CSE 2211 - Database Management System Lab

Project Name: Shop Management System

Submitted To

Abu Ahmed Ferdaus

Associate Professor

Dr. Md Mamun-Or-Rashid

Professor

Department of Computer Science & Engineering

University of Dhaka

Submitted By

Md. Tauseef - Ur - Rahman (Roll 24)

Faiaz Mahmud (Roll 46)

28th Batch

Submitted On

9th March 2025

Table of Contents

1. Brief Introduction of our Database Project

The **Shop Management System** is a database-driven application designed to manage retail operations using structured queries efficiently. The system relies on **SQL queries** to handle core functionalities such as user authentication, product management, order processing, and billing. Queries are used to **insert, update, delete, and retrieve** data, ensuring seamless transactions and real-time data updates. Complex **JOIN queries** enable relationships between tables, such as fetching order details along with customer information, while **aggregate functions** calculate total sales and inventory levels and so on. Optimized indexing and constraints ensure data integrity, improving performance and consistency across the system.

2. Schemas with Attributes

There are in total 8 schemas/tables in our database project. Among these, **three** are independent tables namely **users**, **products,** and **workers**. All of the schemas with their attributes are given below :

1. Users (users)

Stores user details.

- user_id (INT, PK) – Unique ID.
- user_name (VARCHAR(50)) – Name.
- user_email (VARCHAR(50)) – Unique Email.
- user_password (VARCHAR(255)) – Password.
- user_contact_no (VARCHAR(20)) – Contact number.
- is_admin (BOOLEAN) – Admin status.

2. Products (product)

Stores product details.

- prod_id (INT, PK) – Unique product ID.
- prod_name (VARCHAR(255)) – Name.
- prod_image (VARCHAR(255)) – Image URL.
- prod_quantity (INT) – Stock quantity.
- prod_price (DECIMAL(10,2)) – Price.
- rating_stars (INT) – Rating.
- prod_discount (DECIMAL(5,2)) – Discount.

3. Workers (worker)

Stores employee details.

- worker_id (INT, PK) – Unique ID.
- worker_name (VARCHAR(50)) – Name.
- worker_email (VARCHAR(50)) – Unique Email.
- worker_contact_no (VARCHAR(20)) – Contact number.
- worker_salary (DOUBLE) – Salary.

4. Shopping Cart (shopping_cart)

Tracks items in a cart.

- cart_id (INT, PK) – Unique cart ID.
- prod_id (INT, FK) – Product ID.
- user_id (INT, FK) – User ID.

5. Orders (orders)

Stores order information.

- order_id (INT, PK) – Unique order ID.
- order_date (DATE) – Date.
- user_id (INT, FK) – User who ordered.
- delivery_address (VARCHAR(50)) – Delivery address.
- total_amt (DECIMAL(10,2)) – Order total.
- order_status (VARCHAR(20)) – Status.

6. Order Details (order_detail)

Stores products in an order.

- order_id (INT, FK) – Order ID.
- prod_id (INT, FK) – Product ID.
- prod_qty (INT) – Quantity.
- prod_price (DOUBLE) – Price.
- prod_total_price (DOUBLE) – Total price.

7. Billing (bill_detail)

Stores billing info.

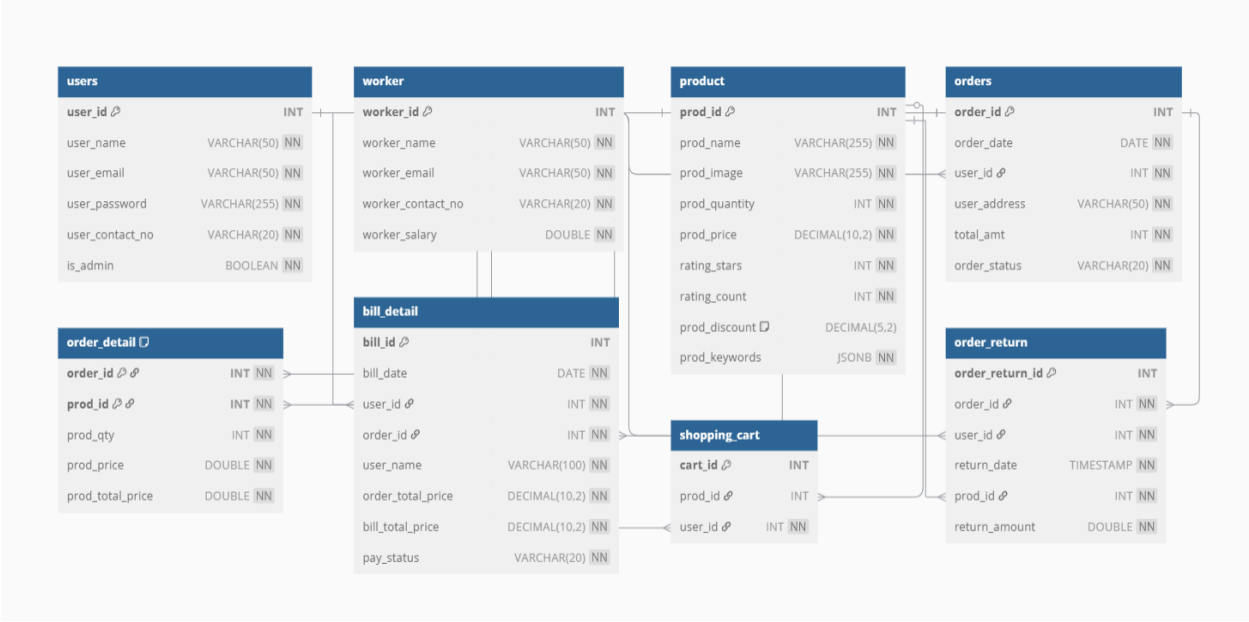
- bill_id (INT, PK) – Unique bill ID.
- bill_date (DATE) – Date.
- user_id (INT, FK) – User ID.
- order_id (INT, FK) – Order ID.
- order_total_price (DECIMAL(10,2)) – Order total.
- bill_total_price (DECIMAL(10,2)) – Final bill amount (after adding discounts).
- pay_status (VARCHAR(20)) – Payment status.

8. Order Returns (order_return)

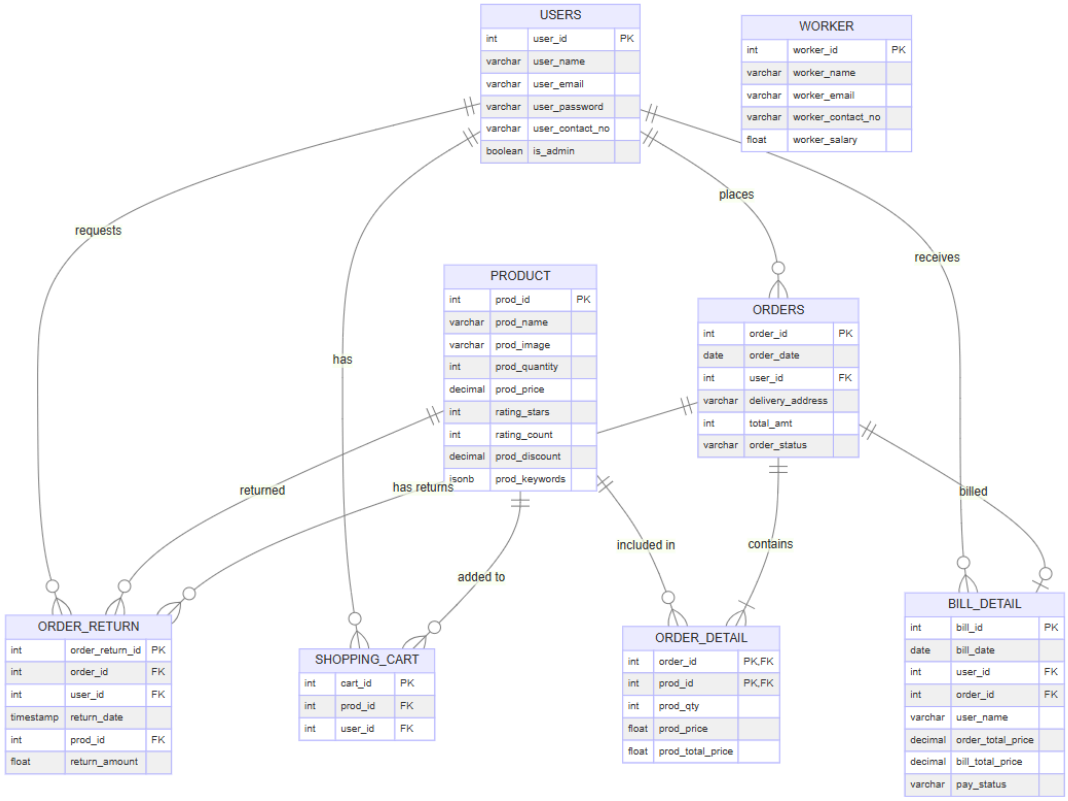
Tracks returned orders.

- order_return_id (INT, PK) – Unique return ID.
- order_id (INT, FK) – Order ID.
- user_id (INT, FK) – User ID.
- return_date (TIMESTAMP) – Return date.
- prod_id (INT, FK) – Product ID.
- return_amount (DOUBLE) – Refund amount.

3. Schema Diagram of the database



4. E-R Diagram of the database



5. Snapshots of SQL DDL of all the schemas/tables

Users Table

```sql
CREATE TABLE IF NOT EXISTS users (
    user_id SERIAL PRIMARY KEY,
    user_name VARCHAR(50) NOT NULL,
    user_email VARCHAR(50) UNIQUE NOT NULL,
    user_password VARCHAR(255) NOT NULL,
    user_contact_no VARCHAR(20) NOT NULL,
    is_admin BOOLEAN NOT NULL
);
```

Product Table

```sql
CREATE TABLE IF NOT EXISTS product (
    prod_id SERIAL PRIMARY KEY,
    prod_name VARCHAR(255) UNIQUE NOT NULL,
    prod_image VARCHAR(255) NOT NULL,
    prod_quantity INTEGER NOT NULL,
    prod_price DECIMAL(10,2) NOT NULL,
    rating_stars INTEGER NOT NULL,
    rating_count INTEGER NOT NULL,
    prod_discount DECIMAL(5,2) DEFAULT 0.0,
    prod_keywords JSONB NOT NULL
);
```

Order_detail Table

```
CREATE TABLE IF NOT EXISTS order_detail (
    order_id INT REFERENCES orders(order_id) NOT NULL,
    prod_id INT REFERENCES product(prod_id) NOT NULL,
    prod_qty INT NOT NULL,
    prod_price DOUBLE PRECISION NOT NULL,
    prod_total_price DOUBLE PRECISION NOT NULL,
    PRIMARY KEY (order_id, prod_id)
);
```

Shopping_cart Table

```
CREATE TABLE IF NOT EXISTS shopping_cart (
    cart_id SERIAL,
    prod_id INT NOT NULL,
    user_id INT NOT NULL,
    PRIMARY KEY (prod_id, user_id),
    FOREIGN KEY (prod_id) REFERENCES product(prod_id),
    CONSTRAINT shopping_cart_user_id_fkey FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE
);
```

Worker Table

```
CREATE TABLE IF NOT EXISTS worker (
    worker_id SERIAL PRIMARY KEY,
    worker_name VARCHAR(50) NOT NULL,
    worker_email VARCHAR(50) NOT NULL,
    worker_contact_no VARCHAR(20) NOT NULL,
    worker_salary DOUBLE PRECISION NOT NULL
);
```

Bill_detail Table

```sql
CREATE TABLE IF NOT EXISTS bill_detail (
    bill_id SERIAL PRIMARY KEY, -- Automatically incrementing bill ID
    bill_date DATE NOT NULL, -- Date of the bill (set to current date)
    user_id INT NOT NULL, -- Foreign key to users table
    order_id INT NOT NULL, -- Foreign key to orders table
    user_name VARCHAR(100) NOT NULL, -- Name of the user
    order_total_price DECIMAL(10, 2) NOT NULL, -- Total price of the order
    bill_total_price DECIMAL(10, 2) NOT NULL, -- Total price of the bill (after any adjustments)
    pay_status VARCHAR(20) CHECK (pay_status IN ('paid', 'unpaid')) NOT NULL -- Payment status of the bill
    CONSTRAINT bill_detail_user_id_fkey FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE,
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
```

Orders Table

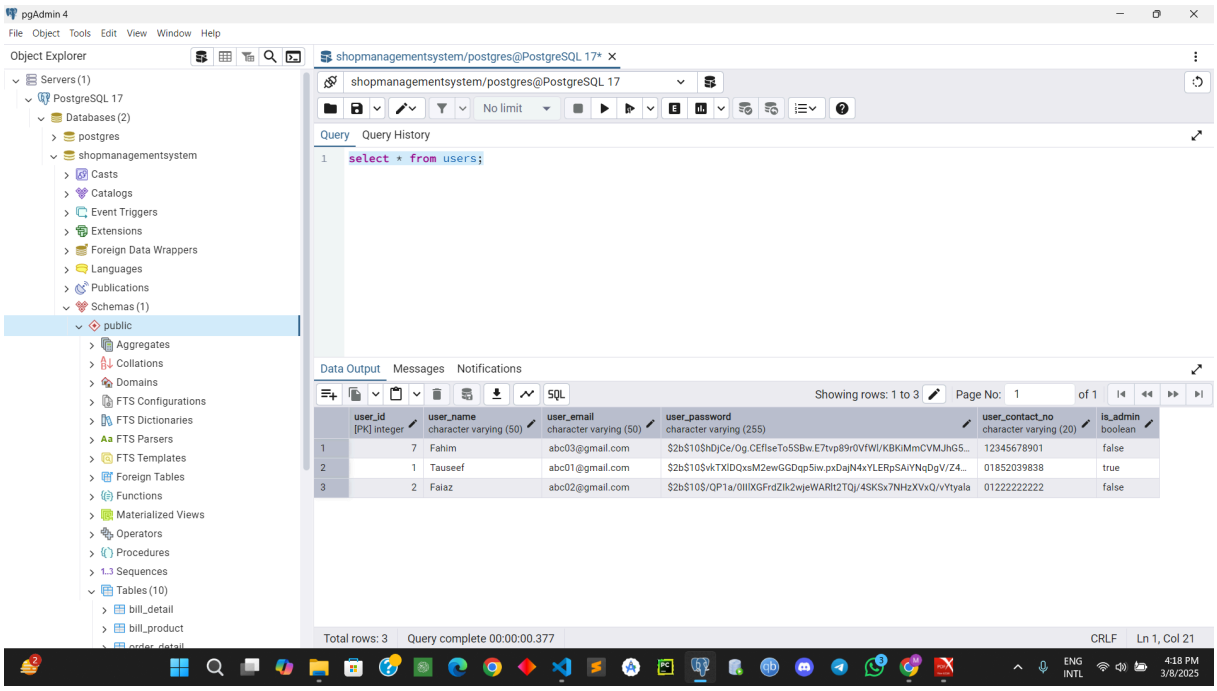
```sql
CREATE TABLE IF NOT EXISTS orders (
    order_id SERIAL PRIMARY KEY,
    order_date DATE NOT NULL,
    user_id INT NOT NULL,
    delivery_address VARCHAR(50) NOT NULL,
    total_amt INT NOT NULL,
    order_status VARCHAR(20) CHECK (order_status IN ('delivered', 'in process')) NOT NULL,
    CONSTRAINT orders_user_id_fkey FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE
);
```

