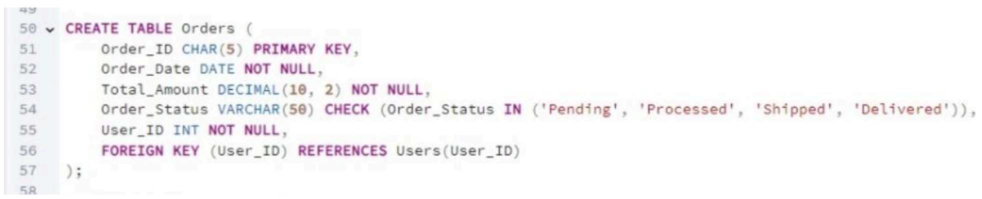


```sql
CREATE TABLE Products (
    Product_ID CHAR(5) PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Description TEXT,
    Price DECIMAL(10, 2) NOT NULL,
    Stock_Quantity INT NOT NULL CHECK (Stock_Quantity >= 0),
    Low_Stock_Threshold INT NOT NULL CHECK (Low_Stock_Threshold >= 0)
    Category_ID CHAR(5) NOT NULL,
    FOREIGN KEY (Category_ID) REFERENCES Categories(Category_ID)
);
```

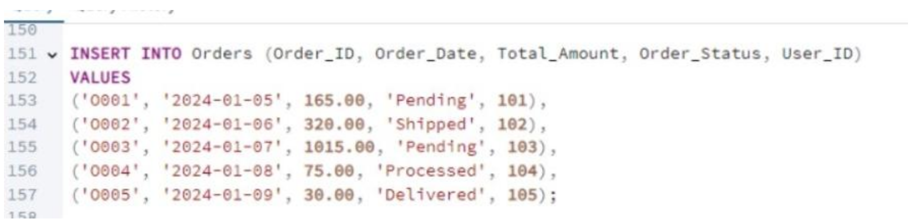


```sql
INSERT INTO Products (Product_ID, Name, Description, Price, Stock_Quantity, Low_Stock_Threshold, Category_ID)
VALUES
('P001', 'Notebook', 'A ruled notebook for students.', 15.00, 100, 10, 'C001'),
('P002', 'Physics Textbook', 'Advanced physics concepts and exercises.', 170.00, 30, 5, 'C001'),
('P003', 'Calculator', 'A scientific calculator for calculations.', 50.00, 200, 20, 'C002'),
('P004', 'Painting Canvas', 'Medium-sized canvas for artists.', 30.00, 50, 5, 'C003'),
('P005', 'Sketchbook', 'Premium quality sketchbook for artists.', 100.00, 40, 10, 'C003'),
('P006', 'Wireless Mouse', 'Ergonomic wireless mouse for office use.', 120.00, 150, 15, 'C004'),
('P007', 'Stapler', 'Heavy-duty stapler for office tasks.', 75.00, 40, 5, 'C005'),
('P008', 'Graphing Calculator', 'Advanced graphing calculator for students.', 400.00, 20, 5, 'C002');
```

- Table Orders

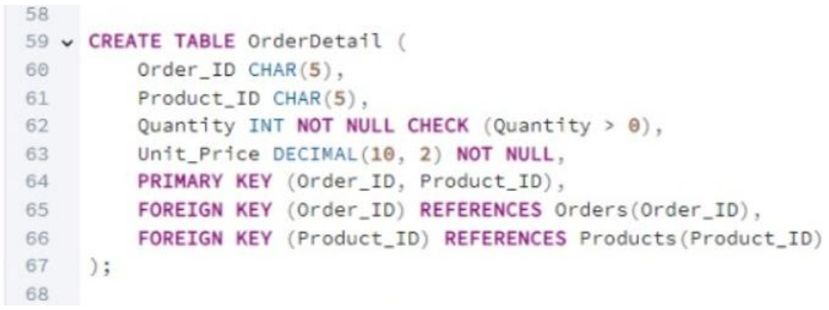


```sql
CREATE TABLE Orders (
    Order_ID CHAR(5) PRIMARY KEY,
    Order_Date DATE NOT NULL,
    Total_Amount DECIMAL(10, 2) NOT NULL,
    Order_Status VARCHAR(50) CHECK (Order_Status IN ('Pending', 'Processed', 'Shipped', 'Delivered')),
    User_ID INT NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);
```

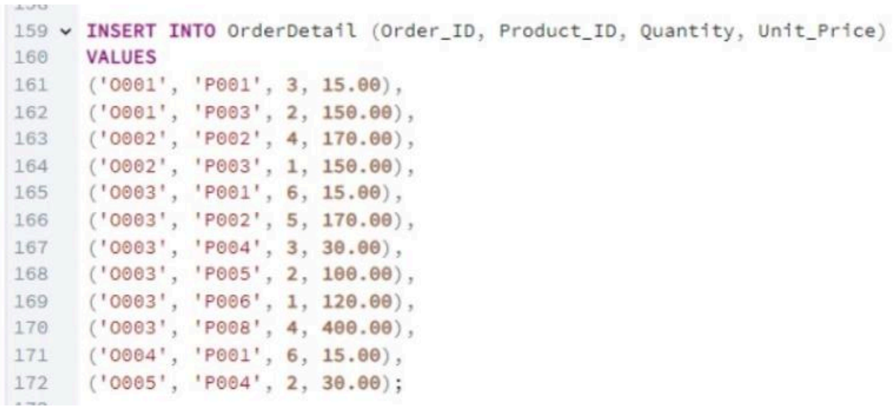


```sql
INSERT INTO Orders (Order_ID, Order_Date, Total_Amount, Order_Status, User_ID)
VALUES
('O001', '2024-01-05', 165.00, 'Pending', 101),
('O002', '2024-01-06', 320.00, 'Shipped', 102),
('O003', '2024-01-07', 1015.00, 'Pending', 103),
('O004', '2024-01-08', 75.00, 'Processed', 104),
('O005', '2024-01-09', 30.00, 'Delivered', 105);
```

- Table Orderdetail

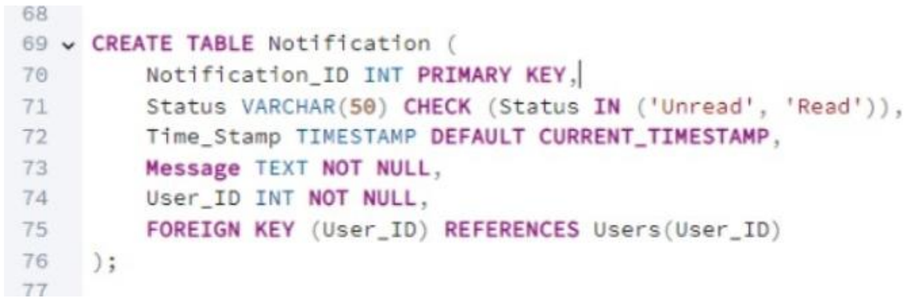


```sql
CREATE TABLE OrderDetail (
    Order_ID CHAR(5),
    Product_ID CHAR(5),
    Quantity INT NOT NULL CHECK (Quantity > 0),
    Unit_Price DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (Order_ID, Product_ID),
    FOREIGN KEY (Order_ID) REFERENCES Orders(Order_ID),
    FOREIGN KEY (Product_ID) REFERENCES Products(Product_ID)
);
```

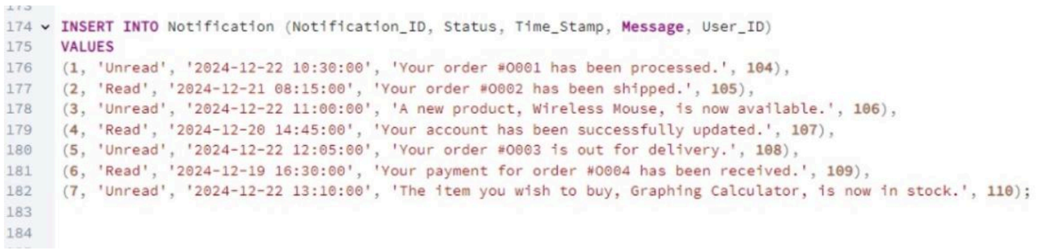


```sql
INSERT INTO OrderDetail (Order_ID, Product_ID, Quantity, Unit_Price)
VALUES
('O001', 'P001', 3, 15.00),
('O001', 'P003', 2, 150.00),
('O002', 'P002', 4, 170.00),
('O002', 'P003', 1, 150.00),
('O003', 'P001', 6, 15.00),
('O003', 'P002', 5, 170.00),
('O003', 'P004', 3, 30.00),
('O003', 'P005', 2, 100.00),
('O003', 'P006', 1, 120.00),
('O003', 'P008', 4, 400.00),
('O004', 'P001', 6, 15.00),
('O005', 'P004', 2, 30.00);
```

- Table Notification



```sql
CREATE TABLE Notification (
    Notification_ID INT PRIMARY KEY,
    Status VARCHAR(50) CHECK (Status IN ('Unread', 'Read')),
    Time_Stamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Message TEXT NOT NULL,
    User_ID INT NOT NULL,
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);
```



```sql
INSERT INTO Notification (Notification_ID, Status, Time_Stamp, Message, User_ID)
VALUES
(1, 'Unread', '2024-12-22 10:30:00', 'Your order #O001 has been processed.', 104),
(2, 'Read', '2024-12-21 08:15:00', 'Your order #O002 has been shipped.', 105),
(3, 'Unread', '2024-12-22 11:00:00', 'A new product, Wireless Mouse, is now available.', 106),
(4, 'Read', '2024-12-20 14:45:00', 'Your account has been successfully updated.', 107),
(5, 'Unread', '2024-12-22 12:05:00', 'Your order #O003 is out for delivery.', 108),
(6, 'Read', '2024-12-19 16:30:00', 'Your payment for order #O004 has been received.', 109),
(7, 'Unread', '2024-12-22 13:10:00', 'The item you wish to buy, Graphing Calculator, is now in stock.', 110);
```

B. SQL Queries and Procedural SQL

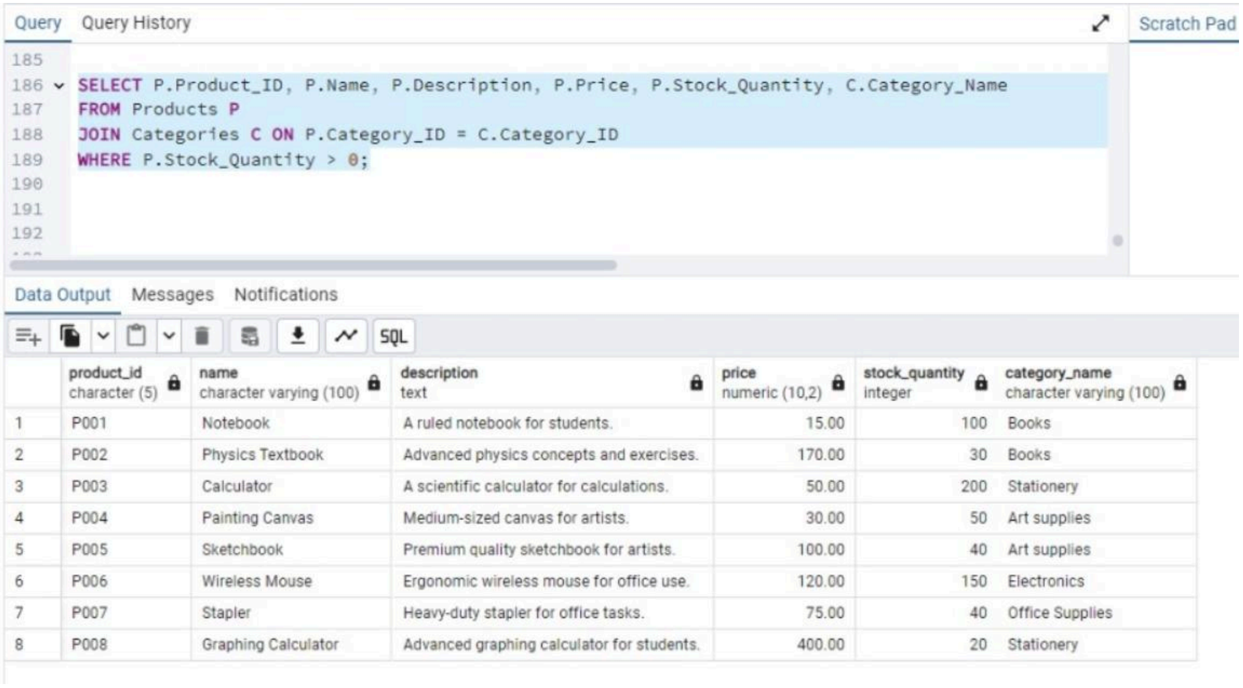
a. SQL Queries to Implement the Requirements/Functionalities of Your DB Application

I. Retrieval Queries

- **Query to View Products in Stock**

Business Rule: Customers can only view products that are currently in stock.

Keeps the product catalog accurate, helping customers avoid ordering out-of-stock items.



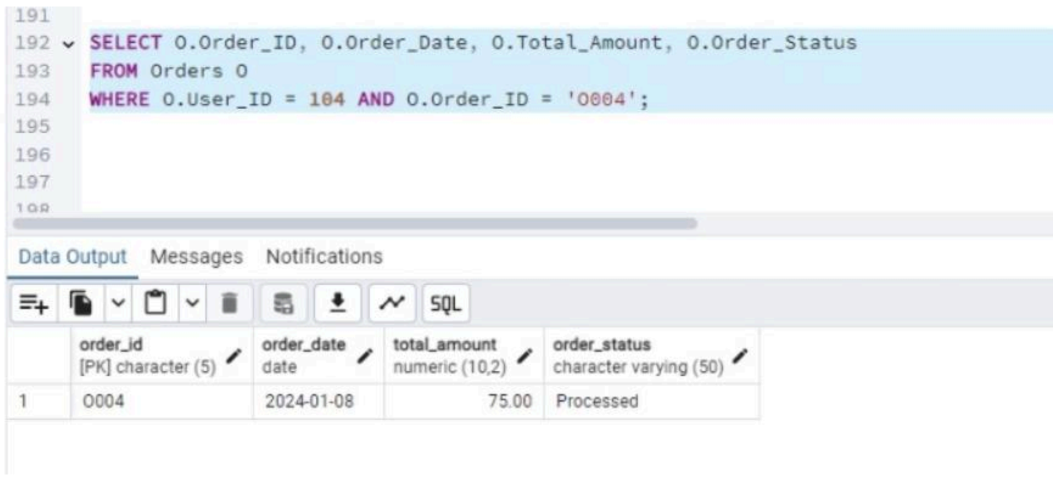
```sql
SELECT P.Product_ID, P.Name, P.Description, P.Price, P.Stock_Quantity, C.Category_Name
FROM Products P
JOIN Categories C ON P.Category_ID = C.Category_ID
WHERE P.Stock_Quantity > 0;
```

	product_id character (5)	name character varying (100)	description text	price numeric (10,2)	stock_quantity integer	category_name character varying (100)
1	P001	Notebook	A ruled notebook for students.	15.00	100	Books
2	P002	Physics Textbook	Advanced physics concepts and exercises.	170.00	30	Books
3	P003	Calculator	A scientific calculator for calculations.	50.00	200	Stationery
4	P004	Painting Canvas	Medium-sized canvas for artists.	30.00	50	Art supplies
5	P005	Sketchbook	Premium quality sketchbook for artists.	100.00	40	Art supplies
6	P006	Wireless Mouse	Ergonomic wireless mouse for office use.	120.00	150	Electronics
7	P007	Stapler	Heavy-duty stapler for office tasks.	75.00	40	Office Supplies
8	P008	Graphing Calculator	Advanced graphing calculator for students.	400.00	20	Stationery

- **Query to Track Order Status**

Business Rule: Customers should be able to view the current status of their orders

Provides transparency to customers about their order's progress, enhancing customer experience.



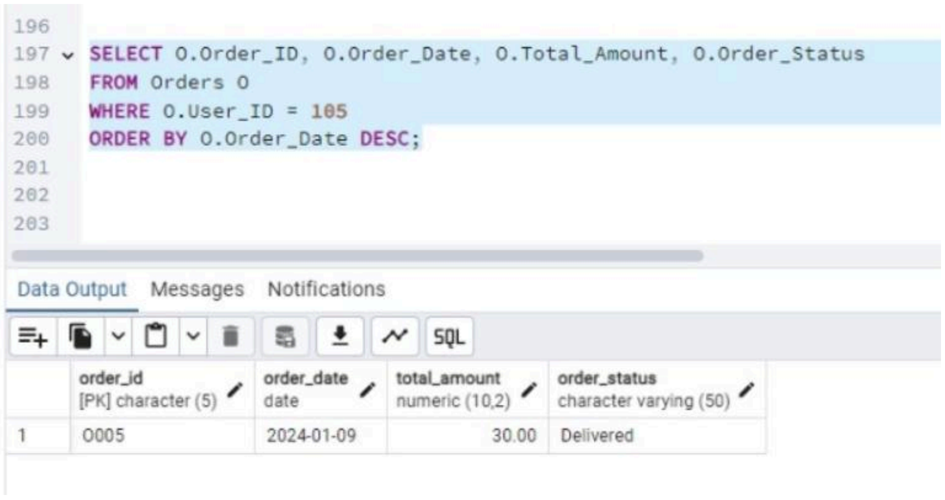
```sql
SELECT O.Order_ID, O.Order_Date, O.Total_Amount, O.Order_Status
FROM Orders O
WHERE O.User_ID = 104 AND O.Order_ID = 'O004';
```

	order_id [PK] character (5)	order_date date	total_amount numeric (10,2)	order_status character varying (50)
1	O004	2024-01-08	75.00	Processed

- **Query to List Orders by Customer**

Business Rule: A customer can view all their previous orders.

Essential for customers to review their past purchases, which is often used for reordering or tracking.



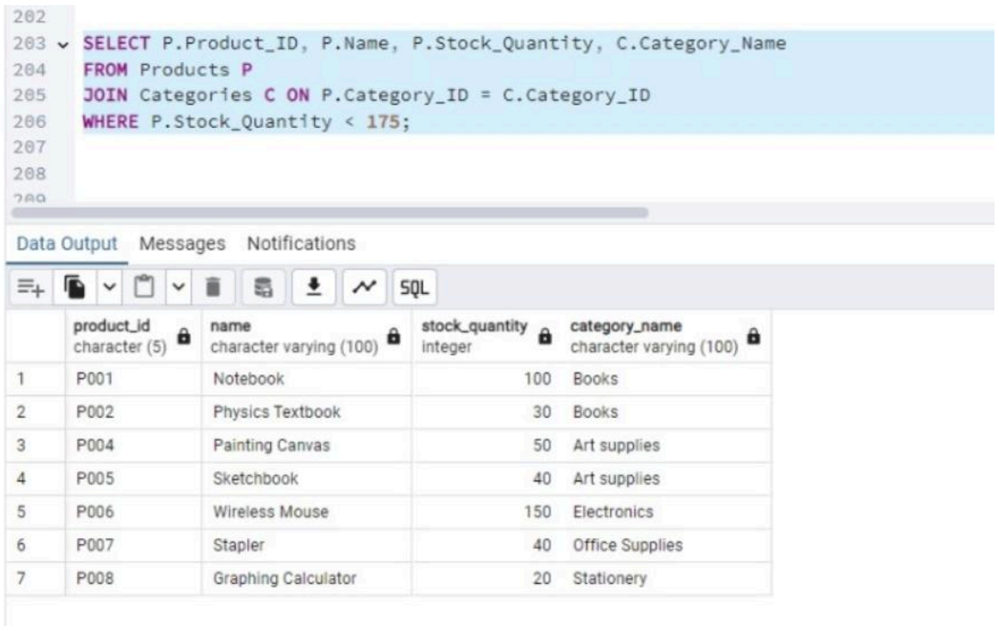
```sql
SELECT O.Order_ID, O.Order_Date, O.Total_Amount, O.Order_Status
FROM Orders O
WHERE O.User_ID = 105
ORDER BY O.Order_Date DESC;
```

	order_id [PK] character (5)	order_date date	total_amount numeric (10,2)	order_status character varying (50)
1	O005	2024-01-09	30.00	Delivered

- **Query to View Low Stock Products**

Business Rule: The system must identify products that are low in stock (e.g., under 175 units).

This query helps manage inventory efficiently, reducing the risk of stockouts and ensuring popular items are always available.



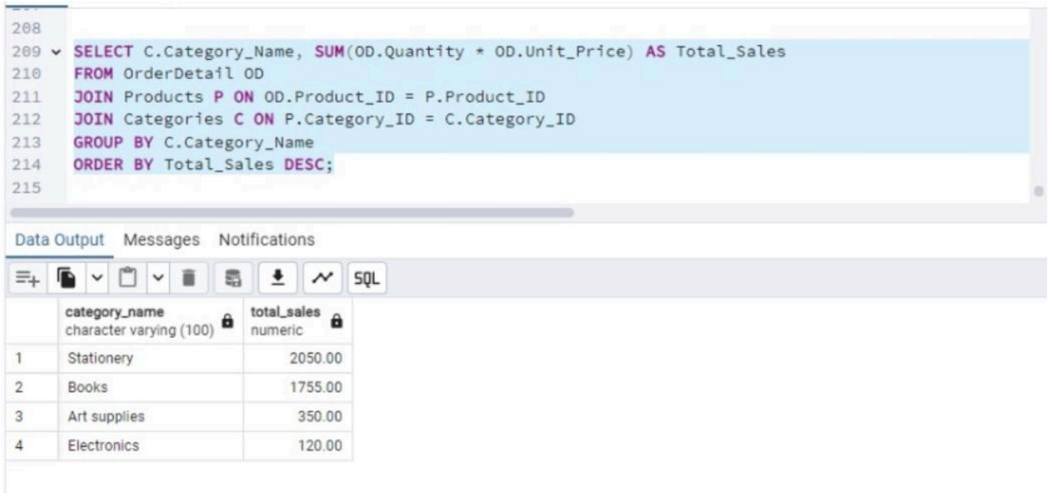
```sql
SELECT P.Product_ID, P.Name, P.Stock_Quantity, C.Category_Name
FROM Products P
JOIN Categories C ON P.Category_ID = C.Category_ID
WHERE P.Stock_Quantity < 175;
```

	product_id character (5)	name character varying (100)	stock_quantity integer	category_name character varying (100)
1	P001	Notebook	100	Books
2	P002	Physics Textbook	30	Books
3	P004	Painting Canvas	50	Art supplies
4	P005	Sketchbook	40	Art supplies
5	P006	Wireless Mouse	150	Electronics
6	P007	Stapler	40	Office Supplies
7	P008	Graphing Calculator	20	Stationery

- **Query to View Sales by Product Category**

Business Rule: The business needs to track total sales by product category for performance analysis.

Important for business analysis, helping to optimize inventory, marketing, and sales strategies by understanding which categories generate the most revenue.



```sql
SELECT C.Category_Name, SUM(OD.Quantity * OD.Unit_Price) AS Total_Sales
FROM OrderDetail OD
JOIN Products P ON OD.Product_ID = P.Product_ID
JOIN Categories C ON P.Category_ID = C.Category_ID
GROUP BY C.Category_Name
ORDER BY Total_Sales DESC;
```

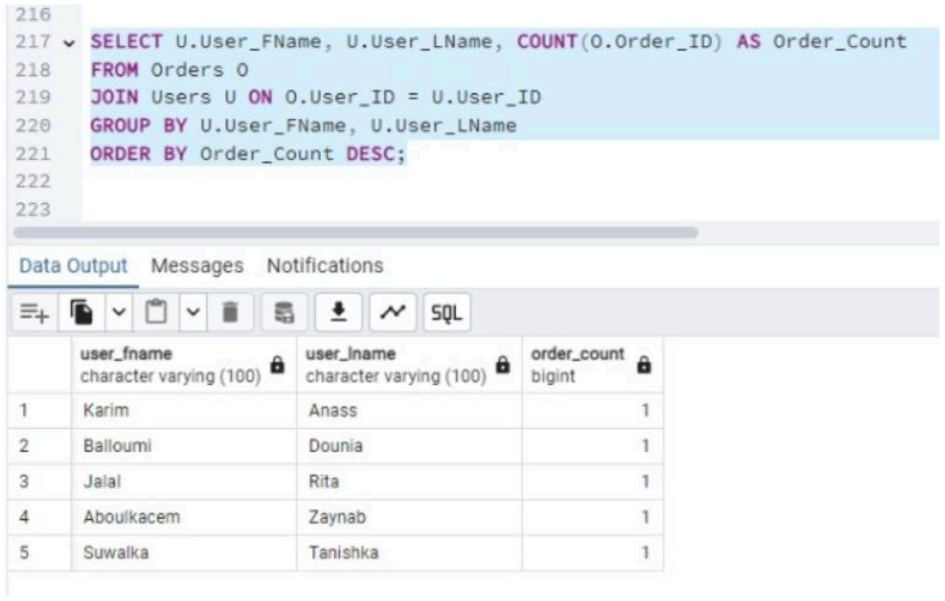
	category_name character varying (100)	total_sales numeric
1	Stationery	2050.00
2	Books	1755.00
3	Art supplies	350.00
4	Electronics	120.00

i. Aggregate and Grouping Queries

- **Query to Count Orders per Customer**

Business Rule: The platform must track how many orders each customer has placed

It provides a way for the business to identify active customers and analyze their purchase behavior.



```
SELECT U.User_FName, U.User_LName, COUNT(O.Order_ID) AS Order_Count
FROM Orders O
JOIN Users U ON O.User_ID = U.User_ID
GROUP BY U.User_FName, U.User_LName
ORDER BY Order_Count DESC;
```