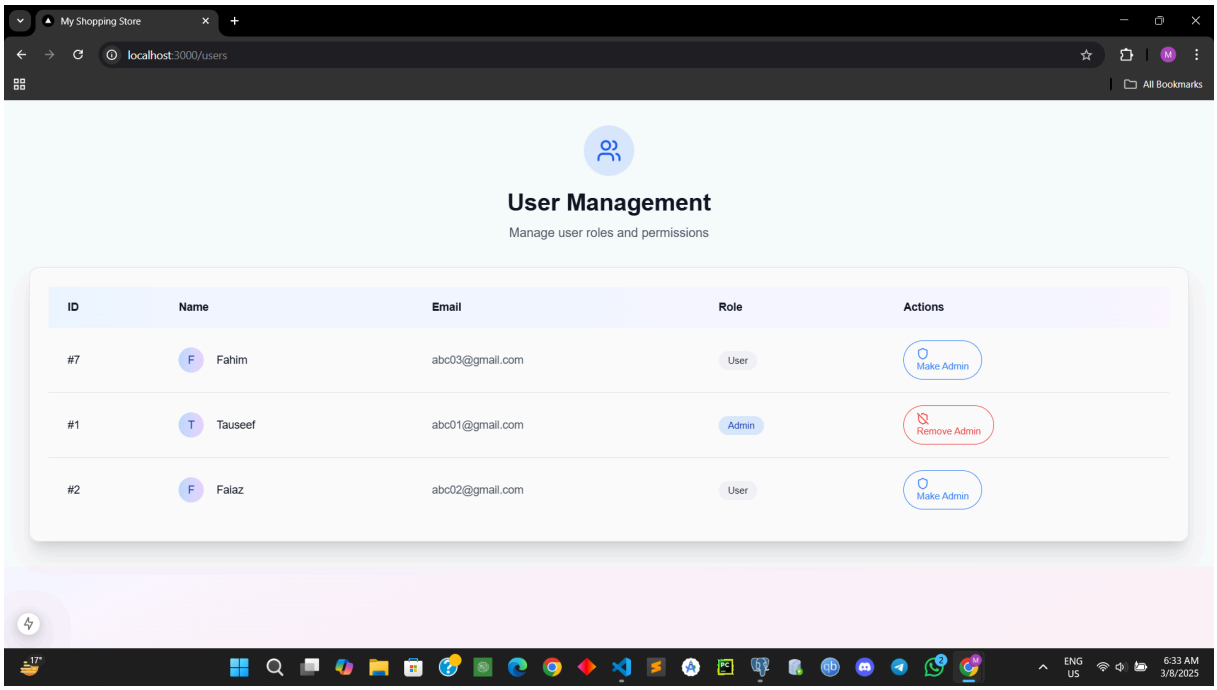
Order_return Table

```sql
CREATE TABLE IF NOT EXISTS order_return (
    order_return_id SERIAL PRIMARY KEY,
    order_id INT REFERENCES orders(order_id) NOT NULL,
    user_id INT REFERENCES users(user_id) NOT NULL,
    return_date TIMESTAMP NOT NULL,
    prod_id INT REFERENCES product(prod_id) NOT NULL,
    return_amount DOUBLE PRECISION NOT NULL
);
```

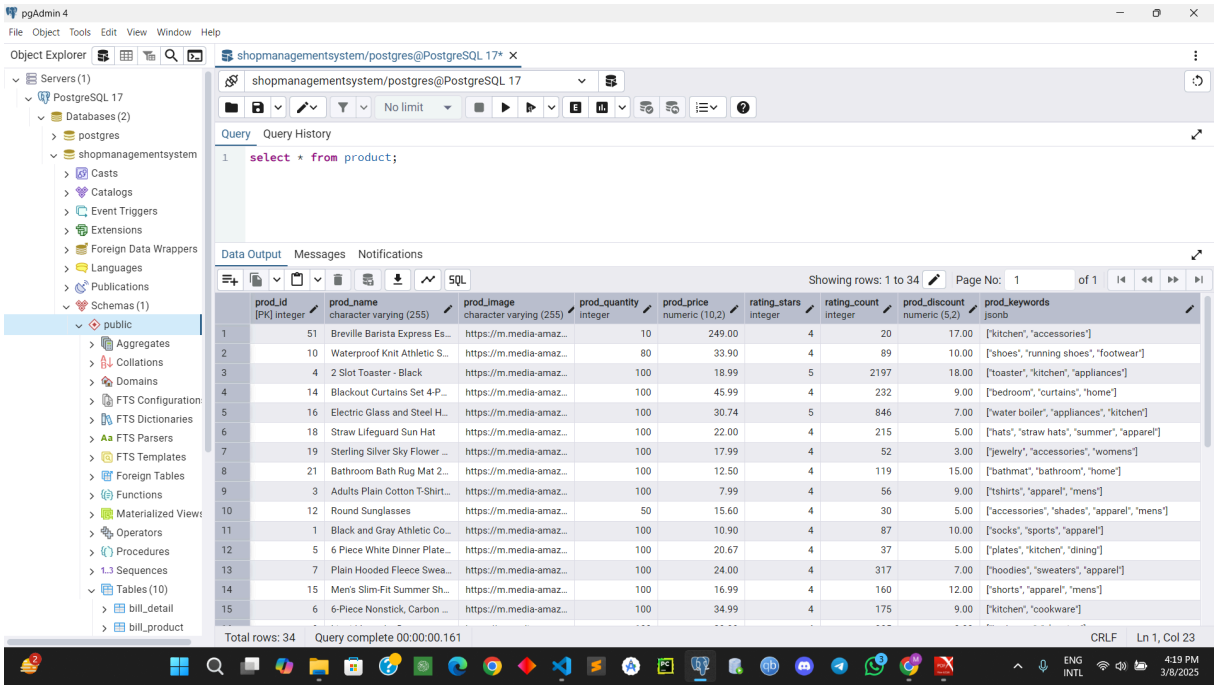
6. Snapshots of the instances (data of the populated tables)