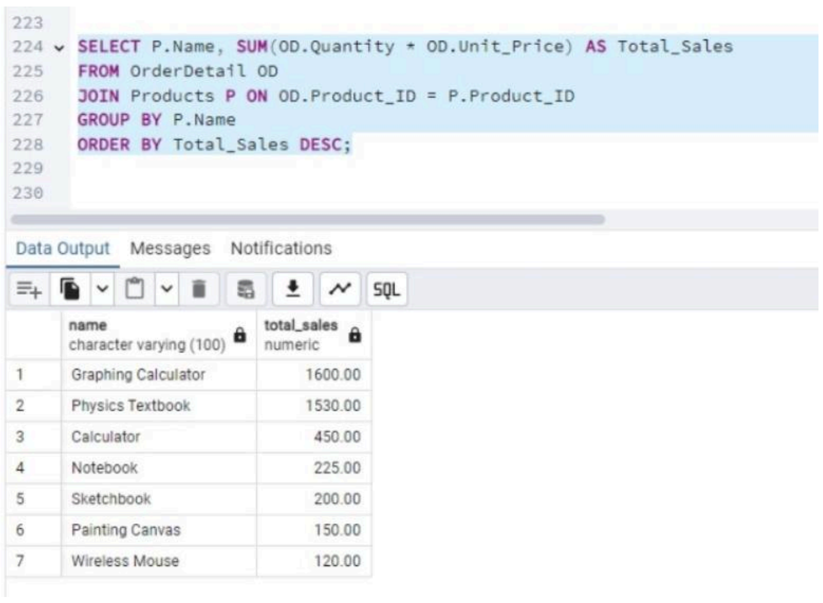
Data Output Messages Notifications

	user_fname character varying (100)	user_lname character varying (100)	order_count bigint
1	Karim	Anass	1
2	Balloumi	Dounia	1
3	Jalal	Rita	1
4	Aboulkacem	Zaynab	1
5	Suwalka	Tanishka	1

- **Query to Calculate Total Sales by Product**

Business Rule: The system must calculate total sales for each product based on the quantities ordered.

It provides insights into the financial performance of each product, helping businesses understand which products are the most profitable.



```sql
SELECT P.Name, SUM(OD.Quantity * OD.Unit_Price) AS Total_Sales
FROM OrderDetail OD
JOIN Products P ON OD.Product_ID = P.Product_ID
GROUP BY P.Name
ORDER BY Total_Sales DESC;
```

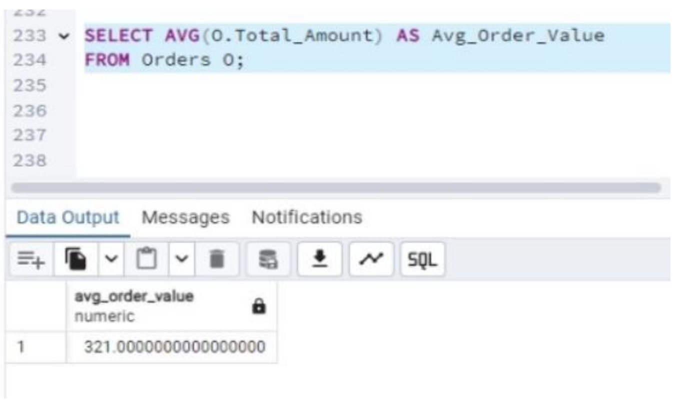
Data Output Messages Notifications

	name character varying (100)	total_sales numeric
1	Graphing Calculator	1600.00
2	Physics Textbook	1530.00
3	Calculator	450.00
4	Notebook	225.00
5	Sketchbook	200.00
6	Painting Canvas	150.00
7	Wireless Mouse	120.00

- **Query to Calculate Average Order Value**

  Business Rule: The platform should track the average order value for all orders

  Understanding the average order value is crucial for pricing, promotions, and sales strategy.



```sql
SELECT AVG(O.Total_Amount) AS Avg_Order_Value
FROM Orders O;
```

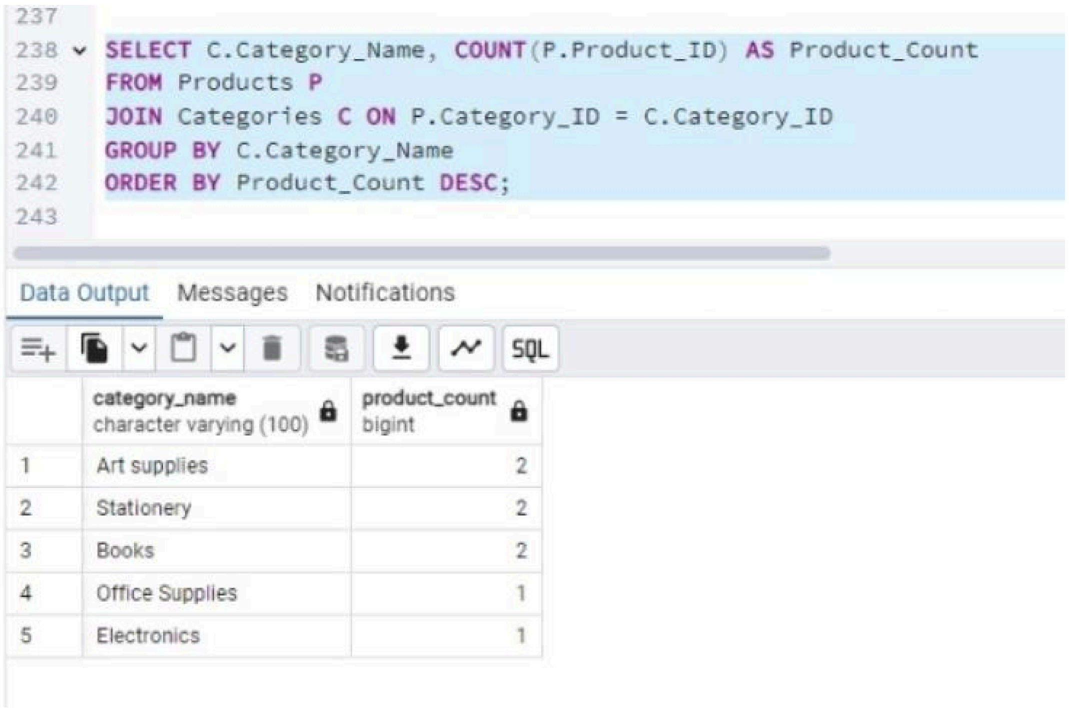
Data Output Messages Notifications

	avg_order_value numeric
1	321.0000000000000000

- **Query to Count Products in Each Category**

  Business Rule: The catalog must be categorized by product types, and the system must track how many products exist in each category.

It helps inventory managers understand the distribution of products across categories, ensuring there's a balanced catalog.



```sql
SELECT C.Category_Name, COUNT(P.Product_ID) AS Product_Count
FROM Products P
JOIN Categories C ON P.Category_ID = C.Category_ID
GROUP BY C.Category_Name
ORDER BY Product_Count DESC;
```

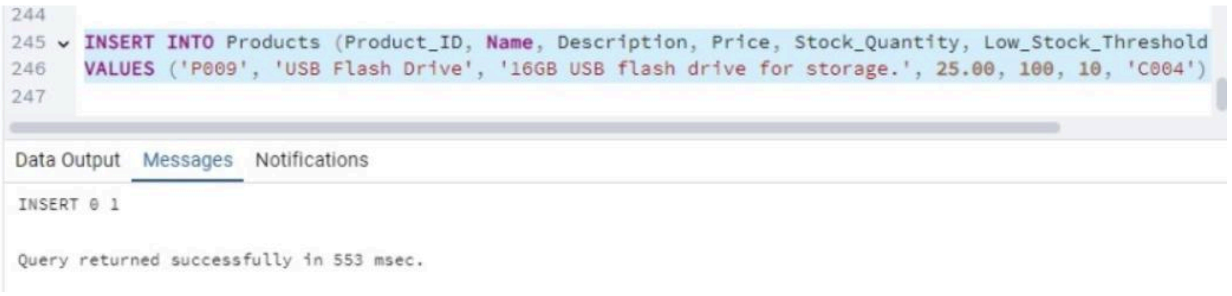
Data Output Messages Notifications

	category_name character varying (100)	product_count bigint
1	Art supplies	2
2	Stationery	2
3	Books	2
4	Office Supplies	1
5	Electronics	1

b. **SQL Queries for Data Manipulation Operations (Insert, Update, and Delete)**

- **Insert New Product into Catalog**

Business Rule: Every item in the catalog must have a unique product ID, name, category, price, and stock quantity.



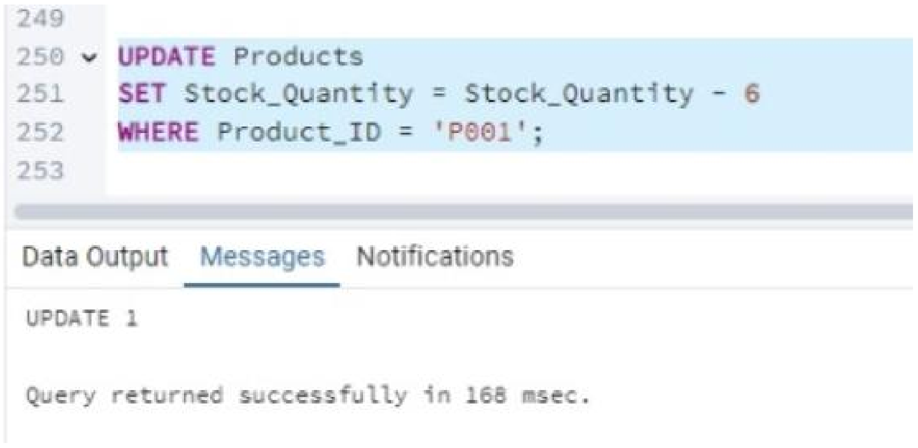
```sql
INSERT INTO Products (Product_ID, Name, Description, Price, Stock_Quantity, Low_Stock_Threshold
VALUES ('P009', 'USB Flash Drive', '16GB USB flash drive for storage.', 25.00, 100, 10, 'C004')
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 553 msec.
```

- **Update Product Stock after Order**

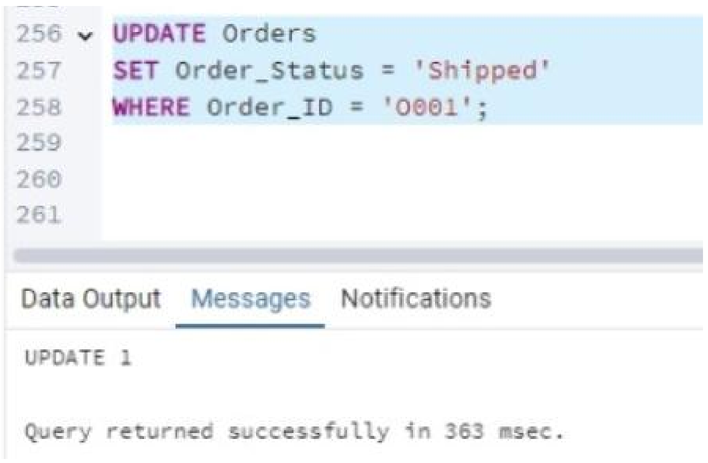
Business Rule: The system must automatically reduce the stock quantity of ordered items upon successful order placement.


```
UPDATE Products
SET Stock_Quantity = Stock_Quantity - 6
WHERE Product_ID = 'P001';
```

```
UPDATE 1

Query returned successfully in 168 msec.
```

- **Update Order Status**

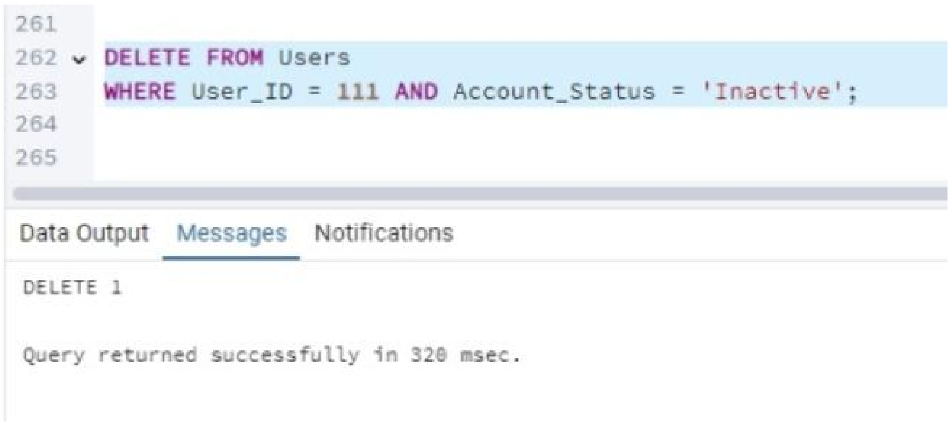
Business Rule: Orders can have one of the following statuses: "Pending," "Processed," "Shipped," or "Delivered."


```
UPDATE Orders
SET Order_Status = 'Shipped'
WHERE Order_ID = 'O001';
```

```
UPDATE 1

Query returned successfully in 363 msec.
```

- **Delete Inactive User**

  Business Rule: Users who are inactive for a long period might need to be removed from the system.



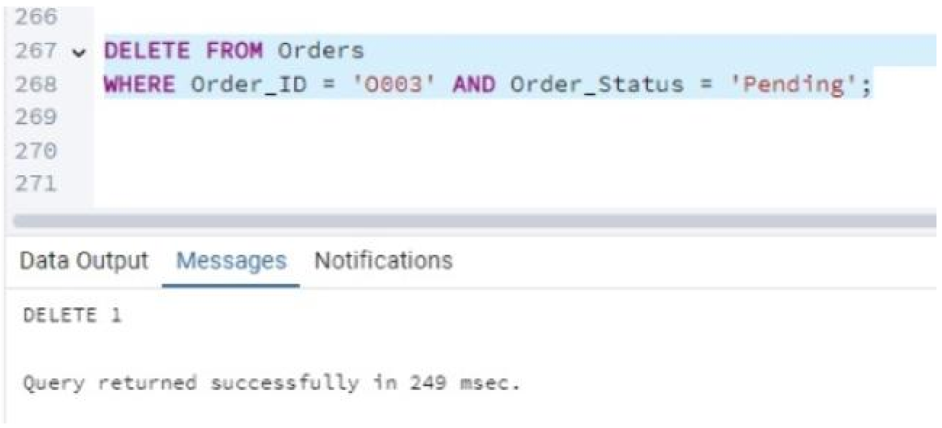
```
DELETE FROM Users
WHERE User_ID = 111 AND Account_Status = 'Inactive';
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 320 msec.
```

- **Delete an Order**

  Business Rule: Users can only place an order if all items in the cart are in stock, and orders that are canceled need to be deleted**.**



```
DELETE FROM Orders
WHERE Order_ID = 'O003' AND Order_Status = 'Pending';
```

Data Output | Messages | Notifications

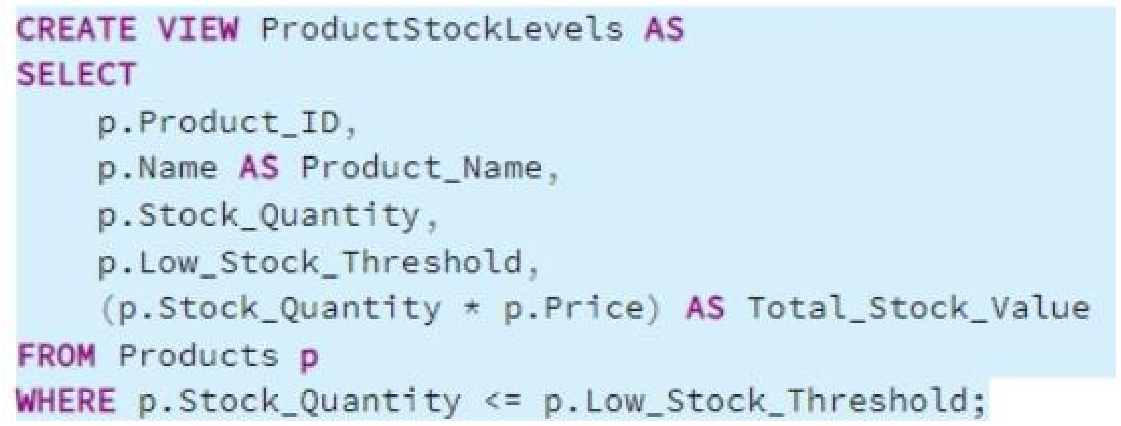
```
DELETE 1

Query returned successfully in 249 msec.
```

c. **Views**

- **View for Product Stock Levels and Low Stock Items**

  This view allows the staff to easily identify products that are running low on stock and the total value of each product's stock. This view is especially important for inventory management.

  This view is critical for inventory management, as it helps the staff or administrators quickly identify products that need to be reordered or restocked, helping to avoid running out of essential items in the catalog.

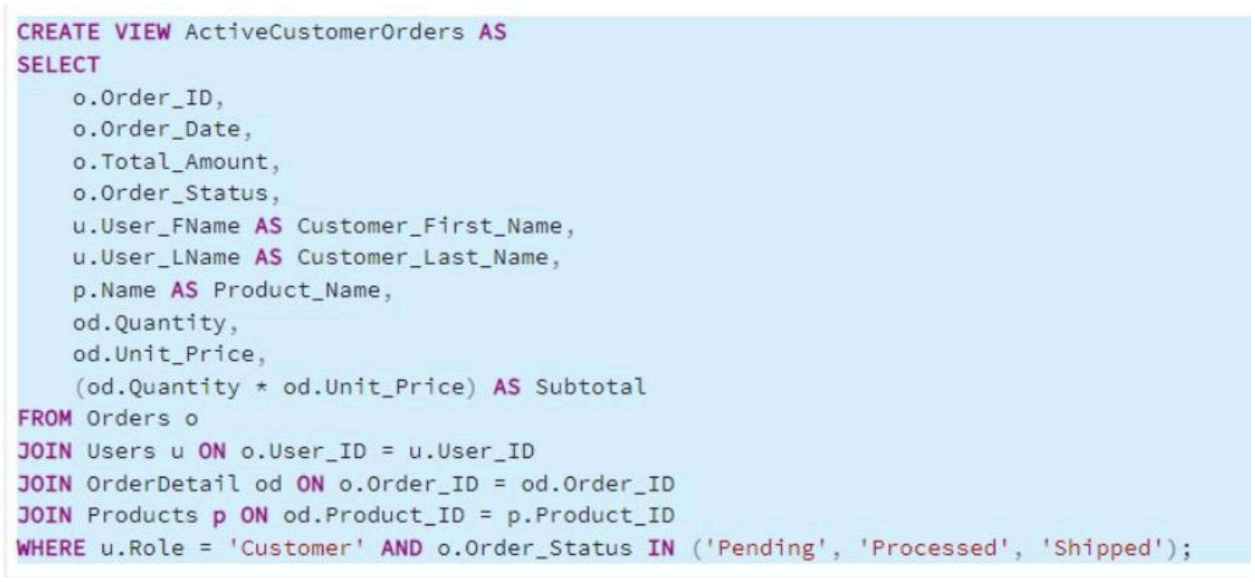


```sql
CREATE VIEW ProductStockLevels AS
SELECT
    p.Product_ID,
    p.Name AS Product_Name,
    p.Stock_Quantity,
    p.Low_Stock_Threshold,
    (p.Stock_Quantity * p.Price) AS Total_Stock_Value
FROM Products p
WHERE p.Stock_Quantity <= p.Low_Stock_Threshold;
```

- **View to Show Active Customer Orders**

  This view will help administrators or staff easily view the active orders placed by customers, including order details, total amounts, and order statuses.

  By creating this view, staff can easily track and manage active orders without needing to join multiple tables for each query. It provides a simplified, user-friendly view of orders that need attention.

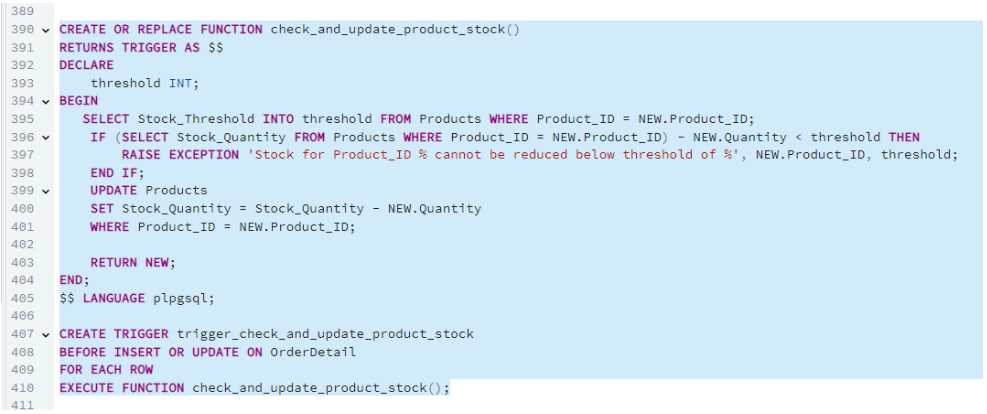


```sql
CREATE VIEW ActiveCustomerOrders AS
SELECT
    o.Order_ID,
    o.Order_Date,
    o.Total_Amount,
    o.Order_Status,
    u.User_FName AS Customer_First_Name,
    u.User_LName AS Customer_Last_Name,
    p.Name AS Product_Name,
    od.Quantity,
    od.Unit_Price,
    (od.Quantity * od.Unit_Price) AS Subtotal
FROM Orders o
JOIN Users u ON o.User_ID = u.User_ID
JOIN OrderDetail od ON o.Order_ID = od.Order_ID
JOIN Products p ON od.Product_ID = p.Product_ID
WHERE u.Role = 'Customer' AND o.Order_Status IN ('Pending', 'Processed', 'Shipped');
```

d. Stored Procedures, Stored Functions, and Triggers

- **Trigger: Check Threshold and Update Stock**

  Ensures product stock levels remain above a defined threshold before processing an order. Validates that the stock won't fall below the threshold after the order quantity is deducted.Raises an exception to block orders that would breach the threshold.Updates the stock by subtracting the order quantity if validation passes.Maintains accurate inventory levels and prevents stockouts.