Users Page

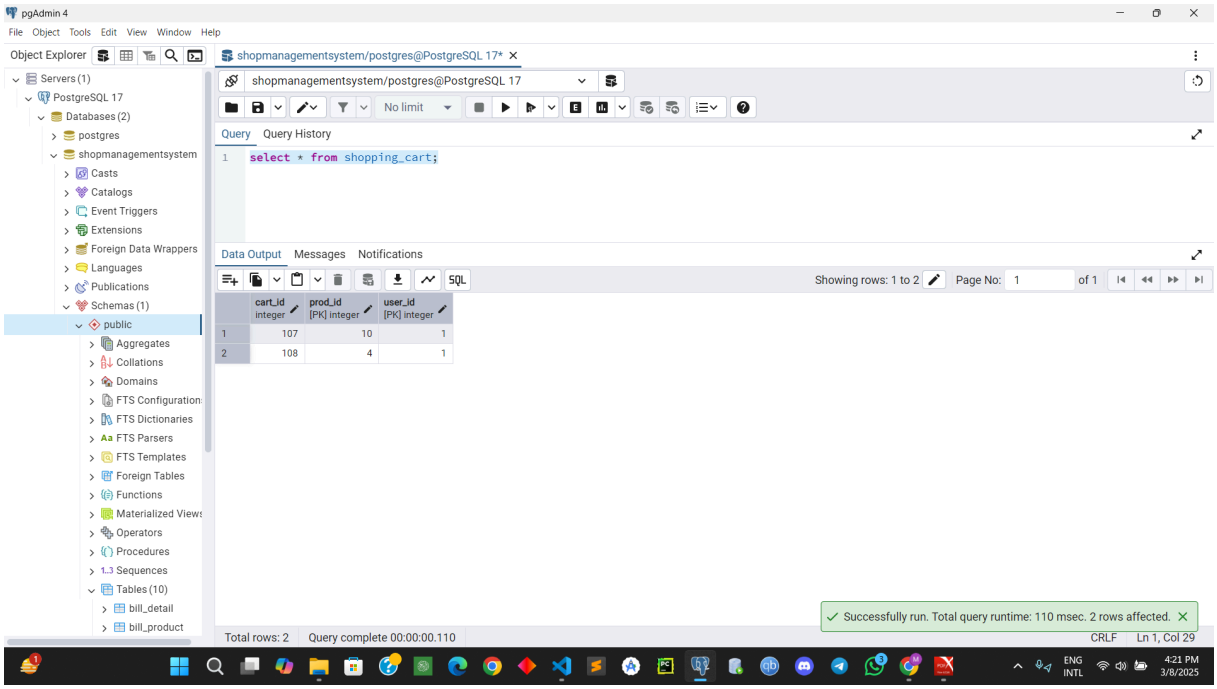




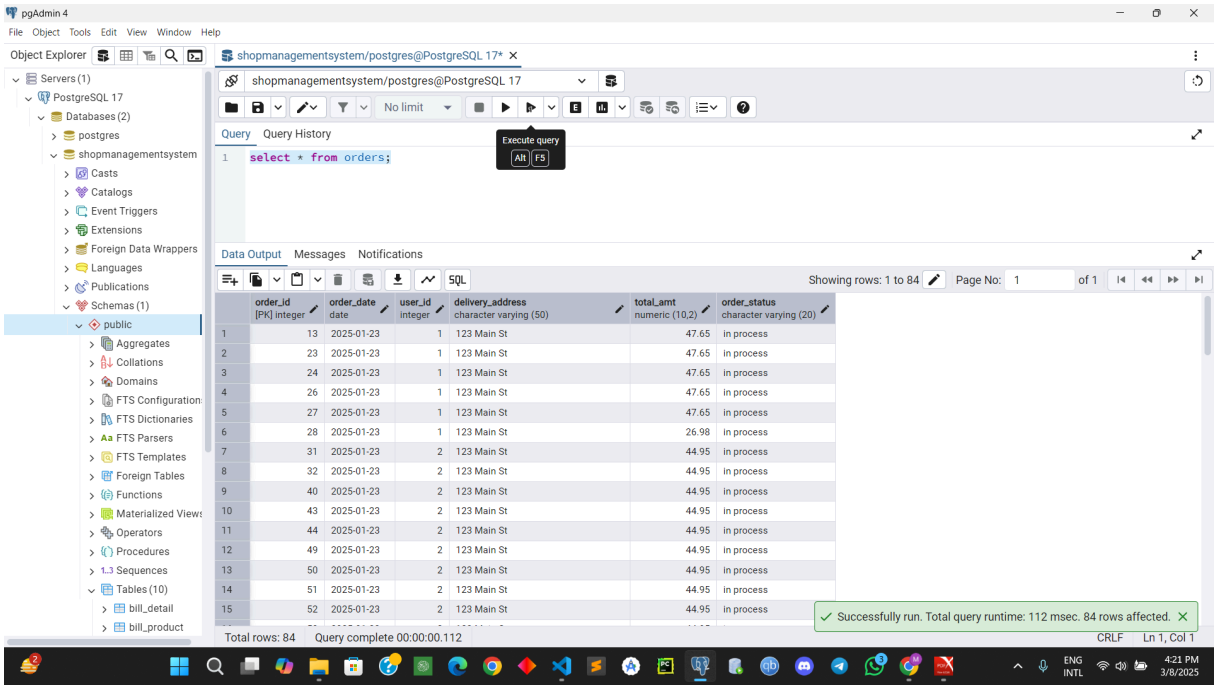
Product Page

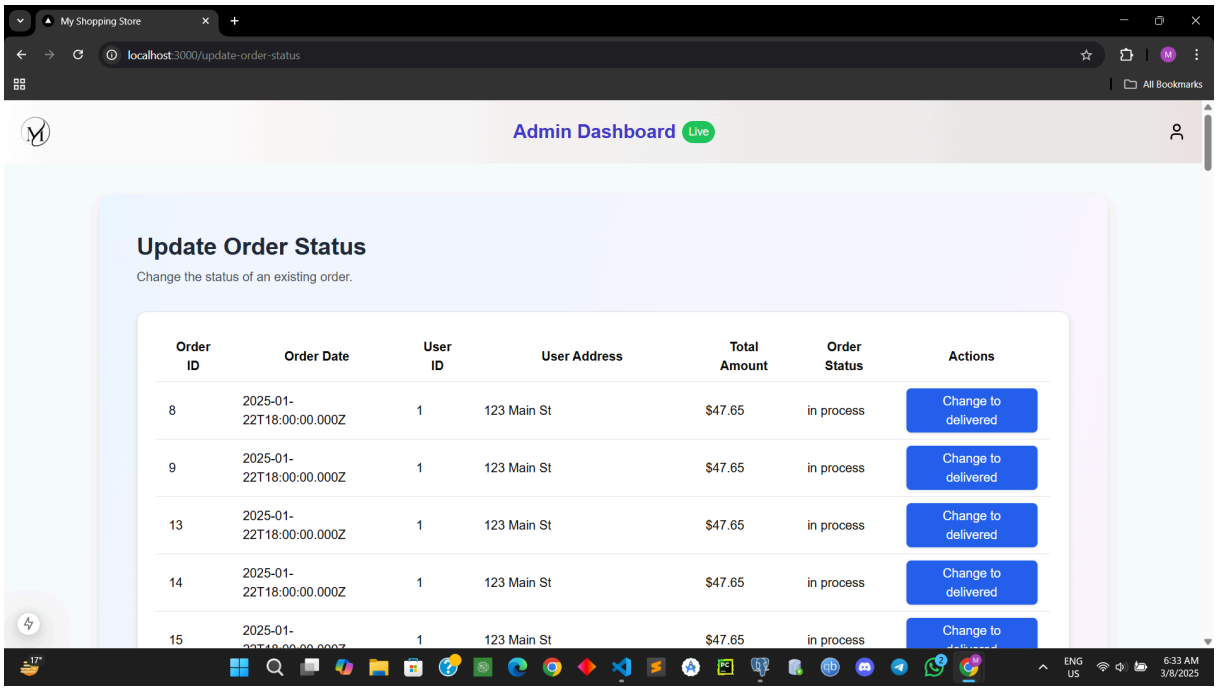


Shopping_cart Page

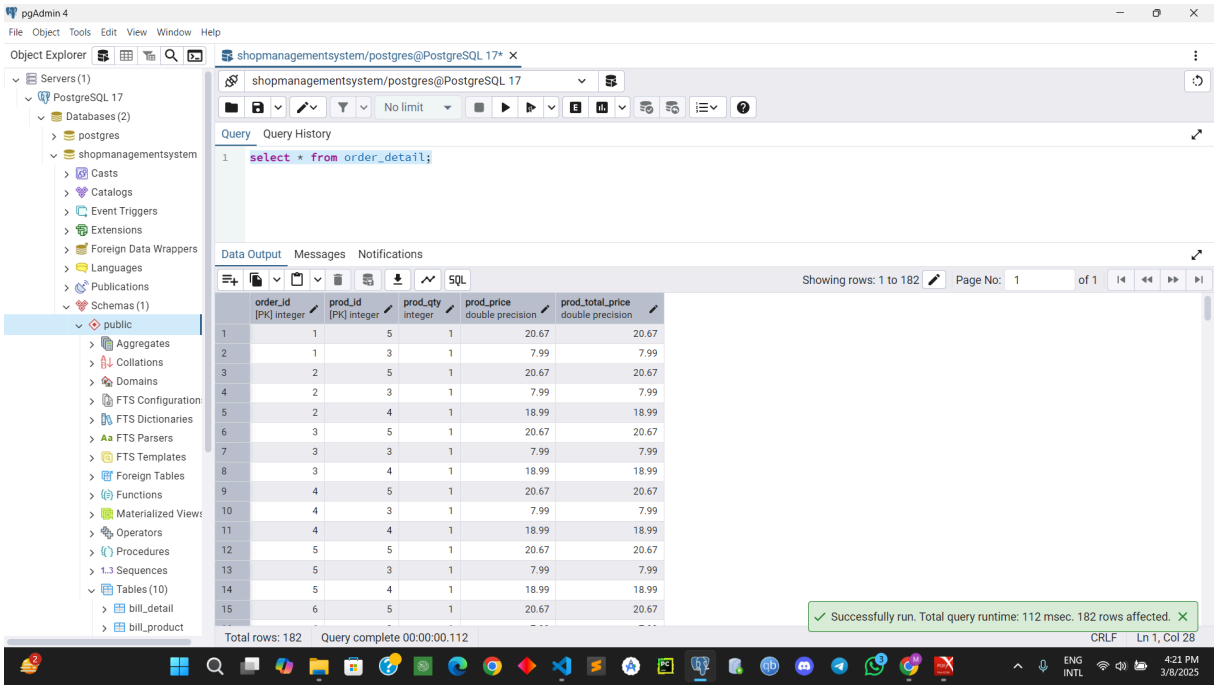


Orders Page

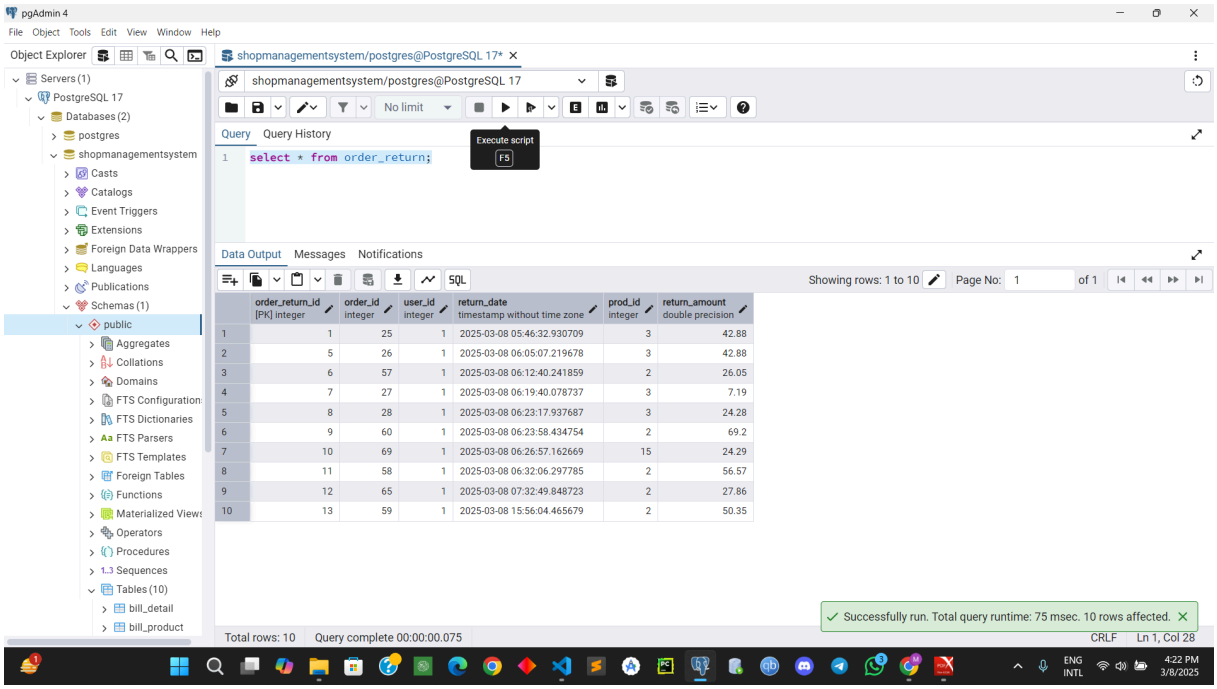

pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
shopmanagementsystem/postgres@PostgreSQL 17*
Servers (1)
PostgreSQL 17
Databases (2)
postgres
shopmanagementsystem
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configuration
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (10)
bill_detail
bill_product
No limit
Query
Query History
Execute query
Alt F5
select * from orders;
Data Output
Messages
Notifications
SQL
Showing rows: 1 to 84
Page No: 1
of 1
order_id [PK] integer
order_date date
user_id integer
delivery_address character varying (50)
total_amt numeric (10,2)
order_status character varying (20)
1 13 2025-01-23 1 123 Main St 47.65 in process
2 23 2025-01-23 1 123 Main St 47.65 in process
3 24 2025-01-23 1 123 Main St 47.65 in process
4 26 2025-01-23 1 123 Main St 47.65 in process
5 27 2025-01-23 1 123 Main St 47.65 in process
6 28 2025-01-23 1 123 Main St 26.98 in process
7 31 2025-01-23 2 123 Main St 44.95 in process
8 32 2025-01-23 2 123 Main St 44.95 in process
9 40 2025-01-23 2 123 Main St 44.95 in process
10 43 2025-01-23 2 123 Main St 44.95 in process
11 44 2025-01-23 2 123 Main St 44.95 in process
12 49 2025-01-23 2 123 Main St 44.95 in process
13 50 2025-01-23 2 123 Main St 44.95 in process
14 51 2025-01-23 2 123 Main St 44.95 in process
15 52 2025-01-23 2 123 Main St 44.95 in process
Successfully run. Total query runtime: 112 msec. 84 rows affected.
Total rows: 84
Query complete 00:00:00.112
CRLF
Ln 1, Col 1
ENG INTL
4:21 PM 3/8/2025


My Shopping Store
localhost:3000/update-order-status
All Bookmarks
Admin Dashboard
Live
Update Order Status
Change the status of an existing order.
Order ID
Order Date
User ID
User Address
Total Amount
Order Status
Actions
8 2025-01-22T18:00:00.000Z 1 123 Main St $47.65 in process Change to delivered
9 2025-01-22T18:00:00.000Z 1 123 Main St $47.65 in process Change to delivered
13 2025-01-22T18:00:00.000Z 1 123 Main St $47.65 in process Change to delivered
14 2025-01-22T18:00:00.000Z 1 123 Main St $47.65 in process Change to delivered
15 2025-01- 1 123 Main St $47.65 in process Change to
ENG US
6:33 AM 3/8/2025

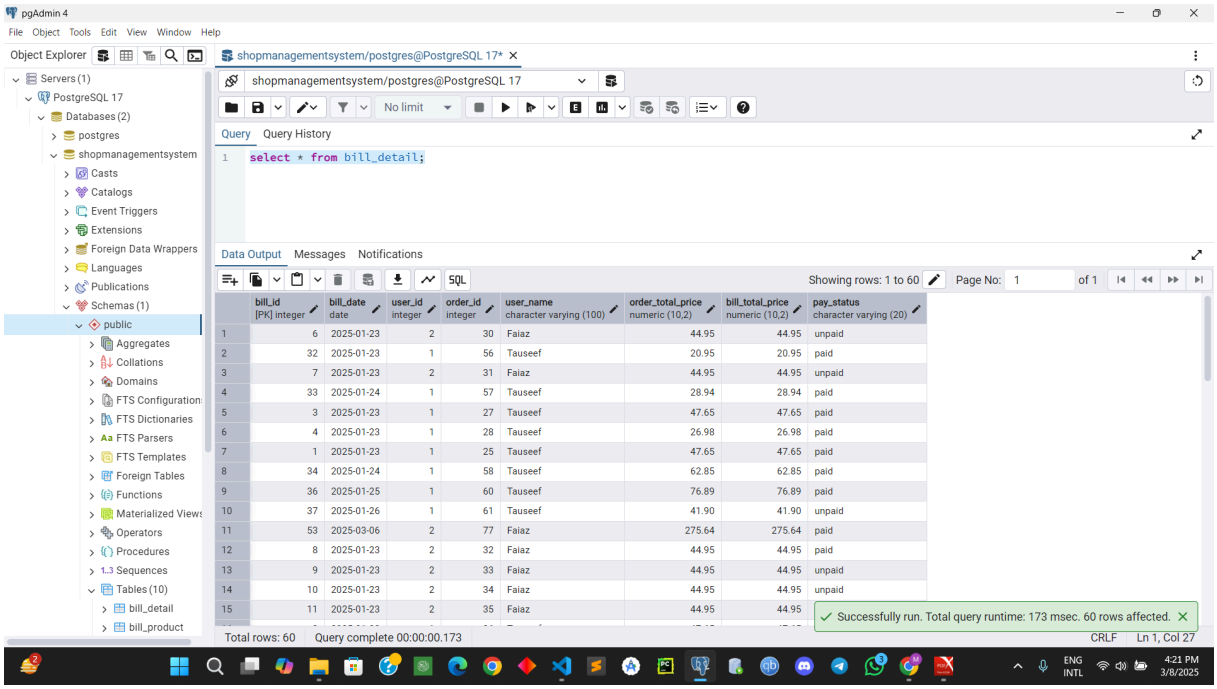
Order_details Page



Order_return Page

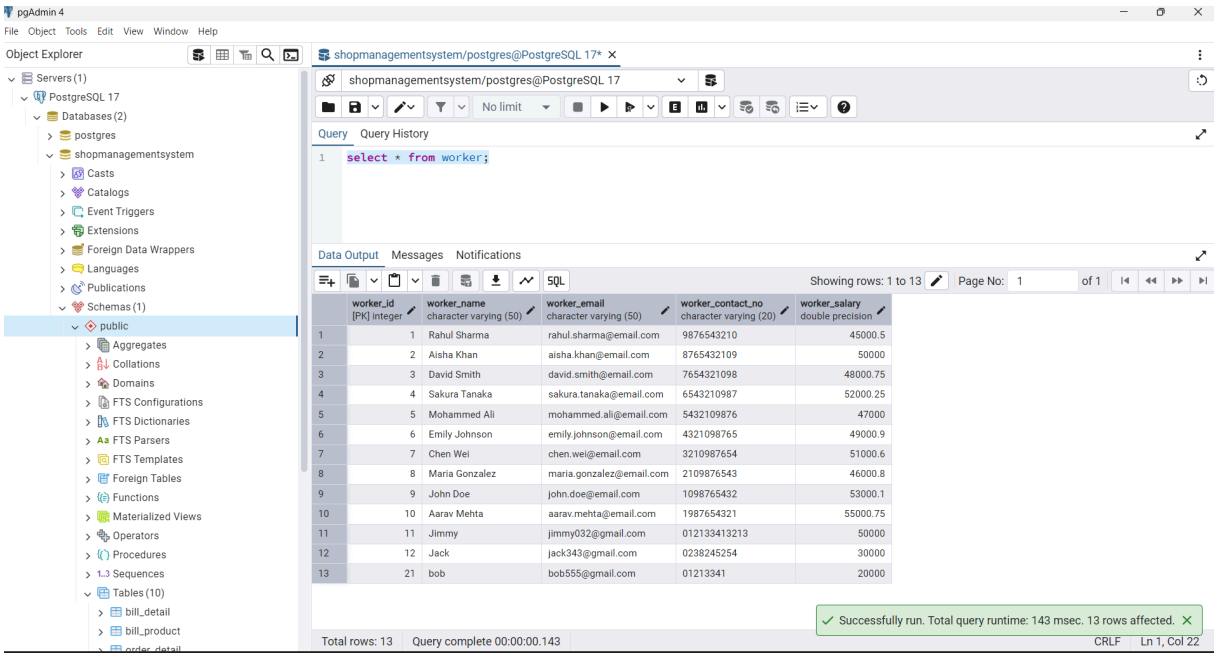


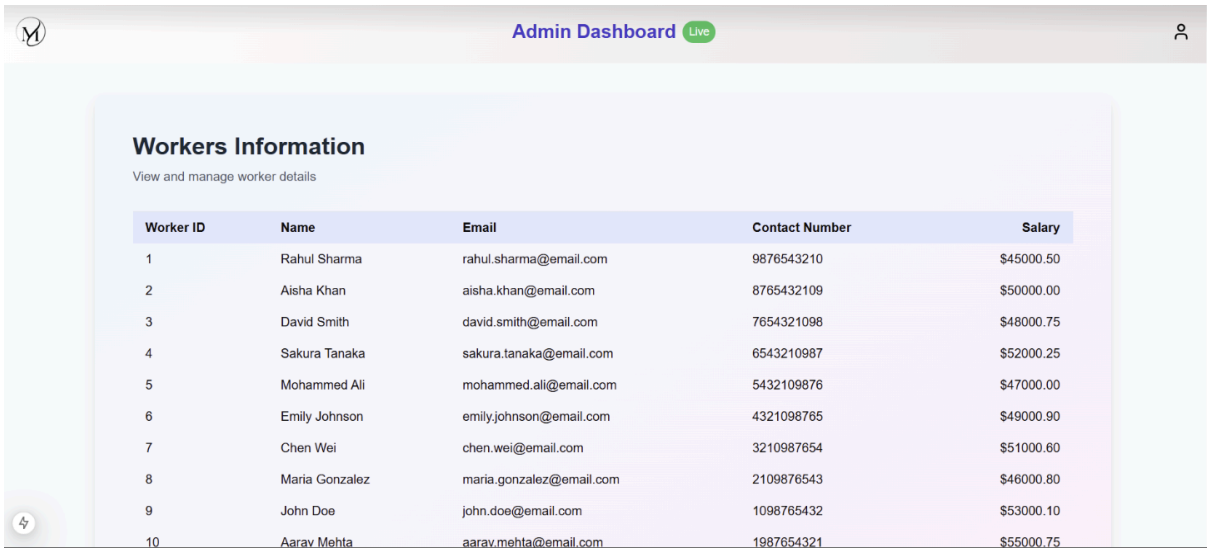
Bill_details Page



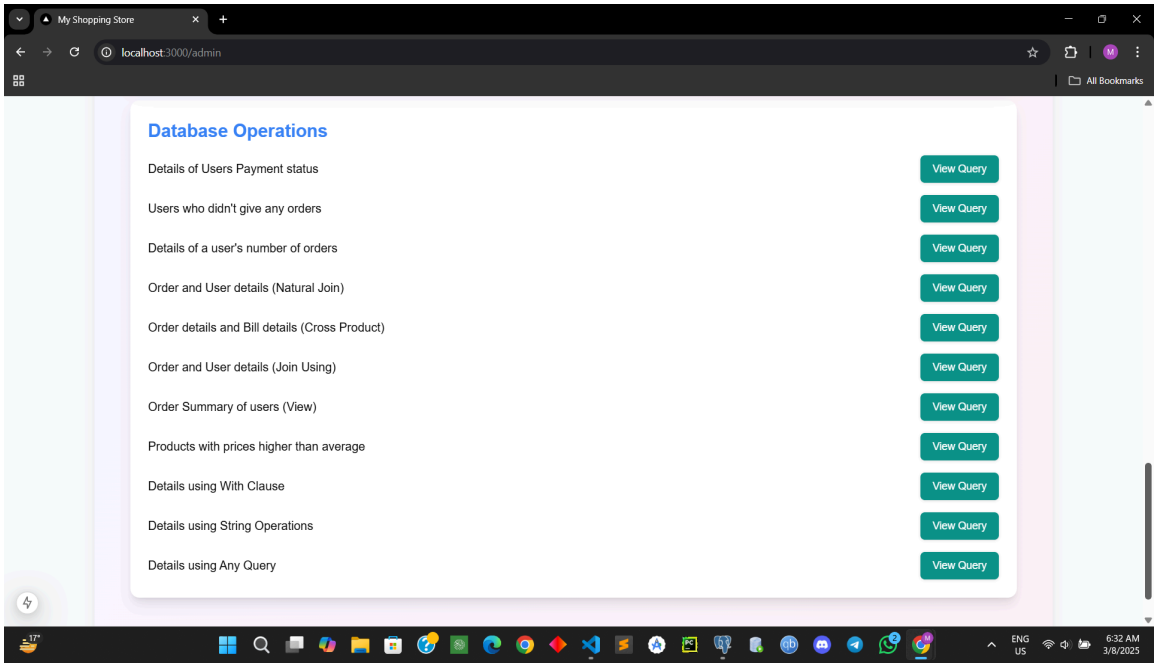
	bill_id [PK] integer	bill_date date	user_id integer	order_id integer	user_name character varying (100)	order_total_price numeric (10,2)	bill_total_price numeric (10,2)	pay_status character varying (20)
1	6	2025-01-23	2	30	Faiaz	44.95	44.95	unpaid
2	32	2025-01-23	1	56	Tauseef	20.95	20.95	paid
3	7	2025-01-23	2	31	Faiaz	44.95	44.95	unpaid
4	33	2025-01-24	1	57	Tauseef	28.94	28.94	paid
5	3	2025-01-23	1	27	Tauseef	47.65	47.65	paid
6	4	2025-01-23	1	28	Tauseef	26.98	26.98	paid
7	1	2025-01-23	1	25	Tauseef	47.65	47.65	paid
8	34	2025-01-24	1	58	Tauseef	62.85	62.85	paid
9	36	2025-01-25	1	60	Tauseef	76.89	76.89	paid
10	37	2025-01-26	1	61	Tauseef	41.90	41.90	unpaid
11	53	2025-03-06	2	77	Faiaz	275.64	275.64	paid
12	8	2025-01-23	2	32	Faiaz	44.95	44.95	paid
13	9	2025-01-23	2	33	Faiaz	44.95	44.95	unpaid
14	10	2025-01-23	2	34	Faiaz	44.95	44.95	unpaid
15	11	2025-01-23	2	35	Faiaz	44.95	44.95	