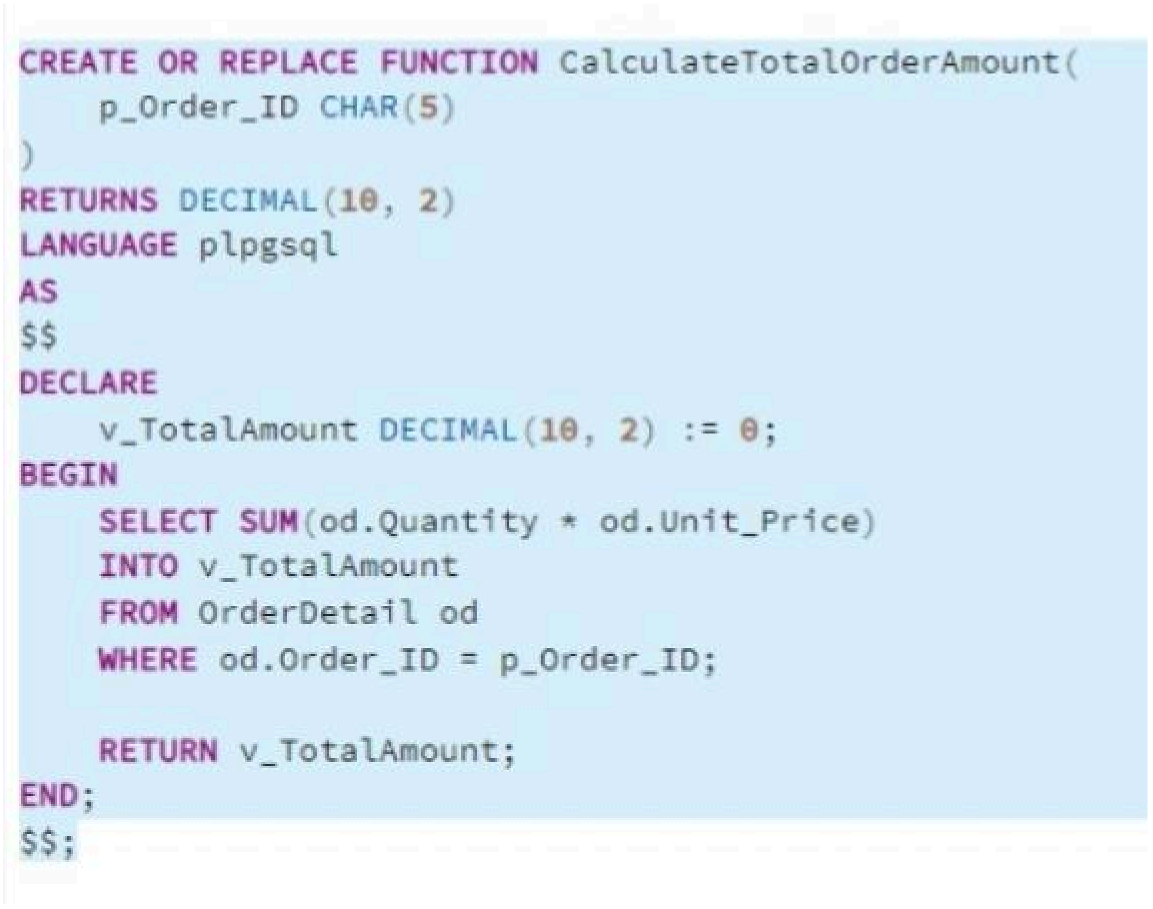


```sql
CREATE OR REPLACE FUNCTION check_and_update_product_stock()
RETURNS TRIGGER AS $$
DECLARE
    threshold INT;
BEGIN
   SELECT Stock_Threshold INTO threshold FROM Products WHERE Product_ID = NEW.Product_ID;
    IF (SELECT Stock_Quantity FROM Products WHERE Product_ID = NEW.Product_ID) - NEW.Quantity < threshold THEN
        RAISE EXCEPTION 'Stock for Product_ID % cannot be reduced below threshold of %', NEW.Product_ID, threshold;
    END IF;
    UPDATE Products
    SET Stock_Quantity = Stock_Quantity - NEW.Quantity
    WHERE Product_ID = NEW.Product_ID;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_and_update_product_stock
BEFORE INSERT OR UPDATE ON OrderDetail
FOR EACH ROW
EXECUTE FUNCTION check_and_update_product_stock();
```

- **Stored Function: Calculate Total Order Amount**

This stored function will calculate the total amount for a specific order based on the quantities and prices of the products in the order.

This function simplifies the calculation of the total order amount for any given order, consolidating the logic into a reusable function.

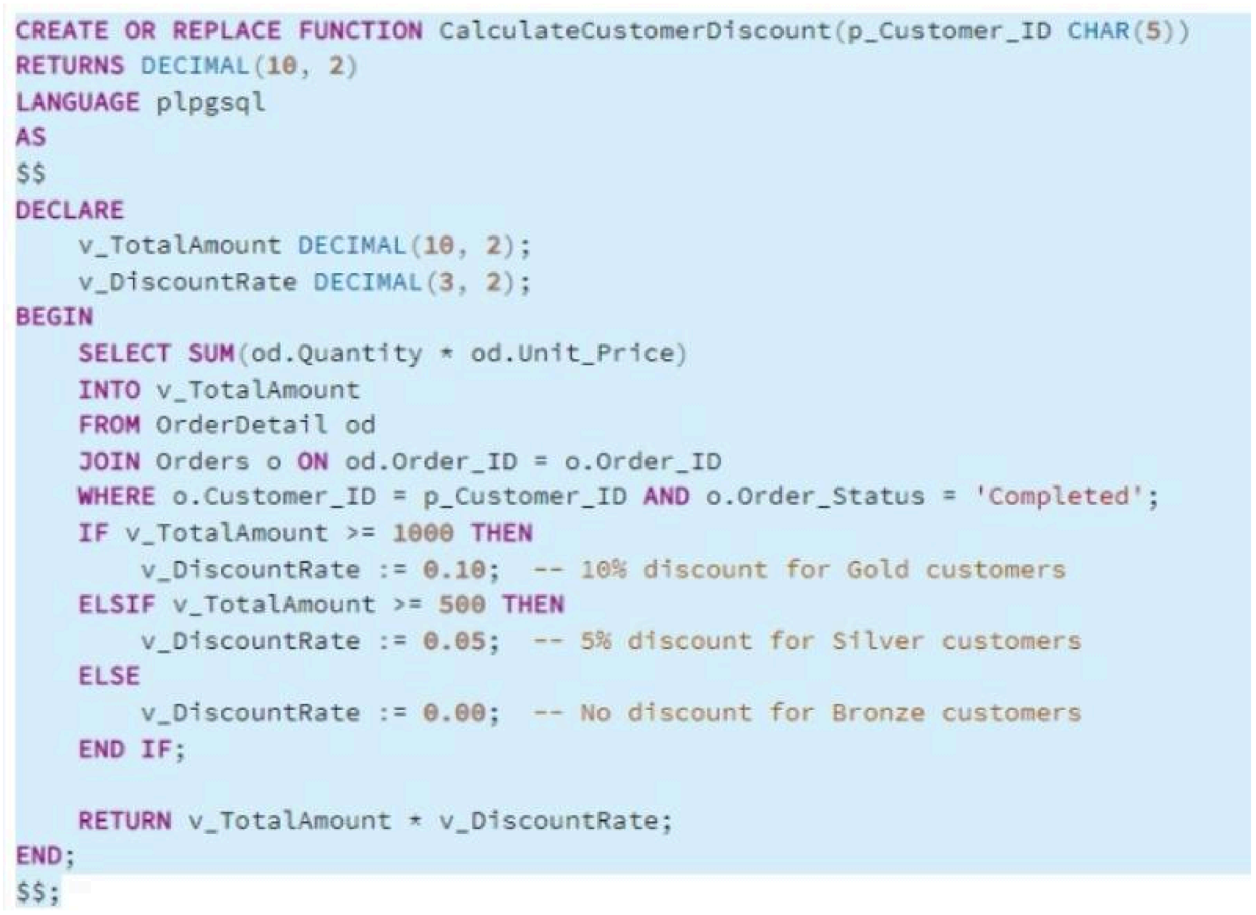


```
CREATE OR REPLACE FUNCTION CalculateTotalOrderAmount(
    p_Order_ID CHAR(5)
)
RETURNS DECIMAL(10, 2)
LANGUAGE plpgsql
AS
$$
DECLARE
    v_TotalAmount DECIMAL(10, 2) := 0;
BEGIN
    SELECT SUM(od.Quantity * od.Unit_Price)
    INTO v_TotalAmount
    FROM OrderDetail od
    WHERE od.Order_ID = p_Order_ID;

    RETURN v_TotalAmount;
END;
$$;
```

- **Stored Function: Calculate Customer Discount**

  This stored function will calculate the discount a customer is eligible for based on their total order amount.

  This function calculates the discount for a customer based on their total order amount and tier. It helps in automating the discount calculation process.

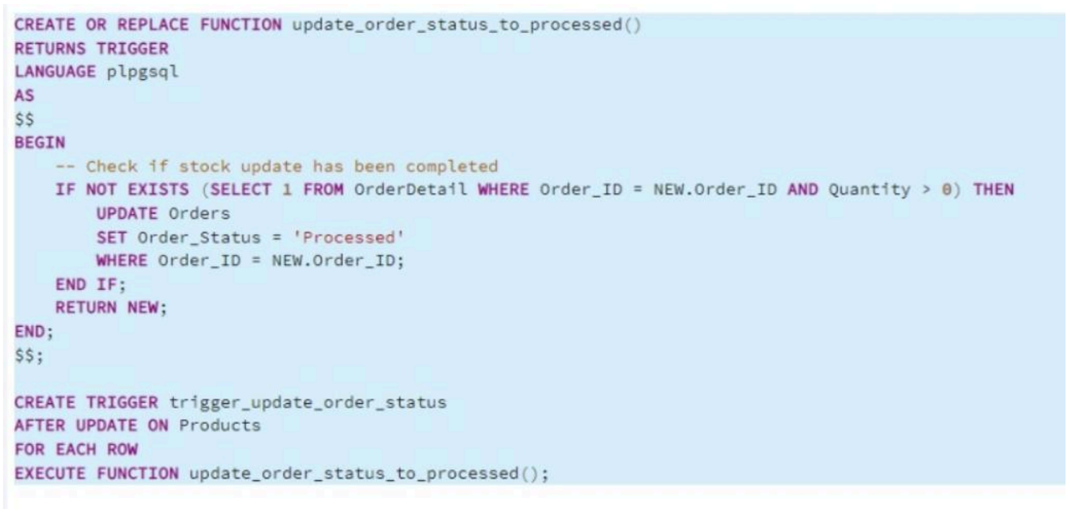


```sql
CREATE OR REPLACE FUNCTION CalculateCustomerDiscount(p_Customer_ID CHAR(5))
RETURNS DECIMAL(10, 2)
LANGUAGE plpgsql
AS
$$
DECLARE
    v_TotalAmount DECIMAL(10, 2);
    v_DiscountRate DECIMAL(3, 2);
BEGIN
    SELECT SUM(od.Quantity * od.Unit_Price)
    INTO v_TotalAmount
    FROM OrderDetail od
    JOIN Orders o ON od.Order_ID = o.Order_ID
    WHERE o.Customer_ID = p_Customer_ID AND o.Order_Status = 'Completed';
    IF v_TotalAmount >= 1000 THEN
        v_DiscountRate := 0.10;  -- 10% discount for Gold customers
    ELSIF v_TotalAmount >= 500 THEN
        v_DiscountRate := 0.05;  -- 5% discount for Silver customers
    ELSE
        v_DiscountRate := 0.00;  -- No discount for Bronze customers
    END IF;

    RETURN v_TotalAmount * v_DiscountRate;
END;
$$;
```

- **Trigger: Automatically Update Order Status to "Processed"**

  This trigger will automatically update the order status to "Processed" once the stock quantities are updated following an order placement.

  This trigger helps in automatically changing the order status to "Processed" after stock has been updated for the products in the order.

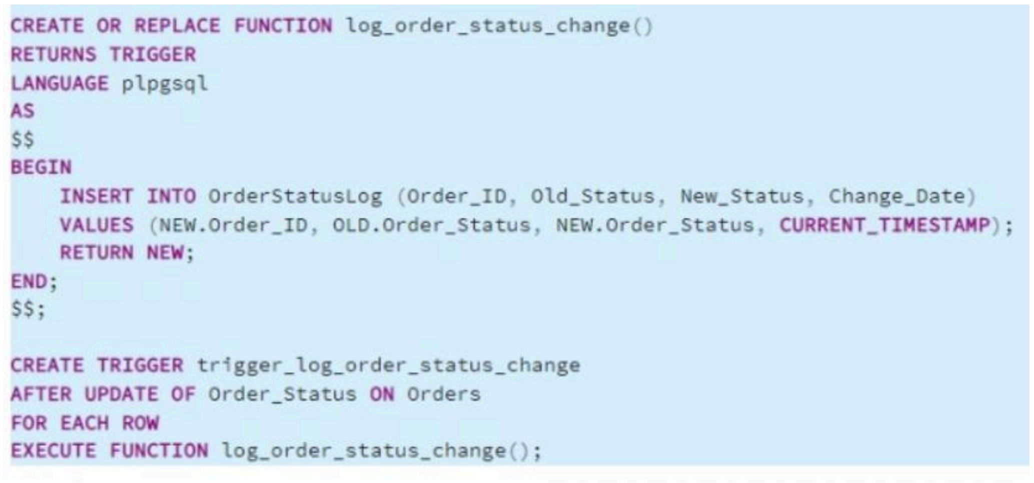


```sql
CREATE OR REPLACE FUNCTION update_order_status_to_processed()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN
    -- Check if stock update has been completed
    IF NOT EXISTS (SELECT 1 FROM OrderDetail WHERE Order_ID = NEW.Order_ID AND Quantity > 0) THEN
        UPDATE Orders
        SET Order_Status = 'Processed'
        WHERE Order_ID = NEW.Order_ID;
    END IF;
    RETURN NEW;
END;
$$;

CREATE TRIGGER trigger_update_order_status
AFTER UPDATE ON Products
FOR EACH ROW
EXECUTE FUNCTION update_order_status_to_processed();
```

- **Trigger: Log Order Status Changes**

  This trigger logs any changes to the status of an order into an OrderStatusLog table, providing an audit trail for status changes.

  This trigger ensures that every change to an order's status is logged in the OrderStatusLog table, providing an audit trail for tracking status changes over time.



```sql
CREATE OR REPLACE FUNCTION log_order_status_change()
RETURNS TRIGGER
LANGUAGE plpgsql
AS
$$
BEGIN
    INSERT INTO OrderStatusLog (Order_ID, Old_Status, New_Status, Change_Date)
    VALUES (NEW.Order_ID, OLD.Order_Status, NEW.Order_Status, CURRENT_TIMESTAMP);
    RETURN NEW;
END;
$$;

CREATE TRIGGER trigger_log_order_status_change
AFTER UPDATE OF Order_Status ON Orders
FOR EACH ROW
EXECUTE FUNCTION log_order_status_change();
```

c. Application architecture: technology and tools used and their interaction/user-friendly interface

An efficient interface to these procedures and functions regarding PostgreSQL will be created using a Python application based on Tkinter. This design architecture ensures seamless integration of the database with an easily accessible user interface.

Technology Stack

- Database Management:
- **PostgreSQL**: A powerful open-source relational database to store users, products, orders, order details and notifications in a reliable manner.

- Programming Language:
- **Python**: Python's flexible nature facilitates streamlined application development and is easily integrated into several other libraries.

- GUI Framework:
- **Tkinter**: An in-built GUI toolkit for Python, making it easy to create user-friendly forms and interactions.

Integration and Interaction

- Database Connection:
- **psycopg2:** This adapter allows the application to connect with PostgreSQL, run SQL queries, and manage transactions in an efficient way.

- Application Logic:
- Custom Python functions are created for user input from the Tkinter interface, processing of data, and performing the required database-related operations using psycopg2.

- User Actions:
- The application responds from user events by firing functions interacting with the database to make the interface dynamic and responsive.

User-Friendly Interface

- Design Features:
  - The application has easy navigation, with clear menus and buttons facilitating the user conducting important tasks such as ordering, product management, and the like. It gives pop-up alert feedback (the tkinter messagebox) for successful actions and errors. In addition, there are input validations to preserve the integrity of the data and display useful error messages when needed.

- Widget Usage:
  - The application employs a variety of Tkinter widgets, such as Entry, Label, and Button, to establish a clean layout with well-structured forms.

Python code implementation:

After reviewing all aspects of the interface, we are satisfied with its current design and functionality. No further adjustments are necessary at this time.

The **Customer Order Management Interface** is a Tkinter-based desktop application designed to manage products, customer orders, and order details for an e-commerce platform. The interface is organized into several functional tabs, each focusing on a specific task. Here's an overview of the main features and functionality:

1. Products Tab

- **View Products**: Displays a list of available products with details like Product ID, Name, and Price. This information is fetched from the database and displayed in a treeview widget.

2. Place Order Tab

- **Place an Order**: Customers can place orders by providing their **User ID**, **Product ID**, and **Quantity**. The system checks if the user and product exist, calculates the total amount based on the quantity and product price, and generates a random **Order ID**. Afterward, the order is saved in the database under **Orders** and **OrderDetail** tables.

3. User Info & Orders Tab

- **View User Info**: Allows users to view their profile information, including **Name**, **Email**, **Account Status**, and **Date Registered**.
- **View Orders**: Displays a list of orders associated with the user, including **Order ID**, **Order Date**, **Total Amount**, and **Order Status**.
- **Total Orders**: Shows the total number of orders placed by the user.
- **Sales by Product**: Displays the sales per product purchased by the user, with the total sales calculated from the order details.

4. Product Order Details Tab

- **Product Order Details**: Provides an overview of the product's order history, including the number of times it was ordered and details about the customers who ordered it. This helps to track the popularity and customer base of each product.

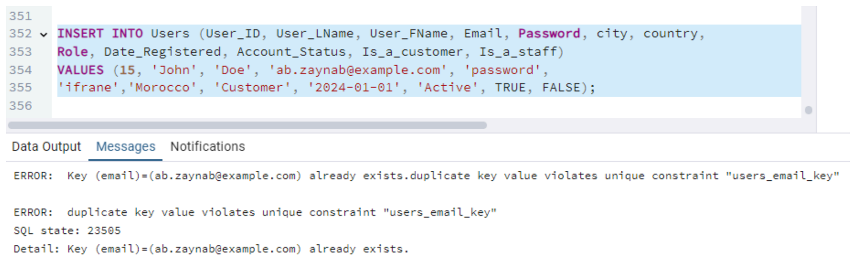
7. Testing and fine-tuning

Unit Testing:

Unit testing focuses on verifying that each individual component of your database schema works correctly in isolation. This includes checking the integrity of constraints, such as uniqueness, non-null constraints, and data types, as well as foreign key relationships.

Actions:

1. **Uniqueness Test**:
    - **Objective**: Verify that columns marked as UNIQUE, such as Email in the Users table, correctly reject duplicate entries.
    - **Query**:



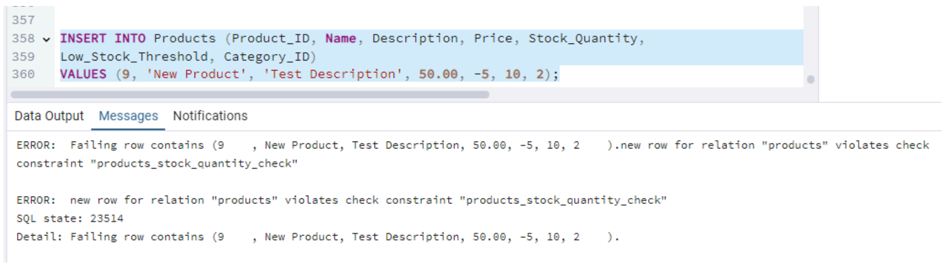
```
INSERT INTO Users (User_ID, User_LName, User_FName, Email, Password, city, country,
Role, Date_Registered, Account_Status, Is_a_customer, Is_a_staff)
VALUES (15, 'John', 'Doe', 'ab.zaynab@example.com', 'password',
'ifrane','Morocco', 'Customer', '2024-01-01', 'Active', TRUE, FALSE);
```

```
ERROR:  Key (email)=(ab.zaynab@example.com) already exists.duplicate key value violates unique constraint "users_email_key"

ERROR:  duplicate key value violates unique constraint "users_email_key"
SQL state: 23505
Detail: Key (email)=(ab.zaynab@example.com) already exists.
```

Failed due to duplicate email.

Non-Negative Constraints Test:

- **Objective**: Verify that fields like Stock_Quantity in the Products table only accept valid, non-negative numbers.
- **Query**:

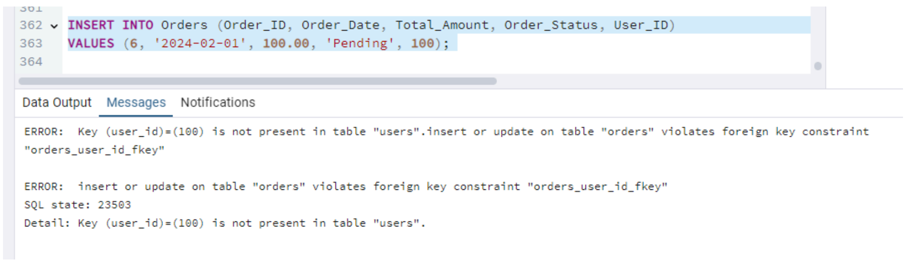


```sql
INSERT INTO Products (Product_ID, Name, Description, Price, Stock_Quantity,
Low_Stock_Threshold, Category_ID)
VALUES (9, 'New Product', 'Test Description', 50.00, -5, 10, 2);
```

```
ERROR:  Failing row contains (9    , New Product, Test Description, 50.00, -5, 10, 2    ).new row for relation "products" violates check
constraint "products_stock_quantity_check"

ERROR:  new row for relation "products" violates check constraint "products_stock_quantity_check"
SQL state: 23514
Detail: Failing row contains (9    , New Product, Test Description, 50.00, -5, 10, 2    ).
```

Foreign Key Integrity Test:

- **Objective**: Ensure that Orders and OrderDetail tables maintain their relationships with the Users and Product tables, respectively.
- **Query**:



```sql
INSERT INTO Orders (Order_ID, Order_Date, Total_Amount, Order_Status, User_ID)
VALUES (6, '2024-02-01', 100.00, 'Pending', 100);
```

```
ERROR:  Key (user_id)=(100) is not present in table "users".insert or update on table "orders" violates foreign key constraint
"orders_user_id_fkey"

ERROR:  insert or update on table "orders" violates foreign key constraint "orders_user_id_fkey"
SQL state: 23503
Detail: Key (user_id)=(100) is not present in table "users".
```

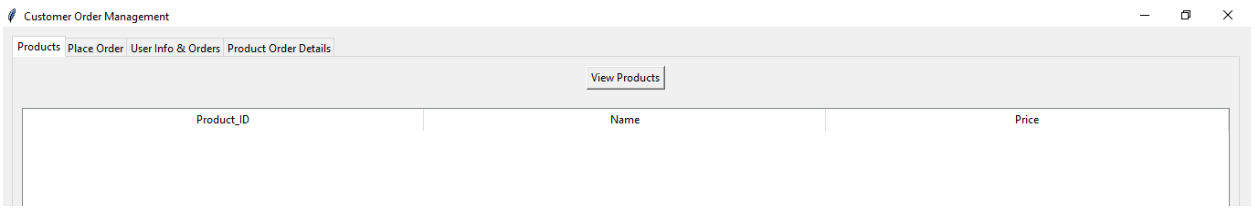
- ## UI Testing Application:

1. Application Launch

Test Objective: Verify that the application window launches successfully.

Expected Result:

- The application window should open with the title "Customer Order Management".
- The main window should contain a tabbed interface with the following tabs:
    - "Products"
    - "Place Order"
    - "User Info & Orders"
    - "Product Order Details"



Product_ID	Name	Price

2. Products Tab

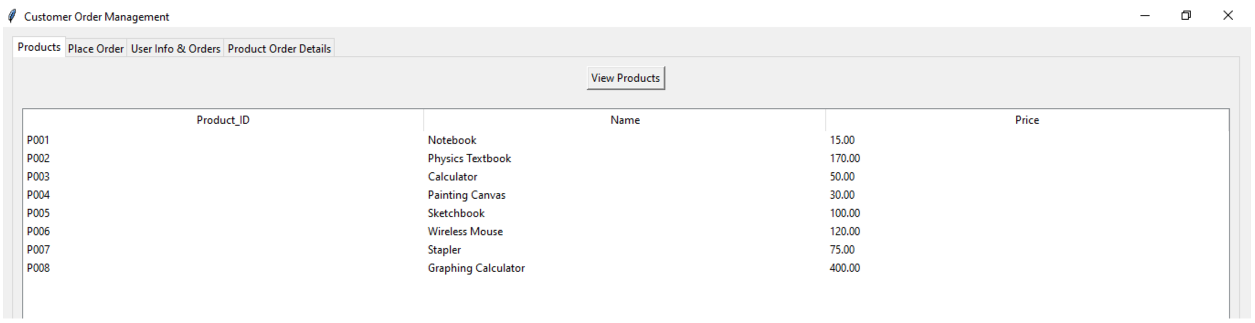
Test Objective: Verify that products are displayed in the treeview when "View Products" is clicked.

Steps:

1. Click the "View Products" button.

Expected Result:

- The treeview should display product details including Product_ID, Name, and Price under respective columns.