Workers Page



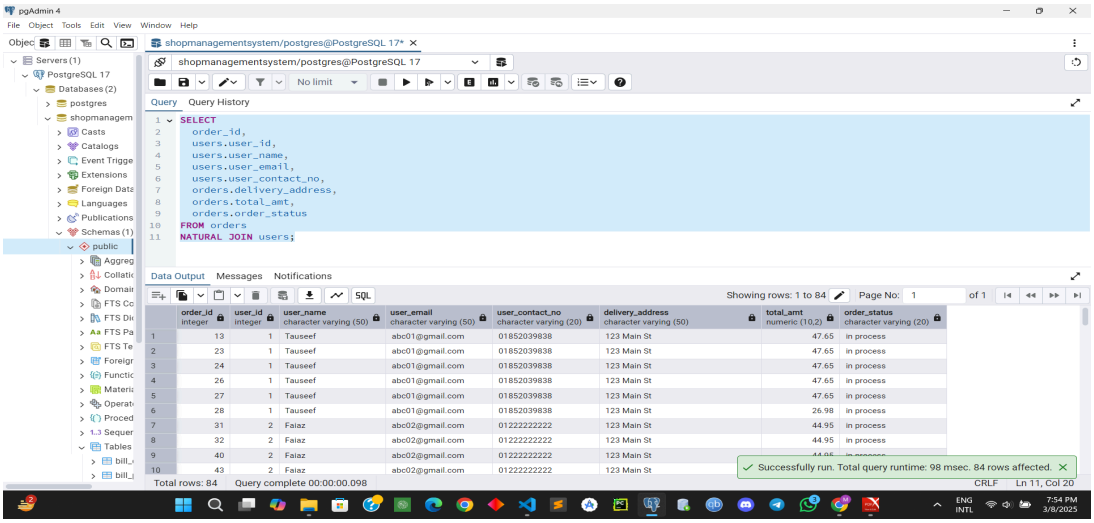


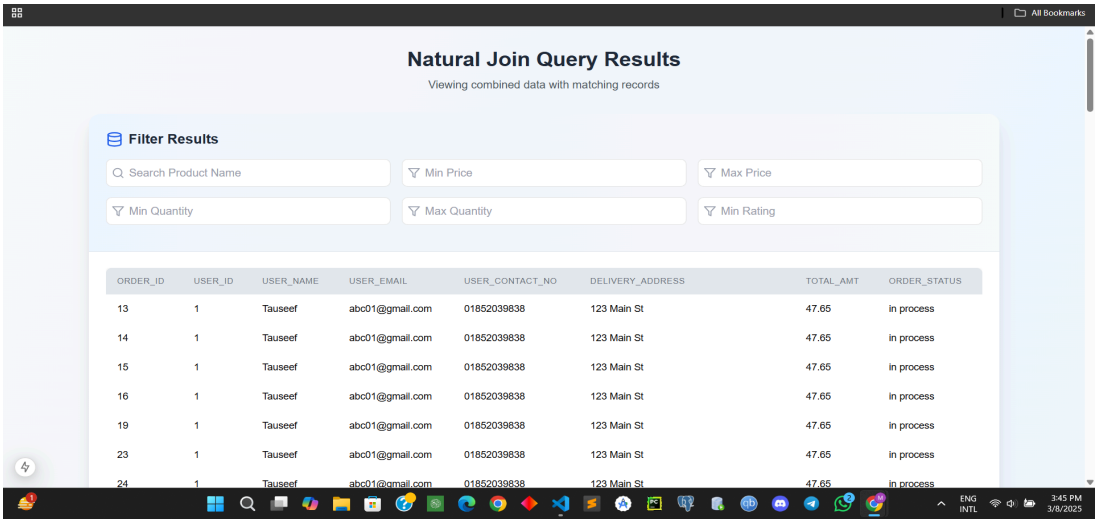
7. Query statement, Relational Algebra Expression, SQL statement and snapshots of the outputs



- ***Natural Join —***
  Relational Algebra Expression:
  π order_id, users.user_id, users.user_name, users.user_email, users.user_contact_no, orders.delivery_address, orders.total_amt, orders.order_status (orders ⋈ users)

```
const response: QueryResult = await api.executeRawQuery(`
  SELECT
      order_id,
      users.user_id,
      users.user_name,
      users.user_email,
      users.user_contact_no,
      orders.delivery_address,
      orders.total_amt,
      orders.order_status
  FROM orders
  NATURAL JOIN users;

`);
```

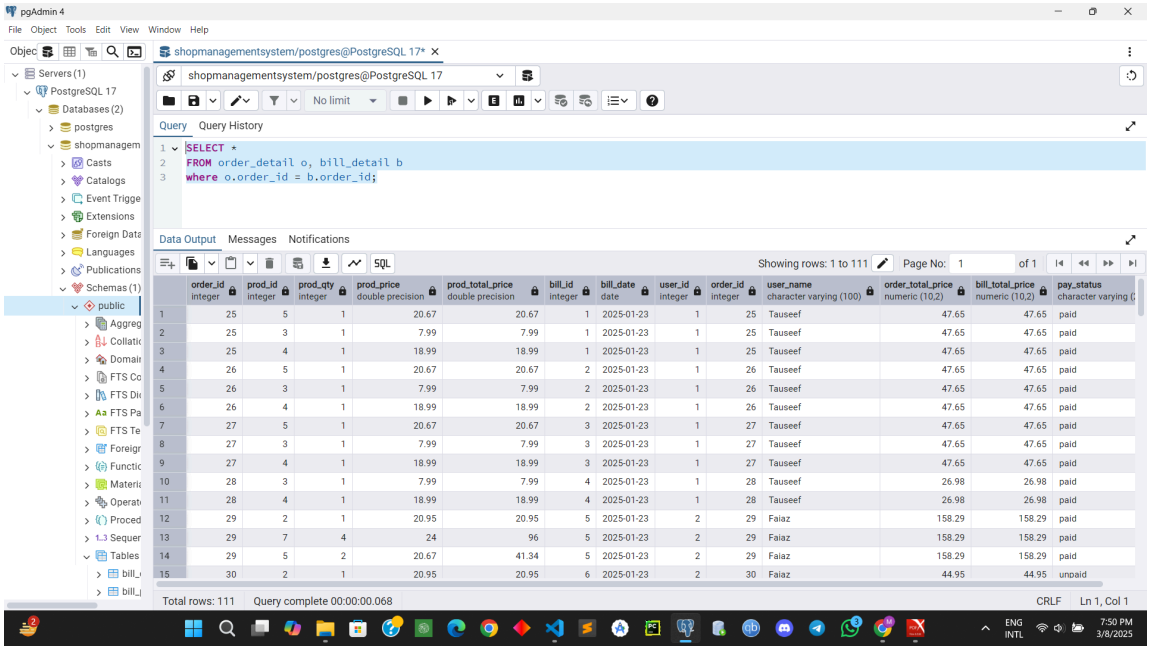


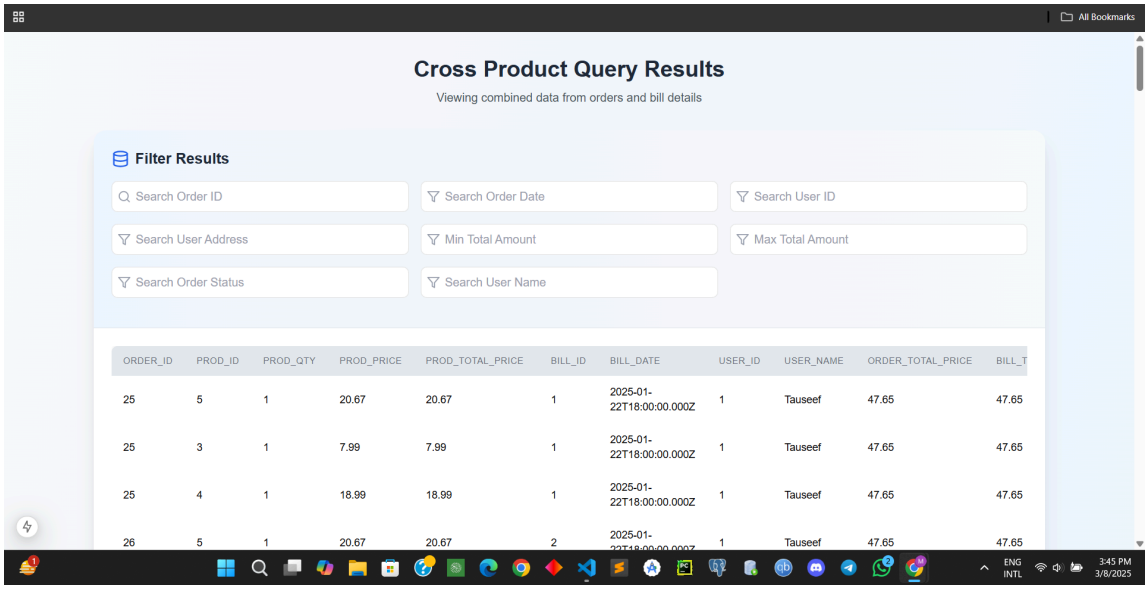


- ***Cross Product —***

Relational algebra expression :

σ o.order_id = b.order_id (order_detail × bill_detail)

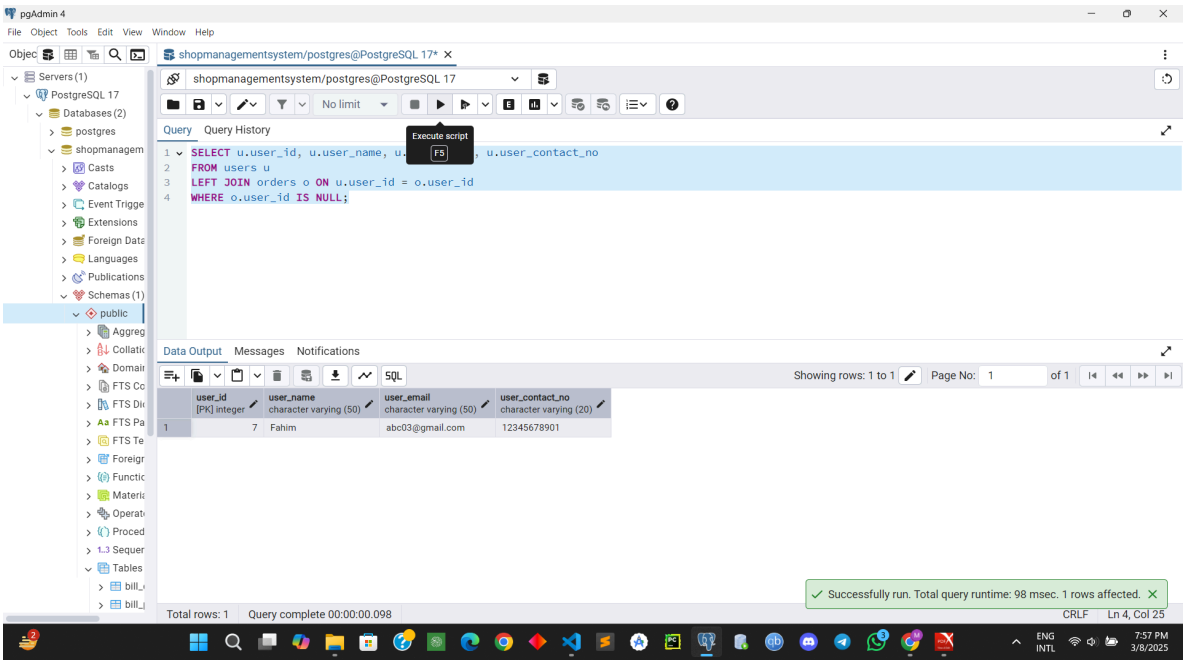


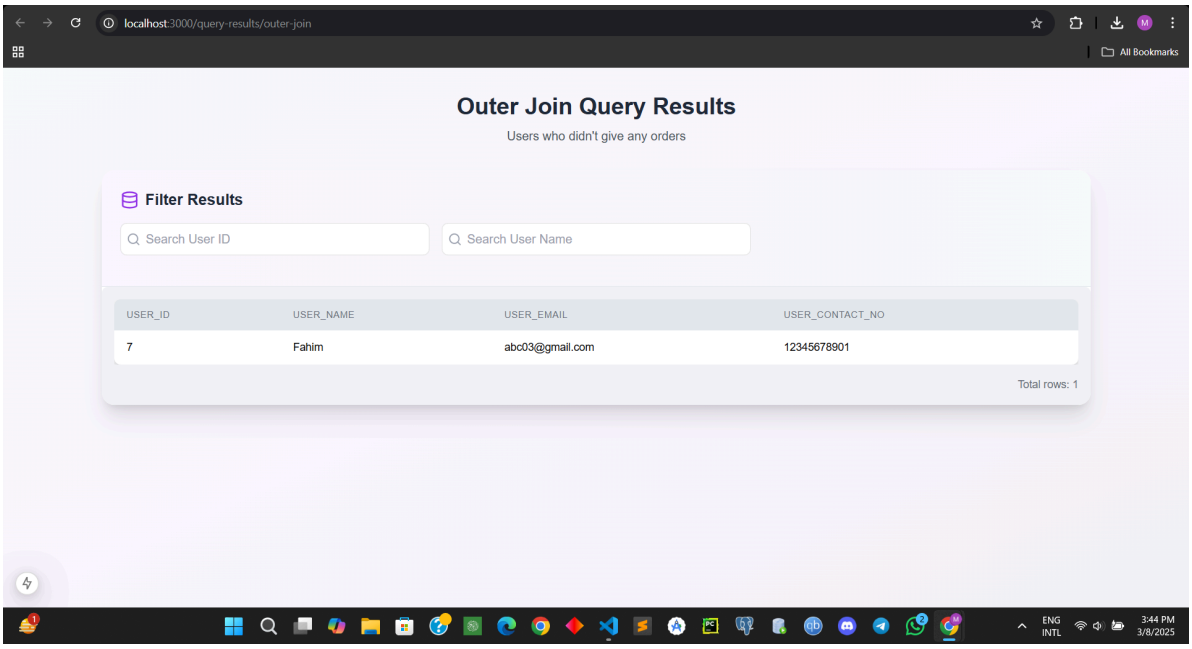


- ***Outer Join —***

Relational Algebra Expression :

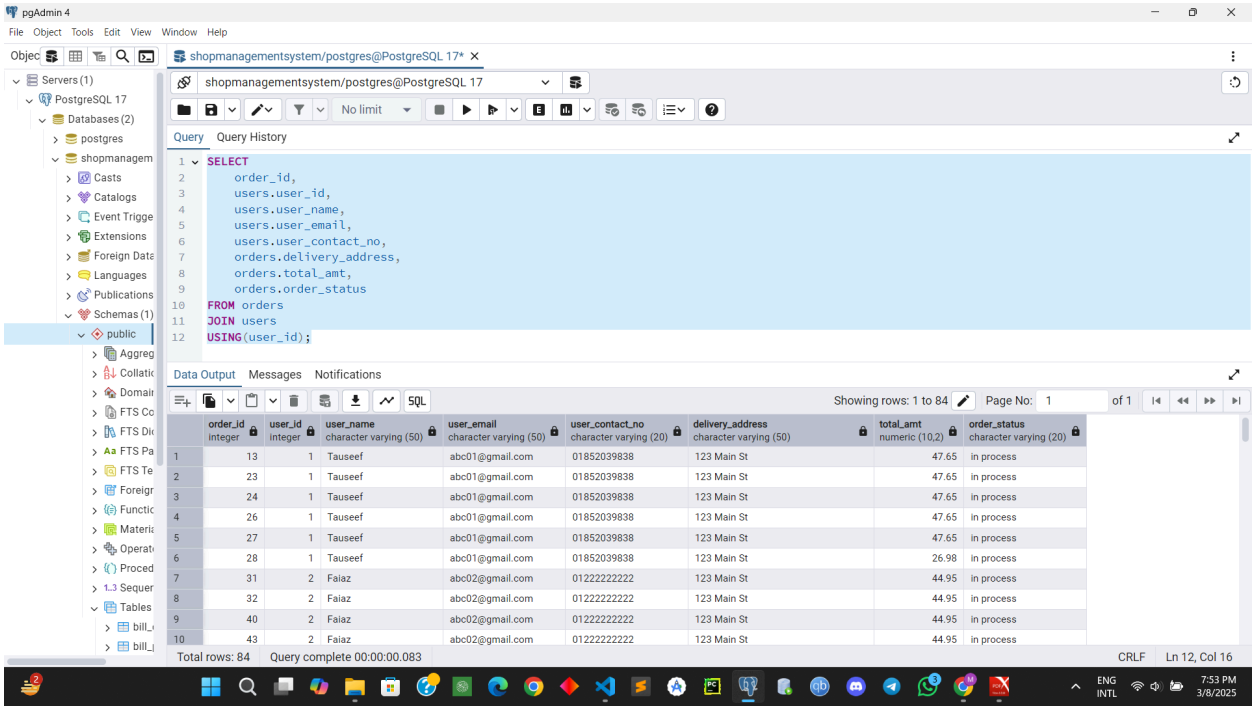
π u.user_id, u.user_name, u.user_email, u.user_contact_no (users ⟕ orders) ⋈ σ o.user_id IS NULL