Product_ID	Name	Price
P001	Notebook	15.00
P002	Physics Textbook	170.00
P003	Calculator	50.00
P004	Painting Canvas	30.00
P005	Sketchbook	100.00
P006	Wireless Mouse	120.00
P007	Stapler	75.00
P008	Graphing Calculator	400.00

3. Place Order Tab

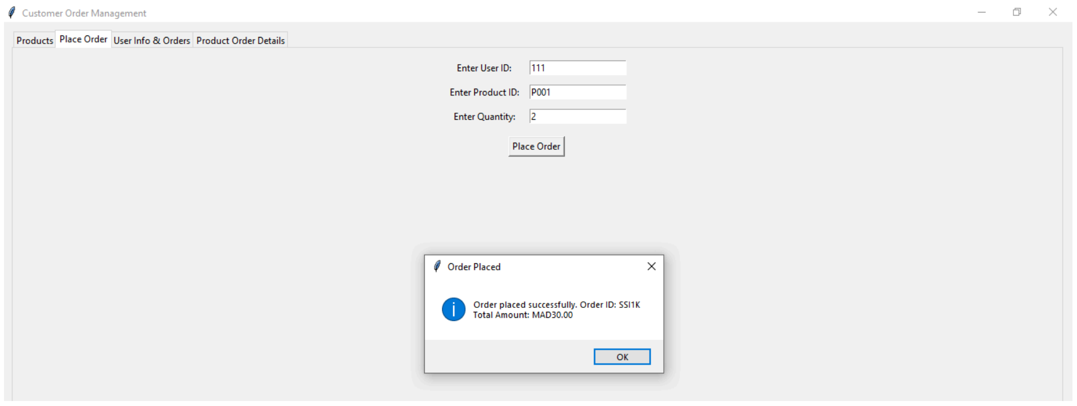
Test Objective: Verify that the order can be placed with valid inputs, and error messages are shown for invalid inputs.

Steps:

1. Enter a valid User ID, Product ID, and Quantity and click "Place Order".

Expected Result:

- A confirmation message box should display the Order ID and Total Amount.
- The order should be successfully placed in the database.



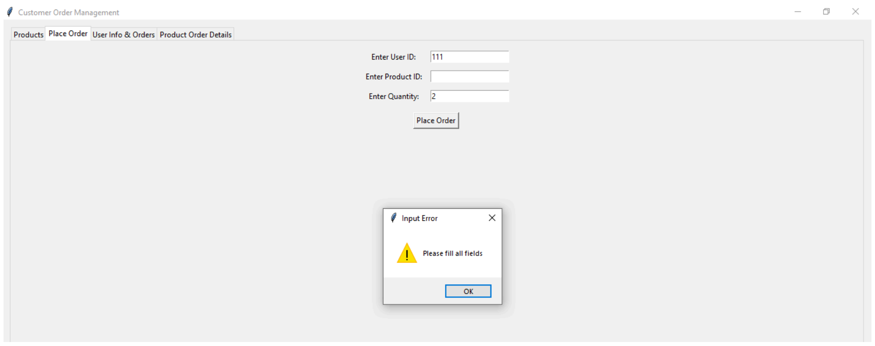
Enter User ID: 111
Enter Product ID: P001
Enter Quantity: 2

Order Placed

Order placed successfully. Order ID: SSI1K
Total Amount: MAD30.00

Edge Cases:

- If any field is empty, display a warning message: "Please fill all fields".

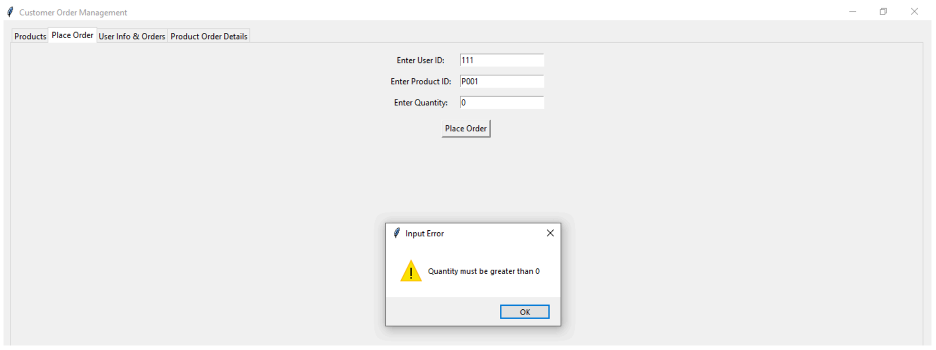


Enter User ID: 111
Enter Product ID:
Enter Quantity: 2

Input Error

Please fill all fields

- If quantity is less than or equal to 0, display a warning: "Quantity must be greater than 0".

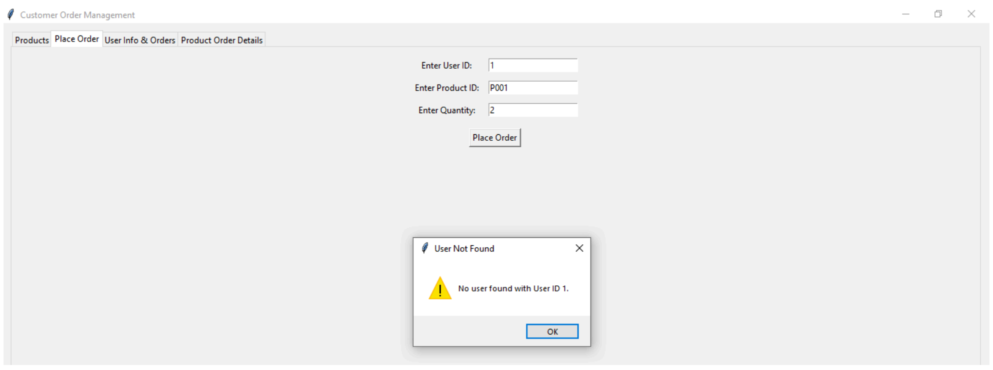


Enter User ID: 111
Enter Product ID: P001
Enter Quantity: 0

Input Error

Quantity must be greater than 0

- If the user or product does not exist, display a warning: "User not found" or "Product not found".


Customer Order Management

Products | Place Order | User Info & Orders | Product Order Details

Enter User ID: 1
Enter Product ID: P001
Enter Quantity: 2

Place Order

User Not Found

No user found with User ID 1.

OK

4. User Info & Orders Tab

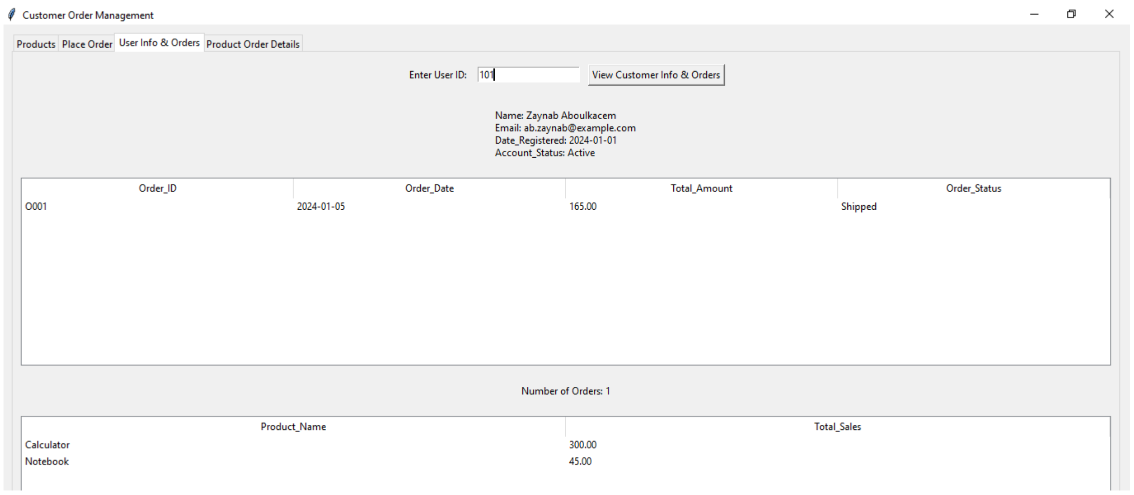
Test Objective: Verify that the user information and order history are displayed correctly.

Steps:

1. Enter a valid User ID and click "View Customer Info & Orders".

Expected Result:

- User information (Name, Email, Date Registered, Account Status) is displayed.
- Orders associated with the user are displayed in the treeview, including Order_ID, Order_Date, Total_Amount, and Order_Status.


Customer Order Management

Products | Place Order | User Info & Orders | Product Order Details

Enter User ID: 101 | View Customer Info & Orders

Name: Zaynab Aboulkacem
Email: ab.zaynab@example.com
Date_Registered: 2024-01-01
Account_Status: Active

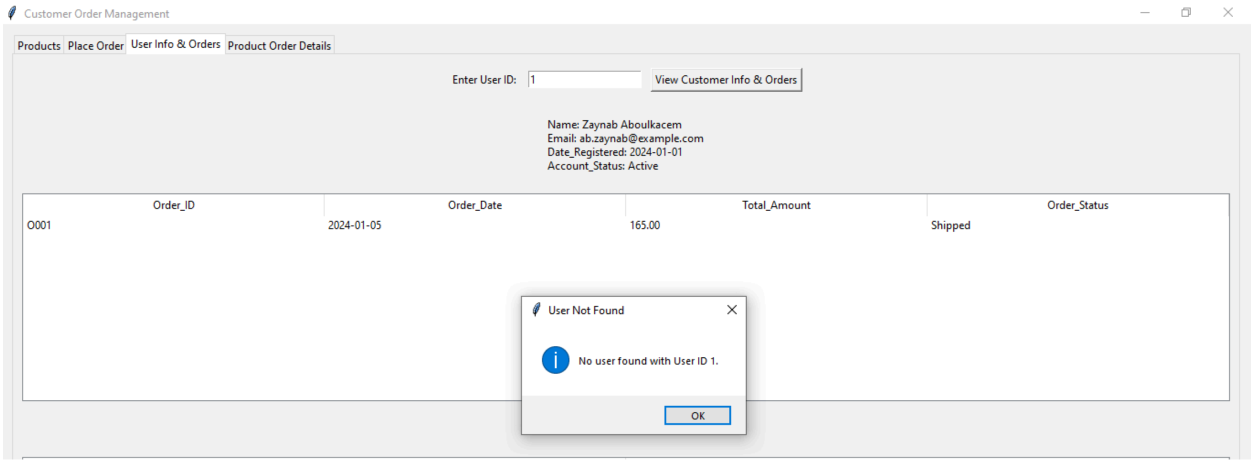
Order_ID	Order_Date	Total_Amount	Order_Status
O001	2024-01-05	165.00	Shipped

Number of Orders: 1

Product_Name	Total_Sales
Calculator	300.00
Notebook	45.00

Edge Cases:

- If no orders are found for the user, show a message: "No orders found for User ID ".


Customer Order Management

Products | Place Order | User Info & Orders | Product Order Details

Enter User ID: 1 | View Customer Info & Orders

Name: Zaynab Aboulkacem
Email: ab.zaynab@example.com
Date_Registered: 2024-01-01
Account_Status: Active

Order_ID	Order_Date	Total_Amount	Order_Status
O001	2024-01-05	165.00	Shipped

User Not Found

No user found with User ID 1.

OK

5. Product Order Details Tab

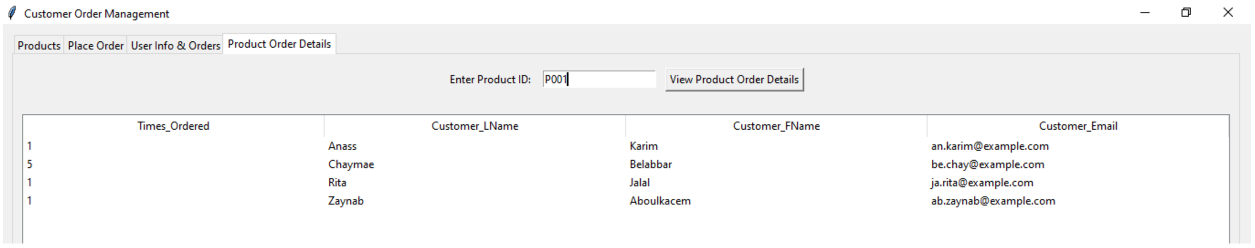
Test Objective: Verify that product order details (how many times ordered and customers who ordered it) are displayed correctly.