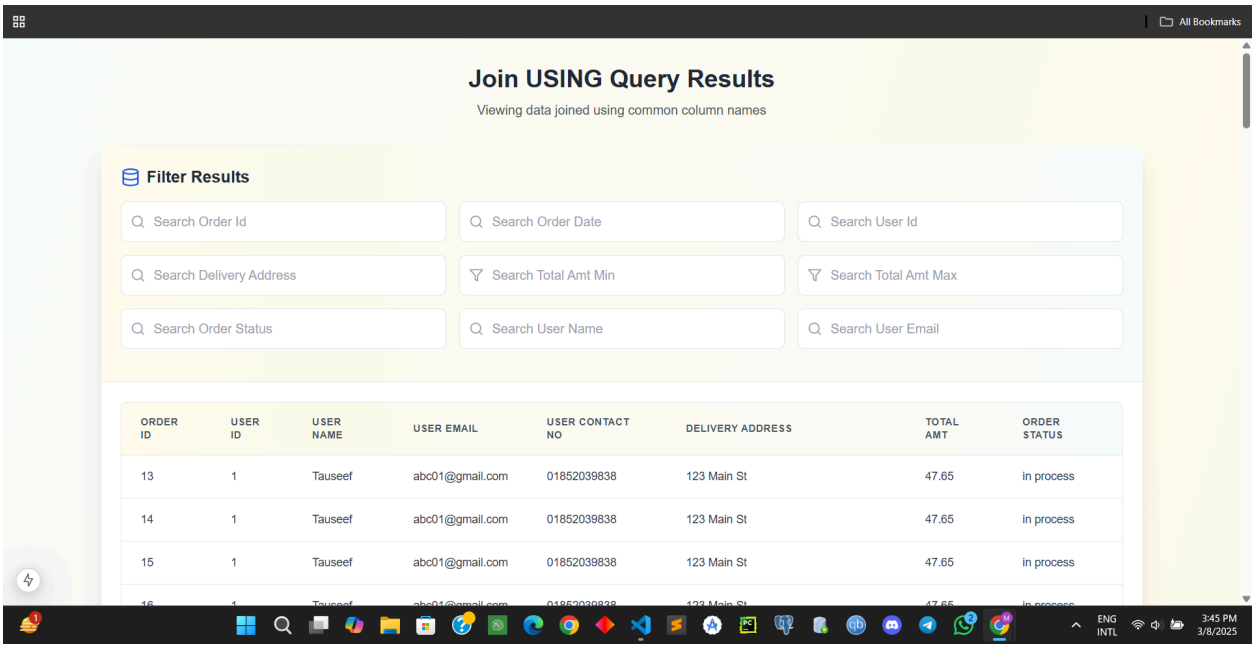




- ***Join with USING —***
  Relational Algebra Expression :

  π order_id, users.user_id, users.user_name, users.user_email, users.user_contact_no, orders.delivery_address, orders.total_amt, orders.order_status (orders ⋈user_id users)



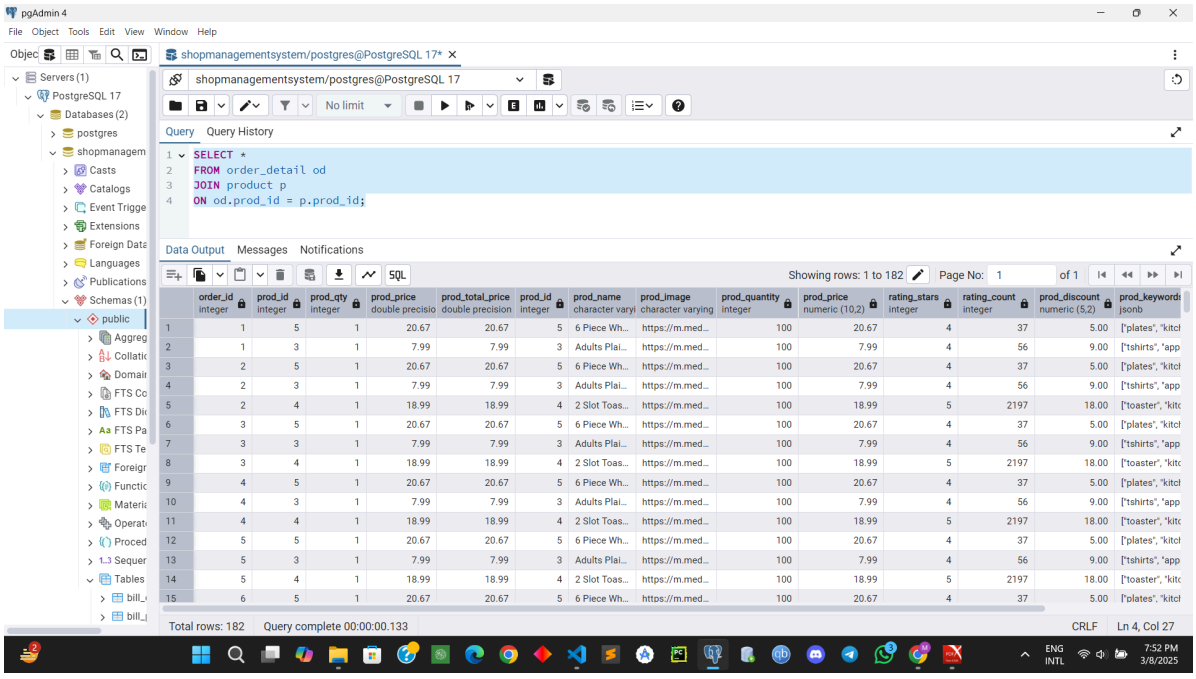


- ***Join with ON —***

Relational algebra expression :

order_detail ⋈ product

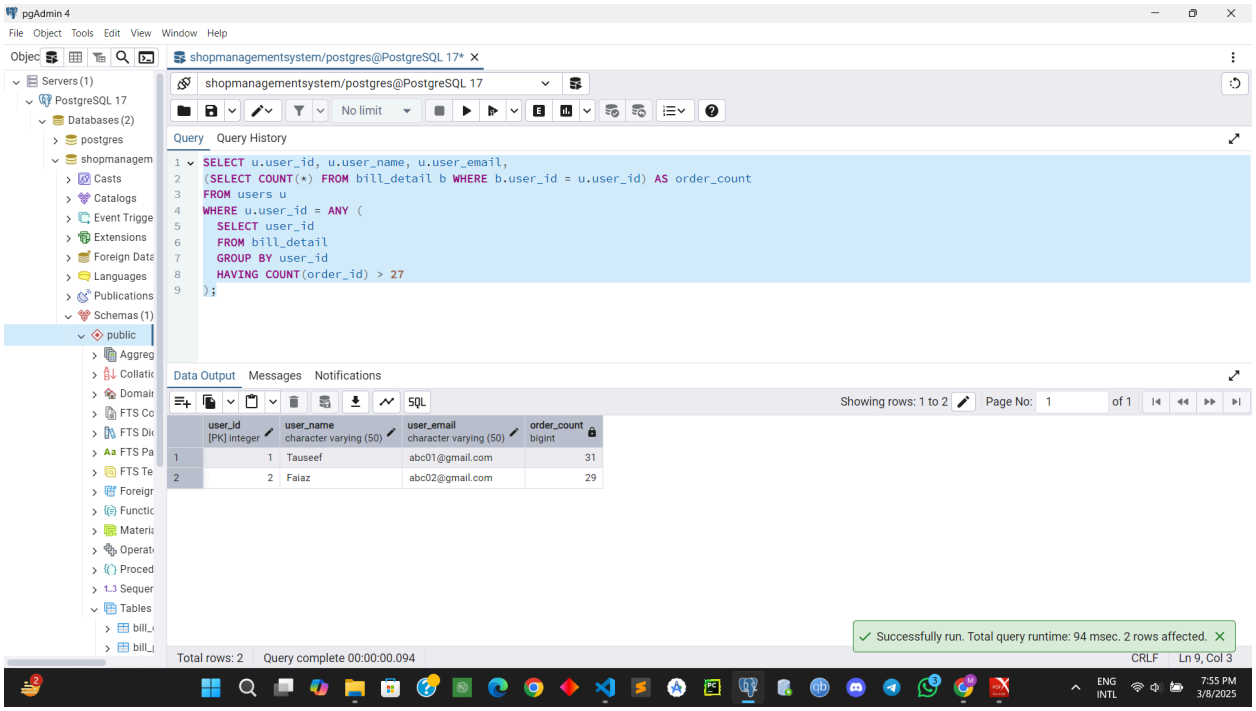
Where the join condition is: order_detail.prod_id = product.prod_id

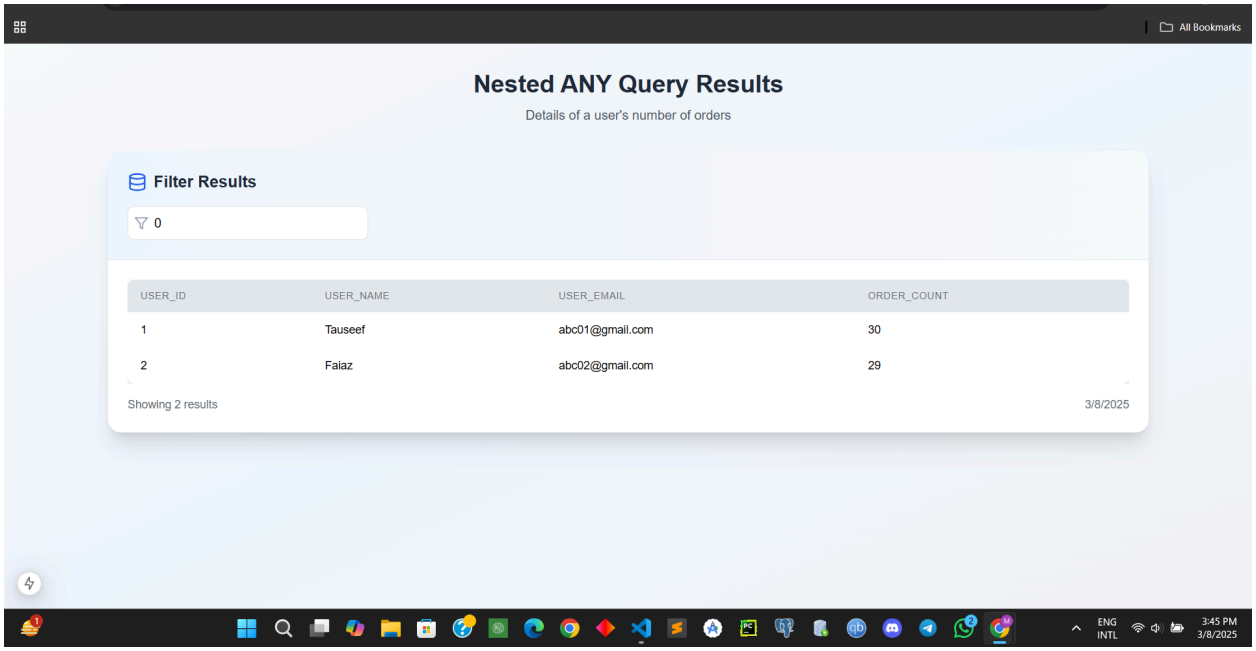


- ***Nested subquery with ANY clause –***

Relational Algebra Expression:

π u.user_id, u.user_name, u.user_email, (G COUNT(*) → order_count (σ b.user_id = u.user_id (bill_detail))) (σ u.user_id ∈ (π user_id (G user_id; COUNT(order_id)>27 (bill_detail))) (users))

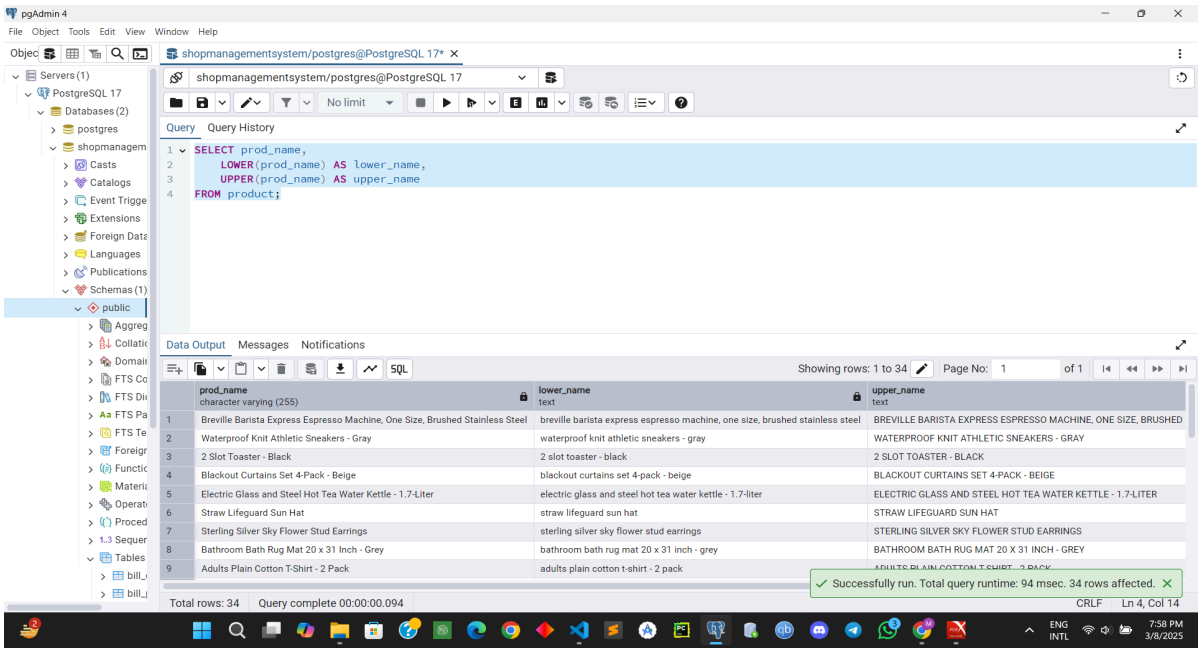


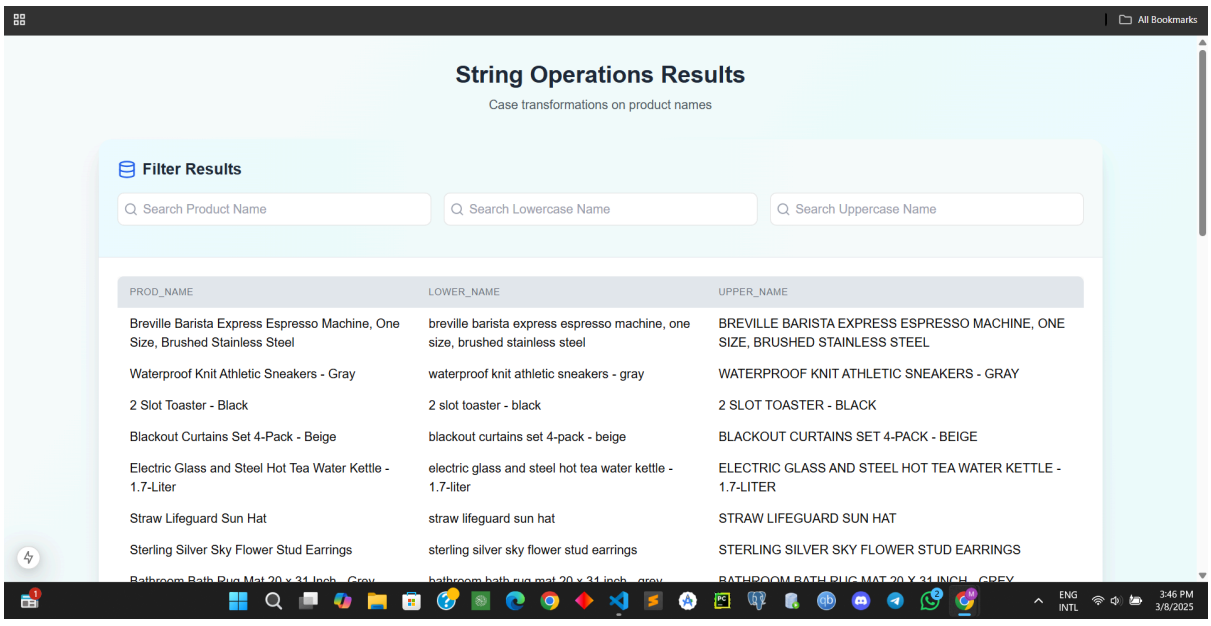


- ***String Operations —***

Relational Algebra Expression :

π prod_name, LOWER(prod_name)→lower_name, UPPER(prod_name)→upper_name (product)

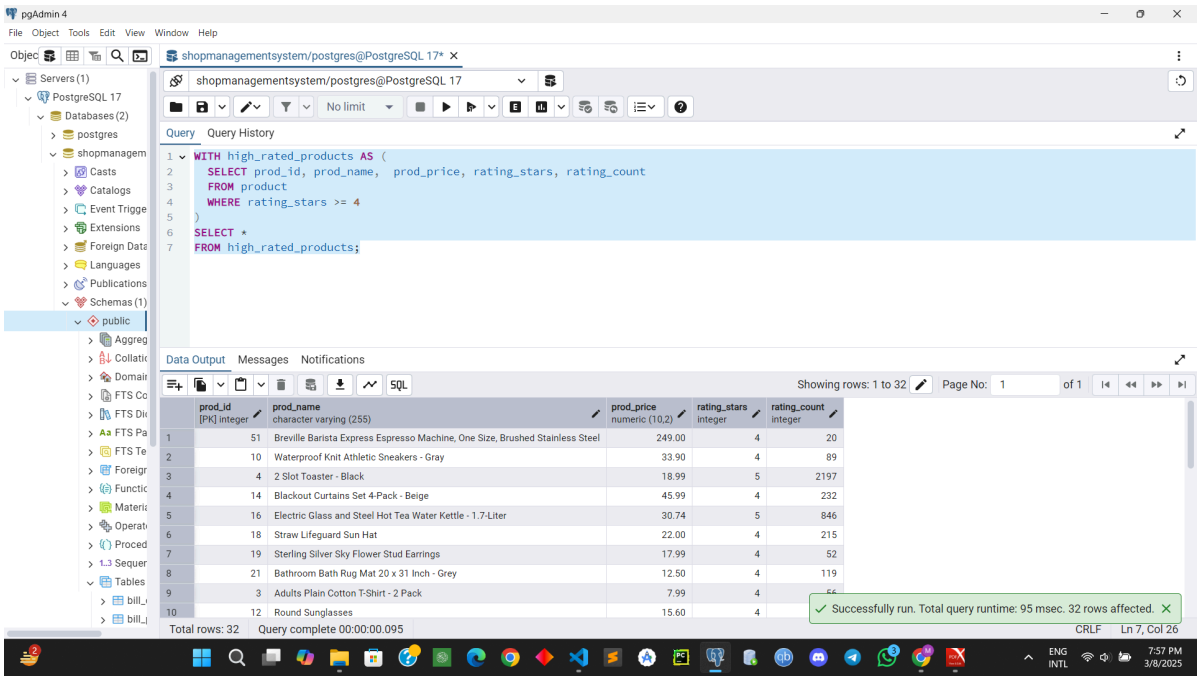


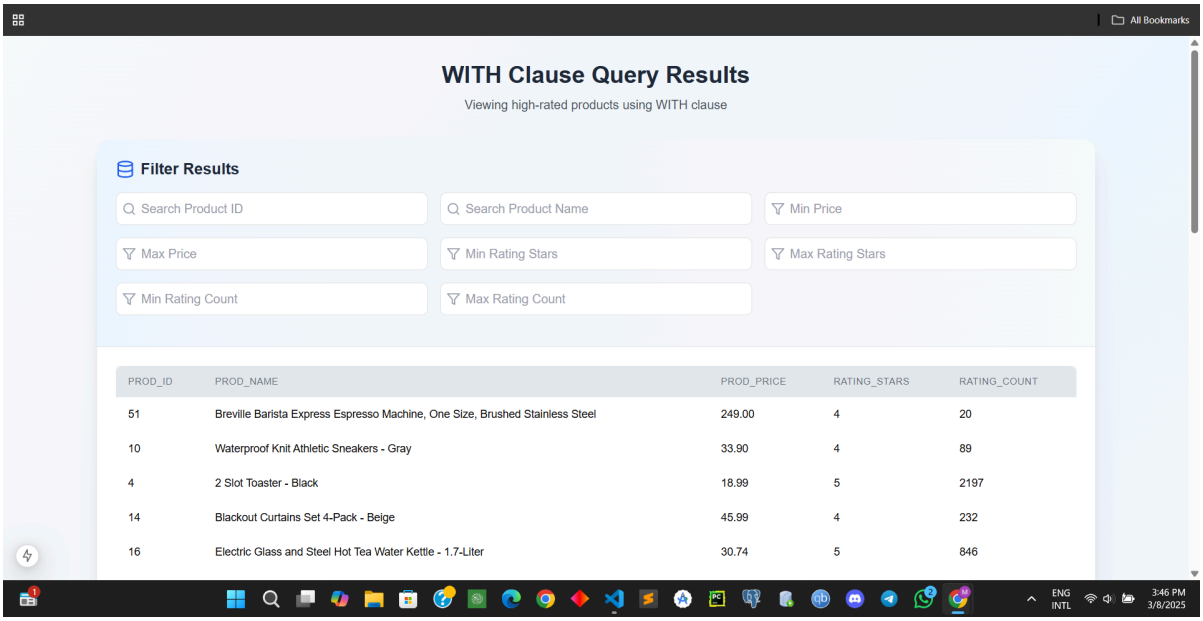


- ***With Clause —***

Relational Algebra Expression:

high_rated_products = π prod_id, prod_name, prod_price, rating_stars, rating_count (σ rating_stars≥4 (product))

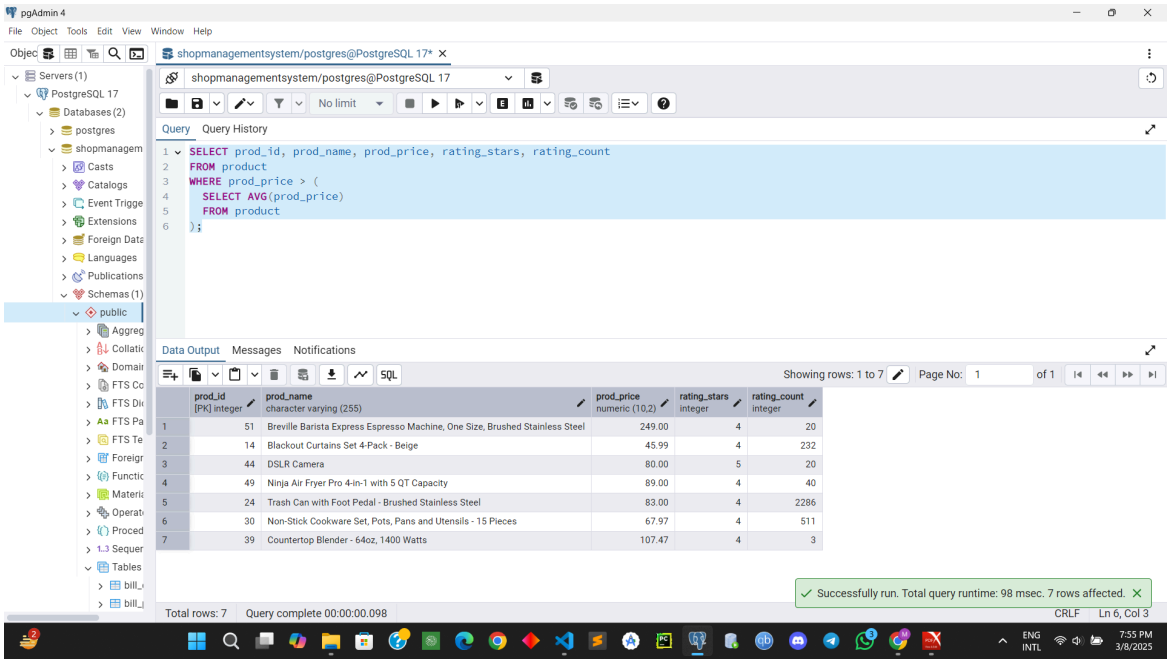


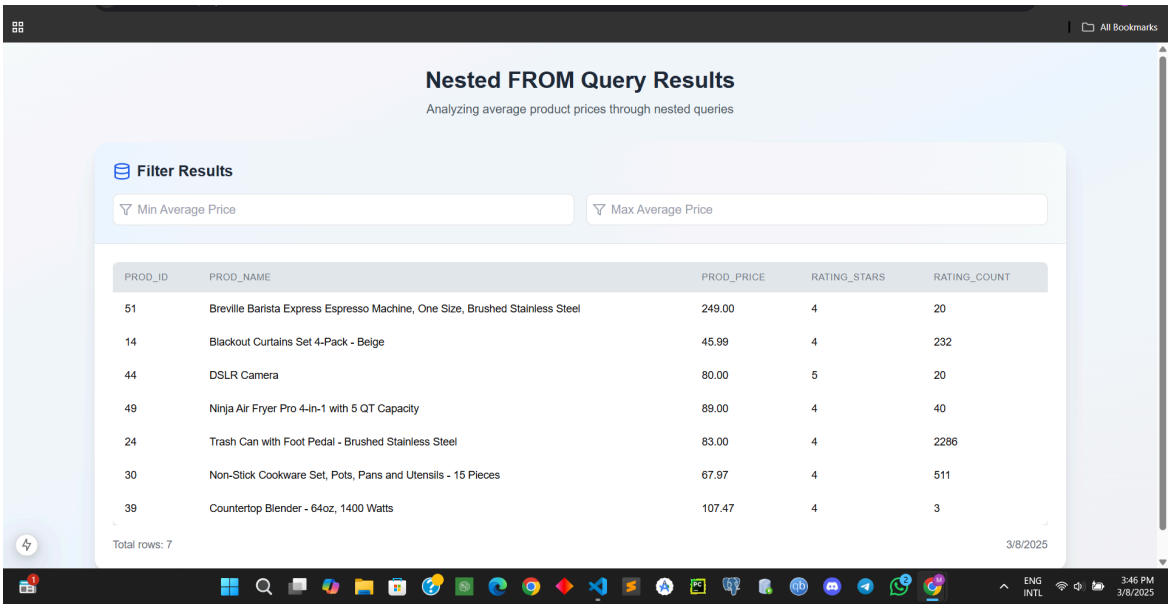


- ***Nested subquery in from and select clause —***

Relational Algebra Expression :

σ prod_price > (G AVG(prod_price) (product)) (π prod_id, prod_name, prod_price, rating_stars, rating_count (product))

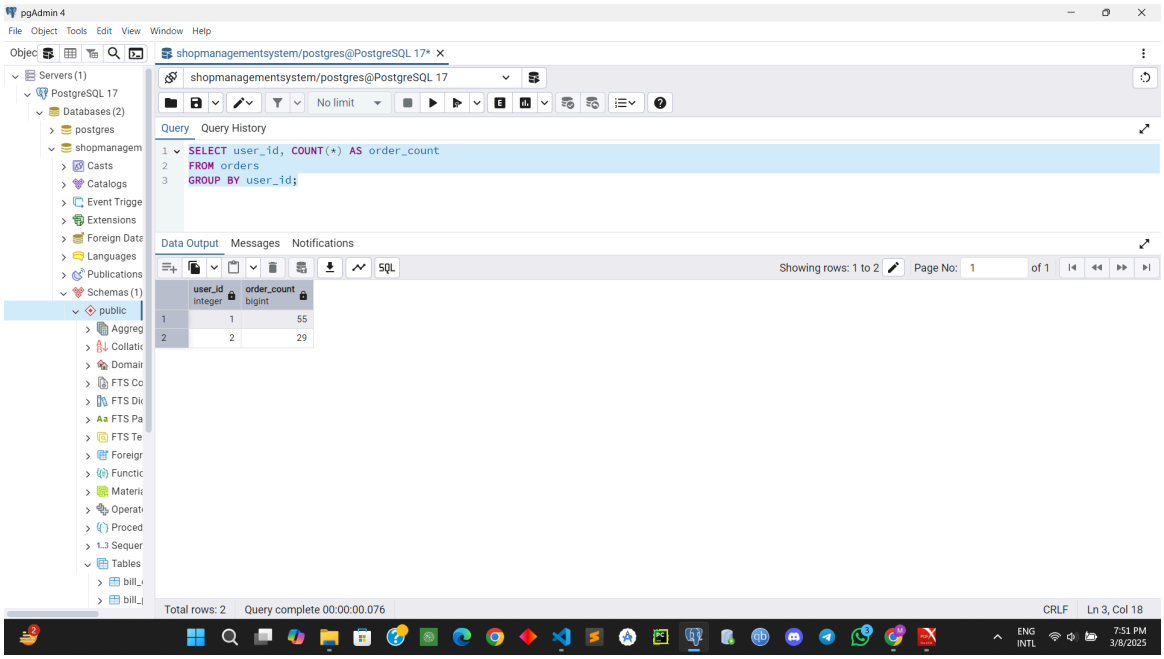




- ***Group By —***

  Relational Algebra Expression:

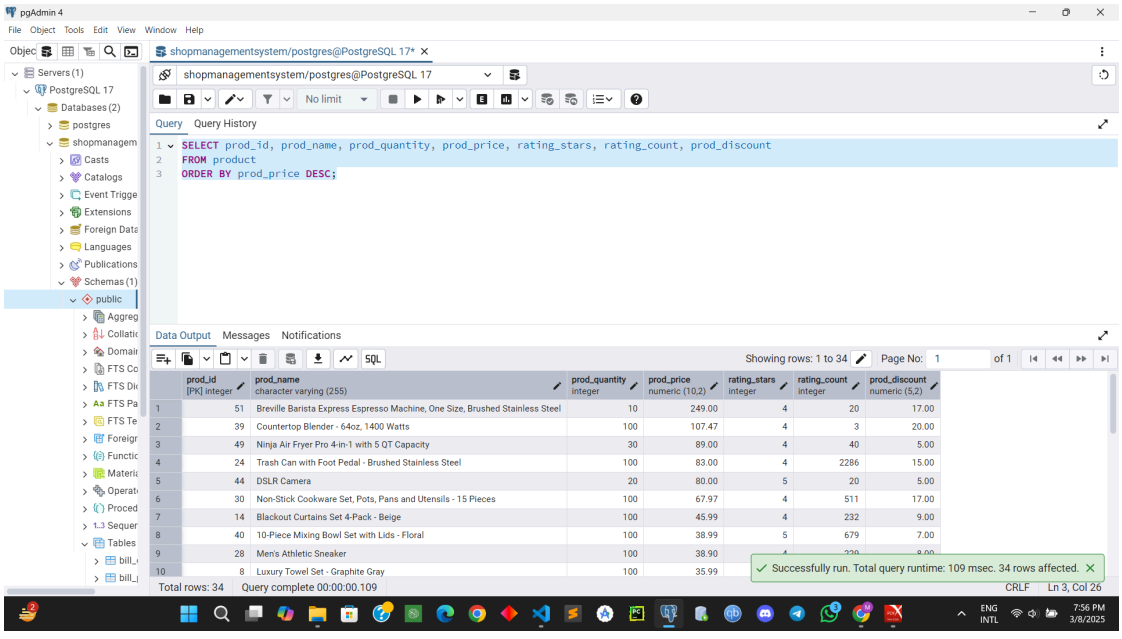
  G user_id; COUNT(*) → order_count (orders)



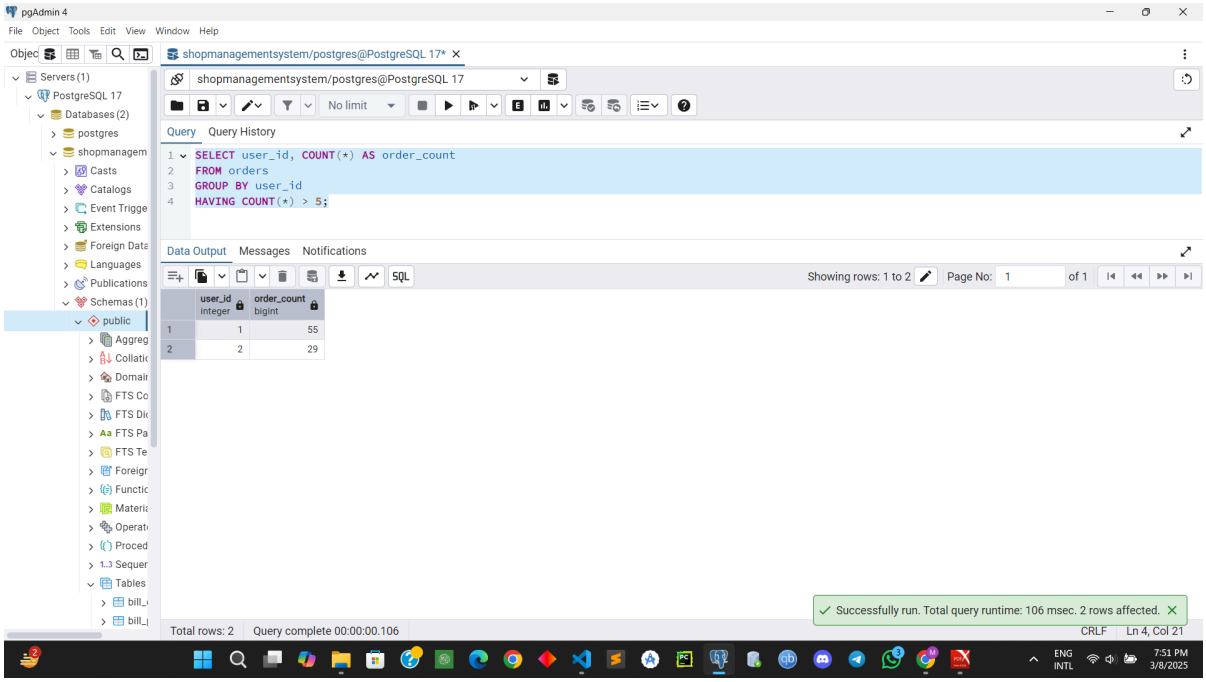
- ***Order By —***

  Relational Algebra Expression :