Steps:

1. Enter a valid Product ID and click "View Product Order Details".

Expected Result:

- The treeview should display the following columns:
  - Times_Ordered
  - Customer_LName
  - Customer_FName
  - Customer_Email


Customer Order Management

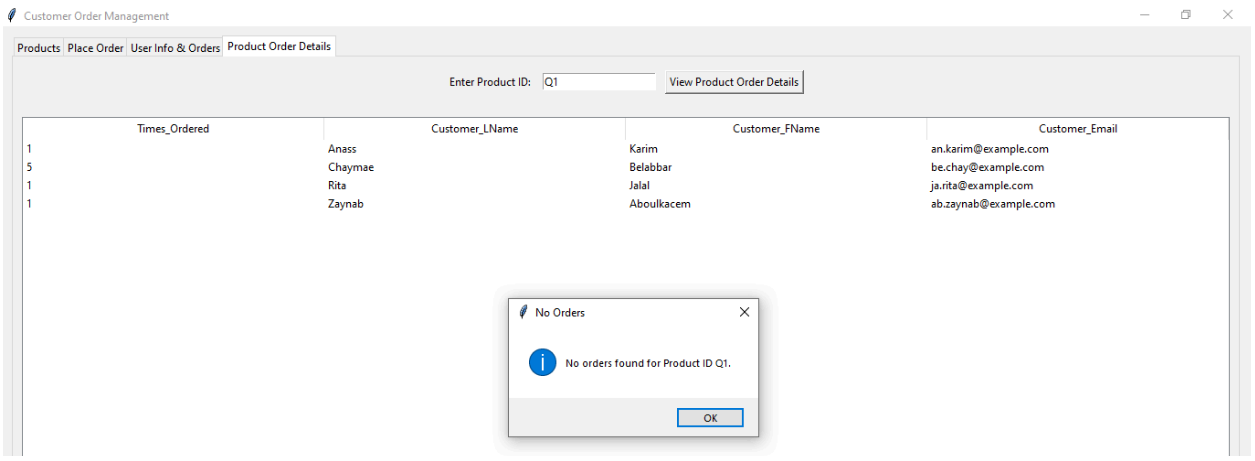
Products | Place Order | User Info & Orders | Product Order Details

Enter Product ID: P001 | View Product Order Details

Times_Ordered	Customer_LName	Customer_FName	Customer_Email
1	Anass	Karim	an.karim@example.com
5	Chaymae	Belabbar	be.chay@example.com
1	Rita	Jalal	ja.rita@example.com
1	Zaynab	Aboulkacem	ab.zaynab@example.com

Edge Cases:

- If no orders are found for the product, show a message: "No orders found for Product ID".



Times_Ordered	Customer_LName	Customer_FName	Customer_Email
1	Anass	Karim	an.karim@example.com
5	Chaymae	Belabbar	be.chay@example.com
1	Rita	Jalal	ja.rita@example.com
1	Zaynab	Aboulkacem	ab.zaynab@example.com

No orders found for Product ID Q1.

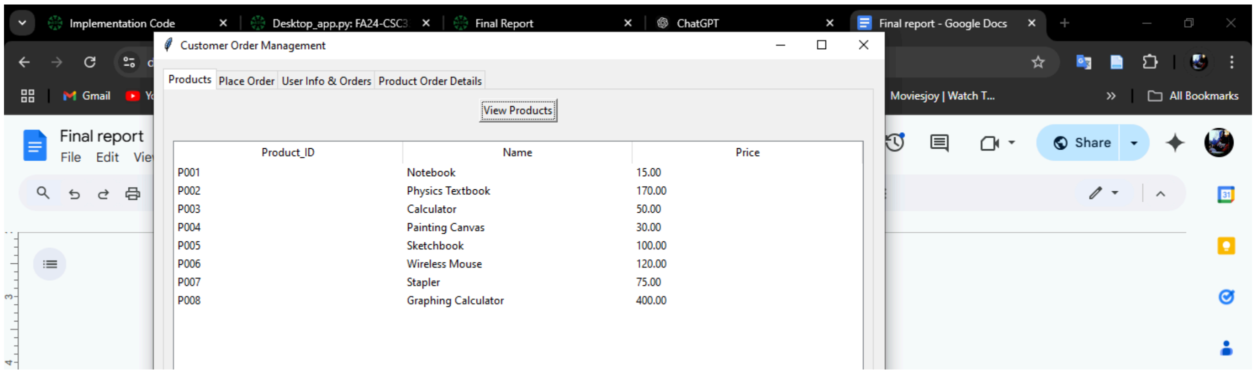
- **UI Fine-Tuning Suggestions**

1. Responsive Layout

Test Objective: Verify that the UI adjusts to different window sizes.

Test Result:

The application window maintains a well-organized layout at all sizes. All elements, including buttons, labels, and treeviews, adjust gracefully with resizing.



Product_ID	Name	Price
P001	Notebook	15.00
P002	Physics Textbook	170.00
P003	Calculator	50.00
P004	Painting Canvas	30.00
P005	Sketchbook	100.00
P006	Wireless Mouse	120.00
P007	Stapler	75.00
P008	Graphing Calculator	400.00

Minimized Version

2. Clear Labels and Instructions

Test Objective: Ensure that labels for input fields are clear and easily understandable.

Test Result:

- All input field labels are concise, clear, and easy to understand.

3. Error Message Clarity

Test Objective: Verify that error messages are clear and informative.

Test Result:

- All error messages are easy to understand and informative, guiding the user on how to resolve issues.

Overall Fine-Tuning Conclusion:

- After reviewing all aspects of the interface, we are satisfied with its current design and functionality. No further adjustments are necessary at this time.

8. User Manual

The **Customer Order Management Application** is a desktop application developed using Tkinter and connected to a PostgreSQL database.

System Requirements

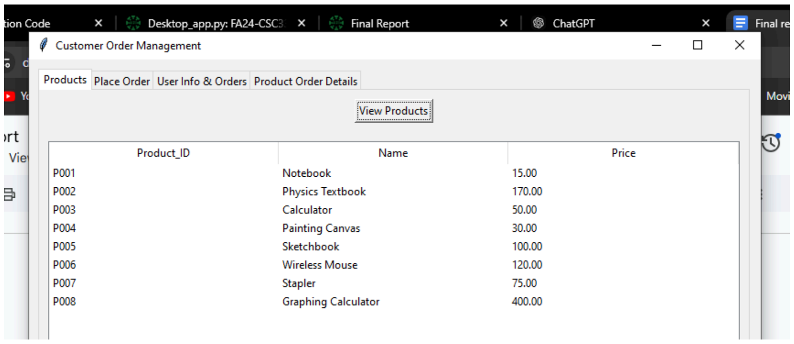
- Python 3.x
- Tkinter (for GUI)
- psycopg2 (for PostgreSQL database connection)
- PostgreSQL database with the following tables:
    - **Products** (Product_ID, Name, Price)
    - **Users** (User_ID, User_LName, User_FName, Email, Date_Registered, Account_Status)
    - **Orders** (Order_ID, User_ID, Order_Date, Order_Status, Total_Amount)
    - **OrderDetail** (Order_ID, Product_ID, Quantity, Unit_Price)
    -

Application Interface

1. Products Tab

Purpose: View all available products with their details (Product ID, Name, Price).

- **Button**: View Products
    - Click the button to fetch and display all products from the database in a table format.
- **Treeview**: Displays product details in a tabular form with the following columns:
    - **Product_ID**
    - **Name**
    - **Price**

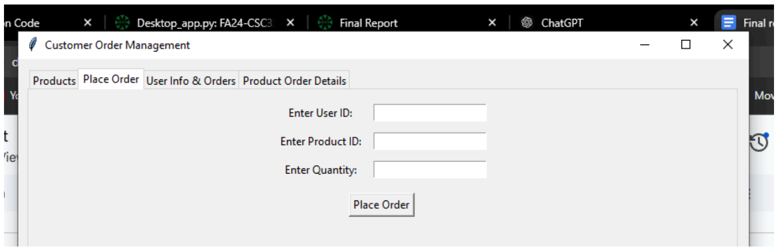


Product_ID	Name	Price
P001	Notebook	15.00
P002	Physics Textbook	170.00
P003	Calculator	50.00
P004	Painting Canvas	30.00
P005	Sketchbook	100.00
P006	Wireless Mouse	120.00
P007	Stapler	75.00
P008	Graphing Calculator	400.00

2. Place Order Tab

Purpose: Place an order by selecting a user, product, and quantity.

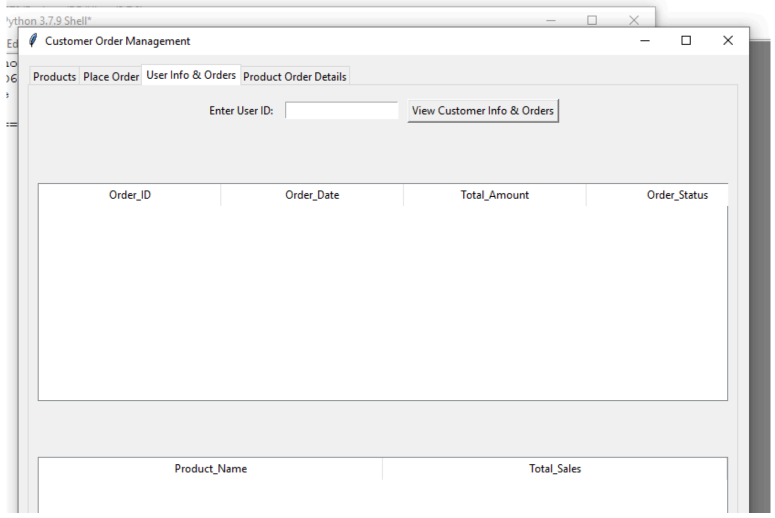
- **Input Fields**:
    - **User ID**: Enter the ID of the user placing the order.
    - **Product ID**: Enter the ID of the product being ordered.
    - **Quantity**: Enter the number of products being ordered.
- **Button**: Place Order
    - When clicked, the system checks the validity of the input fields and places an order in the **Orders** and **OrderDetail** tables.
    - If successful, a confirmation message is shown with the **Order ID** and **Total Amount**.



3. User Info & Orders Tab

Purpose: View user profile details and the list of orders placed by the user.

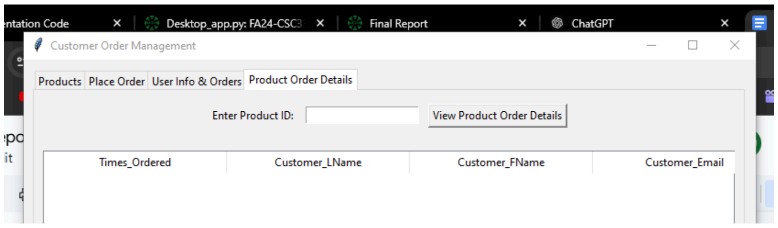
- **Input Field**:
  - **User ID**: Enter the User ID to fetch user information and order details.
- **Button**: View Customer Info & Orders
  - Fetches and displays the following:
    - **User Information**: Name, Email, Registration Date, Account Status.
    - **Order Details**: A list of orders placed by the user with **Order ID**, **Order Date**, **Total Amount**, and **Order Status**.
    - **Order Count**: Displays the number of orders placed by the user.
    - **Sales by Product**: Displays the total sales of each product ordered by the user.



4. Product Order Details Tab

Purpose: View the details of a product's order history.

- **Input Field**:
  - **Product ID**: Enter the Product ID to fetch its order history.
- **Button**: View Product Order Details
  - Fetches and displays:
    - **Times Ordered**: Number of times the product has been ordered.
    - **Customer Details**: Displays the **Customer Name**, **Email** who ordered this product.
    -



Error Handling

1. **Database Connection Error**: If the application is unable to connect to the PostgreSQL database, a message box will show an error. Check the database connection parameters (host, username, password, database name).

2. **Input Errors**: If a required field is empty, or if the quantity is not a positive integer, the application will display a warning message.

3. **User/Order/Product Not Found**: If the entered **User ID** or **Product ID** does not exist, the application will show a message indicating that no such record was found.

9. CONCLUSION:

In conclusion, the project successfully developed and put into operation a complete database system for managing orders, items, and customer interactions. Primary features such as updating product stock after processing an order and calculating the total order amounts, and calculating any customer discounts based on their previous purchases were incorporated into the system. Furthermore, the use of triggers enabled automatic operations such as changing the order status and tracking the changes in those statuses.

The database system was tested using a variety of test scenarios to ensure that each stored function, process and trigger worked as we planned. We had to check whether the stock quantity was properly modified after orders were submitted, if the total of orders were computed precisely, and if the customer discounts were applied correctly. Adding to that , we had to check if the triggers guaranteed that order statuses were updated in real time and that the status changes were properly recorded.

Overall, in this project, we successfully digitised Ifrane marché's library and created an efficient and user-friendly website. The portal allows users (students and staff) to explore available items, make orders, and follow their statuses, improving the entire buying experience. This change to a digital system not only simplifies operations, but it also gives a current solution to suit the AUI community's increasing needs.

10. Future Work

We would like to add some proposed future work such as:

- **Real-time Stock Monitoring:** Implement real-time stock tracking to monitor product availability as orders are placed. This can include notifications when stock levels are running low, enabling automatic reordering or alerts to the purchasing team.
- **Order Management for Multiple Payment Methods:** Expand the system to handle multiple payment methods, including credit cards, bank transfers, and digital wallets. This would offer flexibility to customers and simplify order processing for businesses.