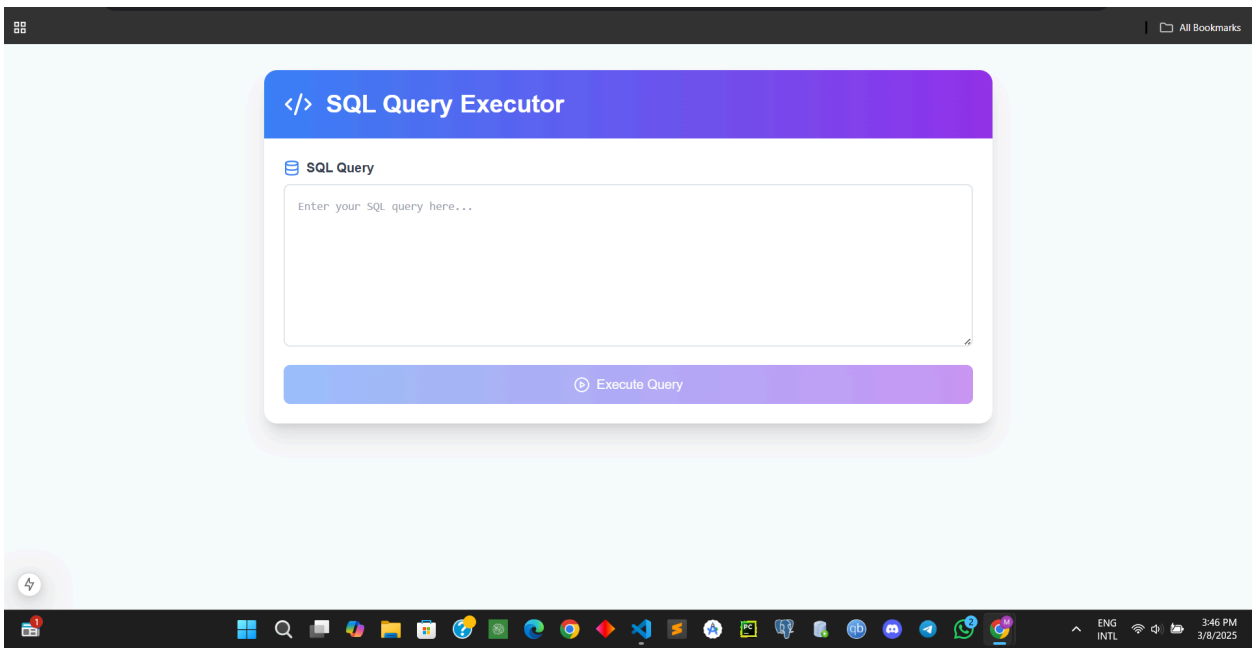
  τ prod_price DESC (π prod_id, prod_name, prod_quantity, prod_price, rating_stars, rating_count, prod_discount (product))



- ***Having clause —***
  Relational Algebra Expression:
  G user_id; COUNT(*) > 5 → order_count (orders)



- ***Provide any query :***



- ***Update function —***

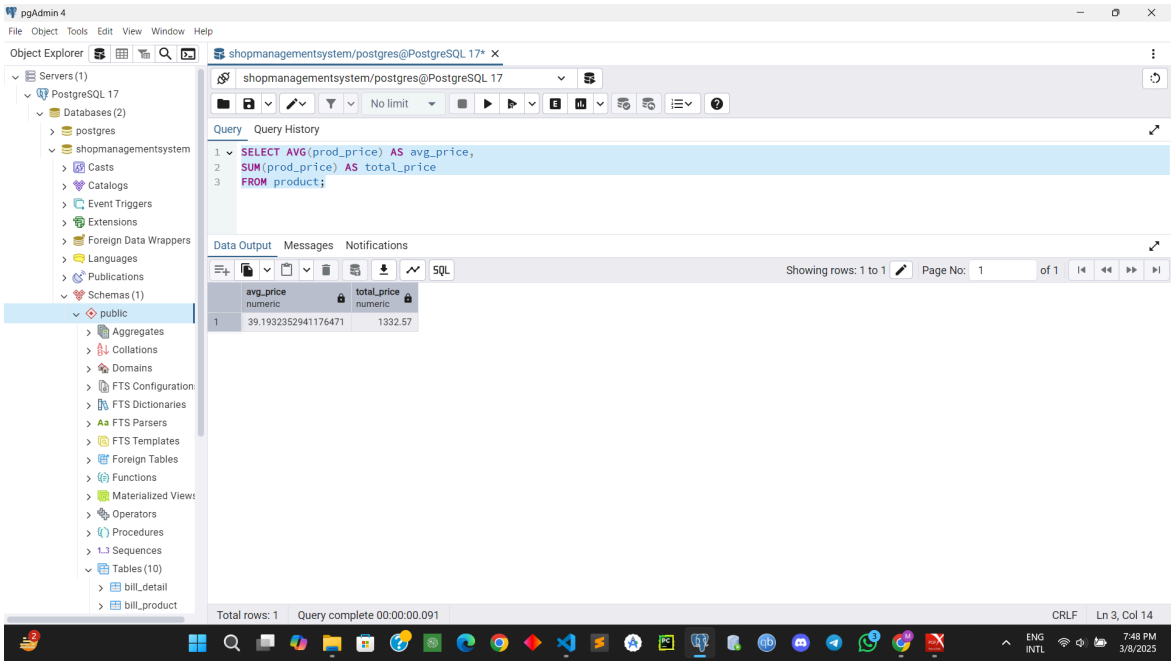
```
CREATE OR REPLACE FUNCTION update_bill_detail_user_name()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE bill_detail
    SET user_name = NEW.user_name
    WHERE user_id = NEW.user_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

- ***Aggregate functions or built-in functions and other functions —***

```
CREATE TRIGGER trigger_update_bill_user_name
AFTER UPDATE ON users
FOR EACH ROW
WHEN (OLD.user_name IS DISTINCT FROM NEW.user_name)
EXECUTE FUNCTION update_bill_detail_user_name();
```

Relational Algebra expression:

π AVG(prod_price)→avg_price, SUM(prod_price)→total_price (product)



8. Create views and use those views in answering queries

View create statement

Here a view named “order_summary” is created.

```sql
CREATE VIEW order_summary AS
SELECT
    o.order_id,
    o.order_date,
    u.user_id,
    u.user_name,
    u.user_email,
    o.delivery_address,
    od.prod_id,
    p.prod_name,
    od.prod_qty,
    od.prod_price,
    od.prod_total_price,
    o.total_amt,
    o.order_status
FROM orders o
JOIN users u ON o.user_id = u.user_id
JOIN order_detail od ON o.order_id = od.order_id
JOIN product p ON od.prod_id = p.prod_id;
```

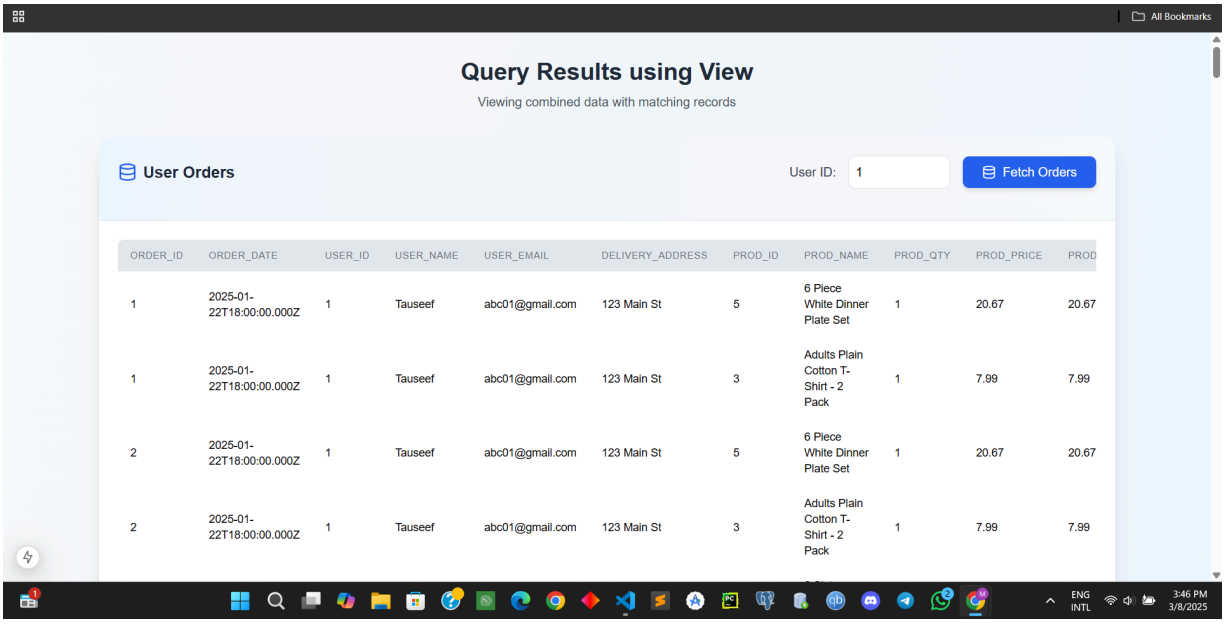
Relational Algebra Expression :

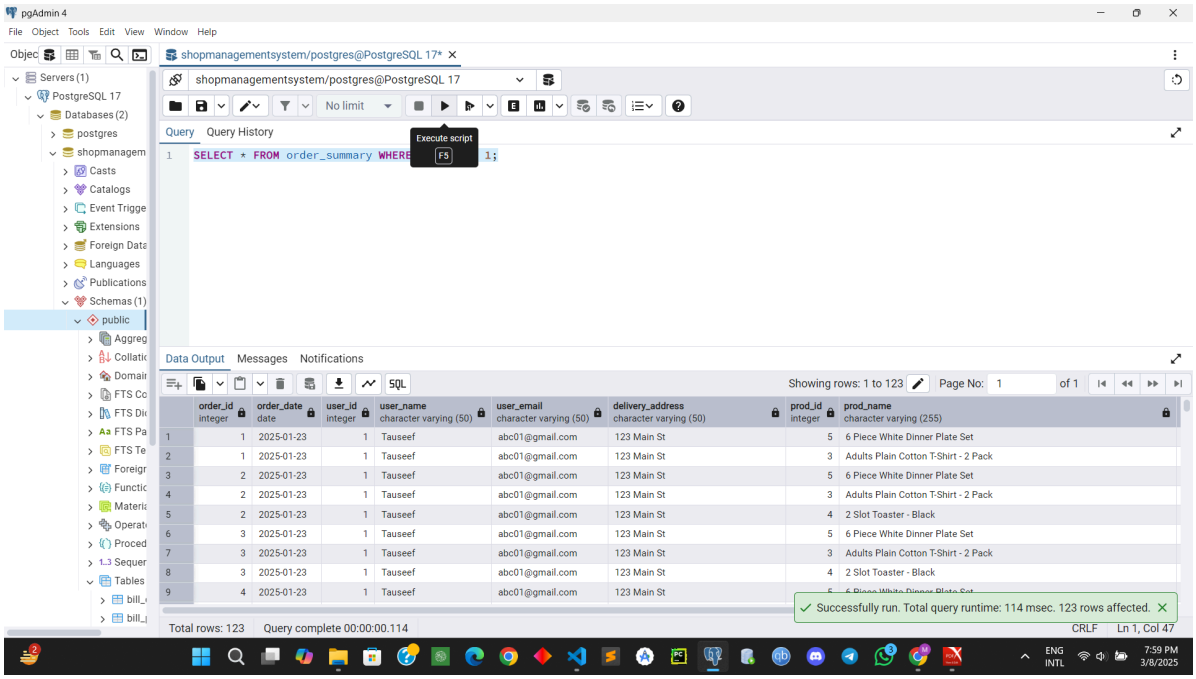
order_summary = π all columns (σ user_id=1 (orders ⋈ users ⋈ order_detail ⋈ product)

Where the join conditions are:

- orders.user_id = users.user_id
- orders.order_id = order_detail.order_id
- order_detail.prod_id = product.prod_id

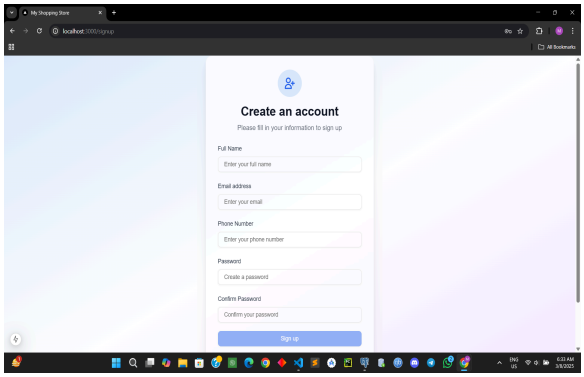
View Result

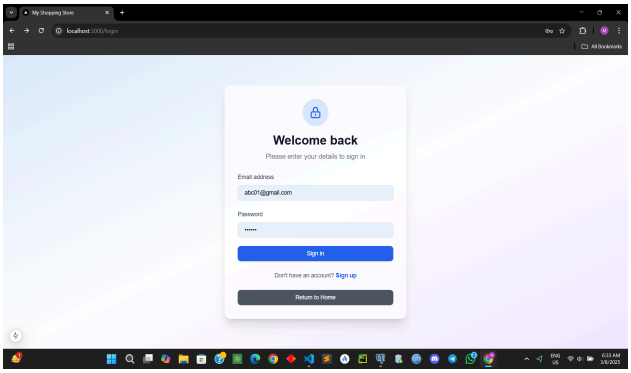

Query Results using View
Viewing combined data with matching records
User Orders
User ID: 1
Fetch Orders
ORDER_ID | ORDER_DATE | USER_ID | USER_NAME | USER_EMAIL | DELIVERY_ADDRESS | PROD_ID | PROD_NAME | PROD_QTY | PROD_PRICE | PROD
1 | 2025-01-22T18:00:00.000Z | 1 | Tauseef | abc01@gmail.com | 123 Main St | 5 | 6 Piece White Dinner Plate Set | 1 | 20.67 | 20.67
1 | 2025-01-22T18:00:00.000Z | 1 | Tauseef | abc01@gmail.com | 123 Main St | 3 | Adults Plain Cotton T-Shirt - 2 Pack | 1 | 7.99 | 7.99
2 | 2025-01-22T18:00:00.000Z | 1 | Tauseef | abc01@gmail.com | 123 Main St | 5 | 6 Piece White Dinner Plate Set | 1 | 20.67 | 20.67
2 | 2025-01-22T18:00:00.000Z | 1 | Tauseef | abc01@gmail.com | 123 Main St | 3 | Adults Plain Cotton T-Shirt - 2 Pack | 1 | 7.99 | 7.99


pgAdmin 4
SELECT * FROM order_summary WHERE
Execute script
F5
Data Output
Messages
Notifications
Showing rows: 1 to 123
Page No: 1
of 1
order_id | order_date | user_id | user_name | user_email | delivery_address | prod_id | prod_name
1 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 5 | 6 Piece White Dinner Plate Set
1 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 3 | Adults Plain Cotton T-Shirt - 2 Pack
2 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 5 | 6 Piece White Dinner Plate Set
2 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 3 | Adults Plain Cotton T-Shirt - 2 Pack
2 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 4 | 2 Slot Toaster - Black
3 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 5 | 6 Piece White Dinner Plate Set
3 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 3 | Adults Plain Cotton T-Shirt - 2 Pack
3 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St | 4 | 2 Slot Toaster - Black
4 | 2025-01-23 | 1 | Tauseef | abc01@gmail.com | 123 Main St
Successfully run. Total query runtime: 114 msec. 123 rows affected.
Total rows: 123
Query complete 00:00:00.114

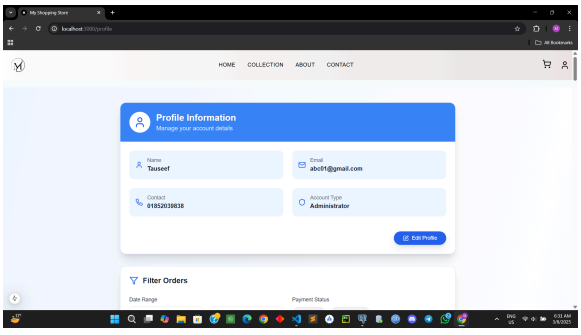
9. Some snaps of our web application

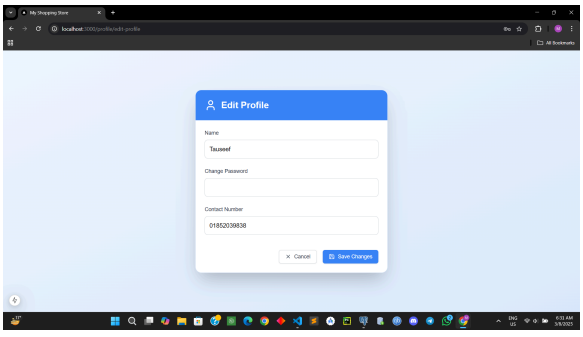
Login / SignUp page



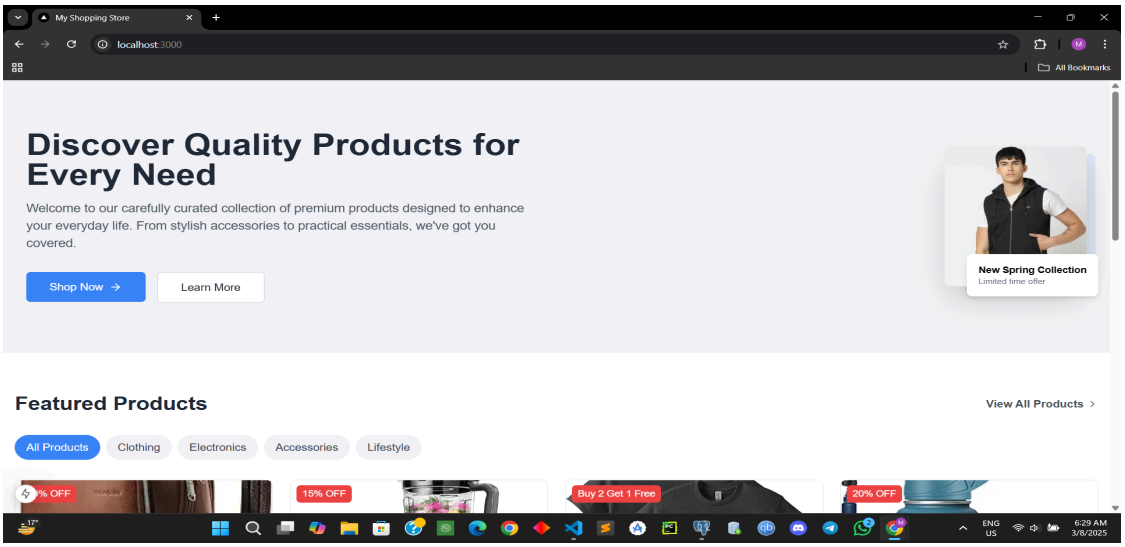


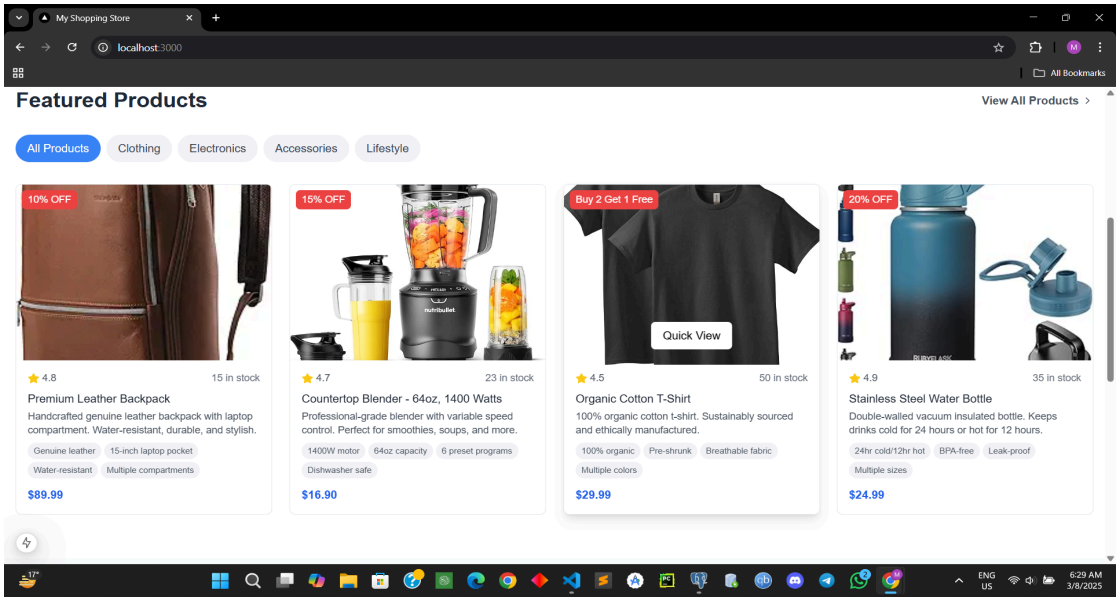
Profile Information and edit profile page

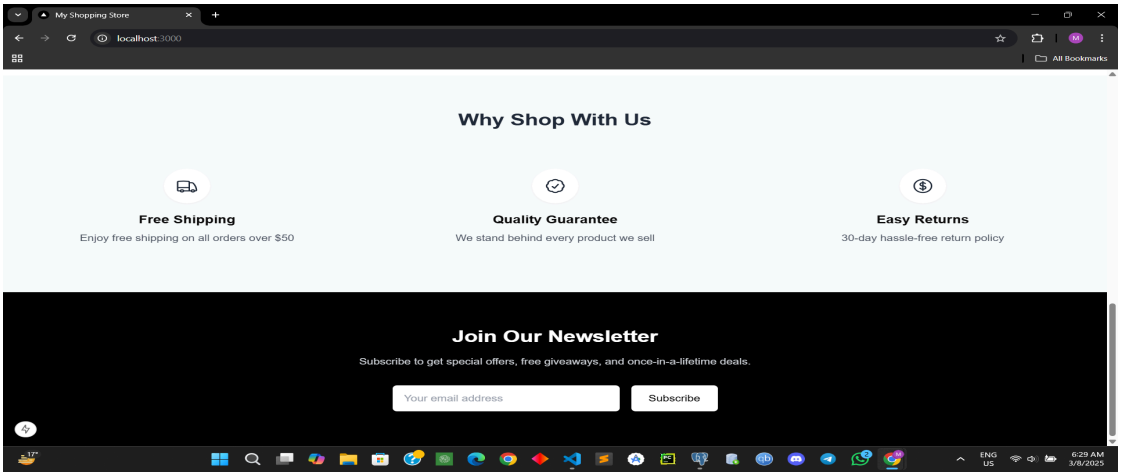


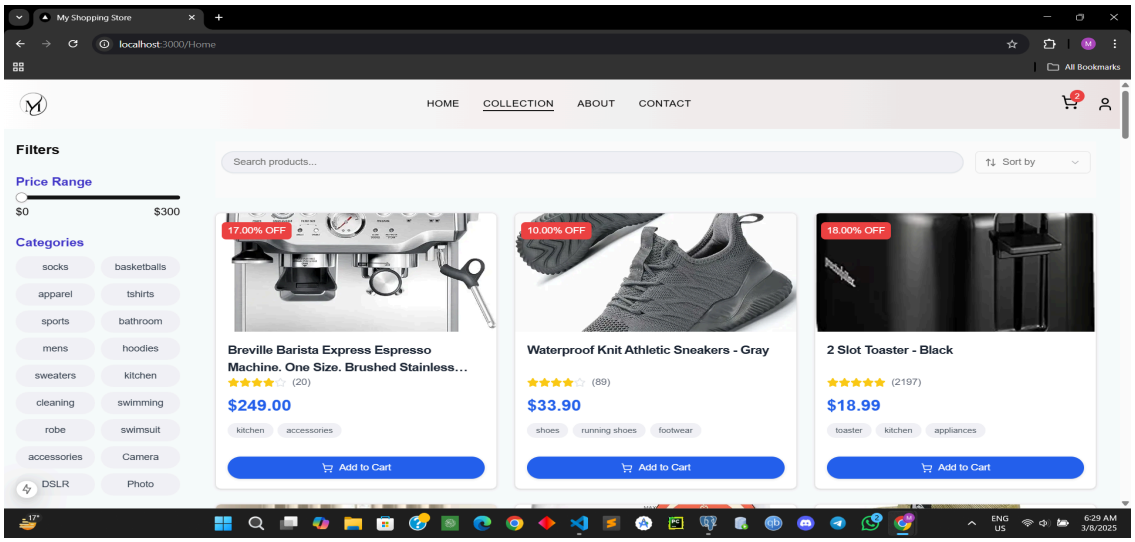


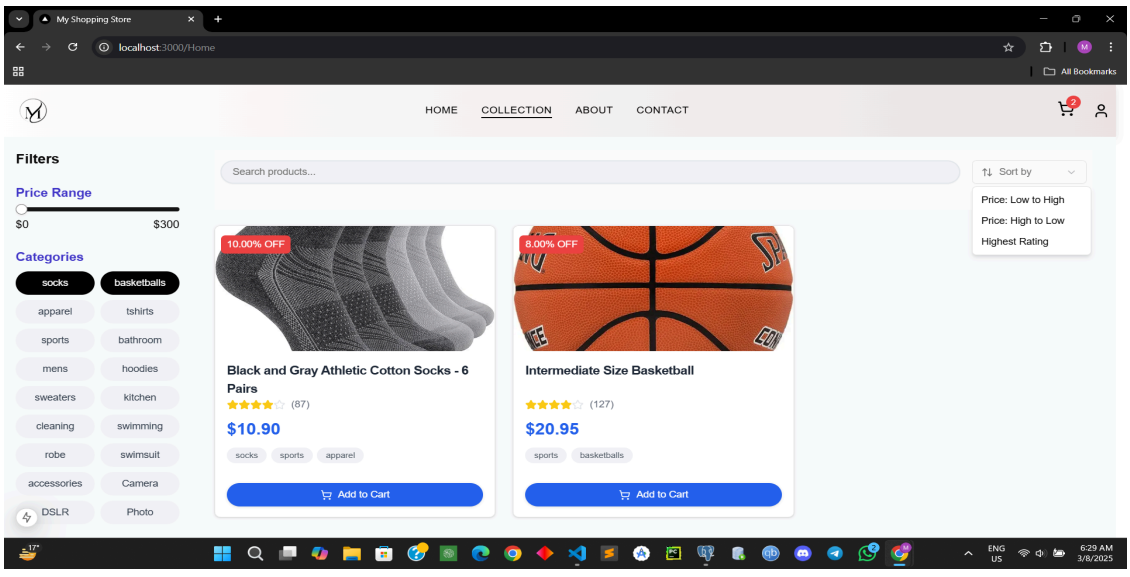
Main Home Page (View Products)

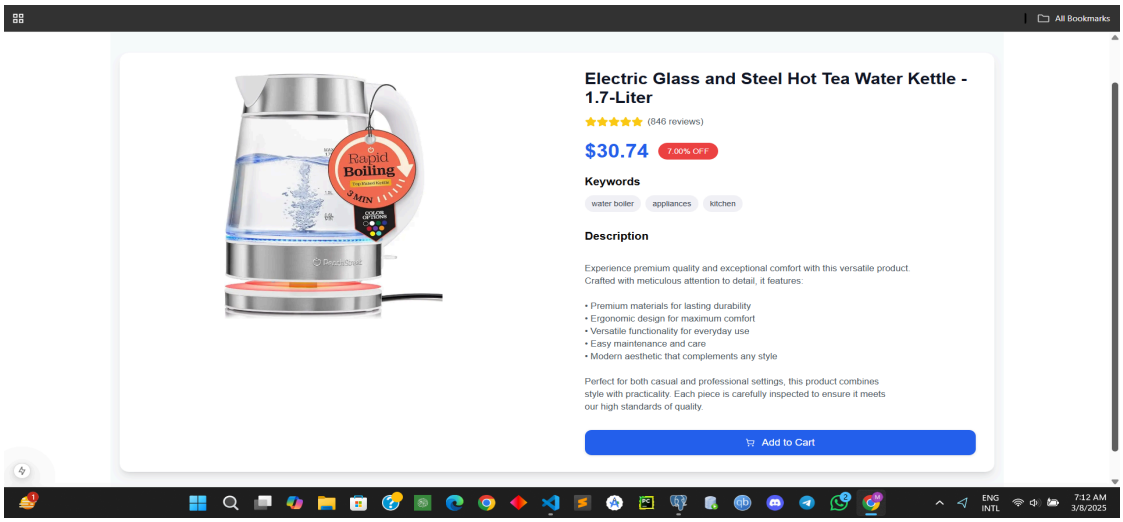




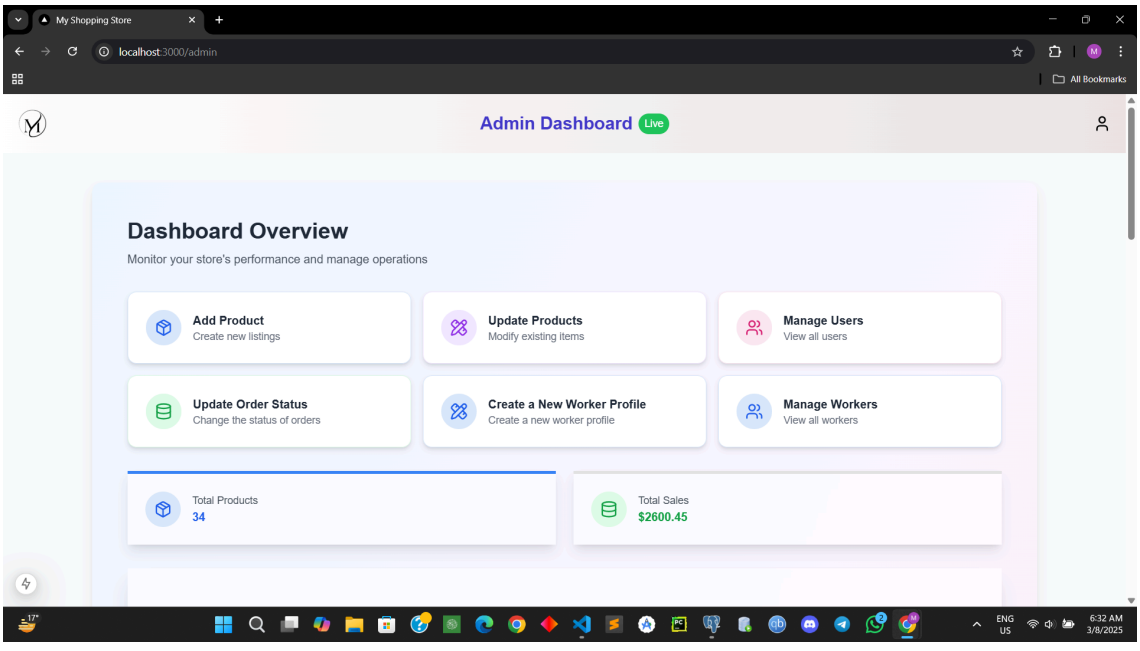

My Shopping Store
localhost:3000
All Bookmarks
Why Shop With Us
Free Shipping
Enjoy free shipping on all orders over $50
Quality Guarantee
We stand behind every product we sell
Easy Returns
30-day hassle-free return policy
Join Our Newsletter
Subscribe to get special offers, free giveaways, and once-in-a-lifetime deals.
Your email address
Subscribe


My Shopping Store
localhost:3000/Home
All Bookmarks
HOME
COLLECTION
ABOUT
CONTACT
Filters
Price Range
$0
$300
Categories
socks
basketballs
apparel
tshirts
sports
bathroom
mens
hoodies
sweaters
kitchen
cleaning
swimming
robe
swimsuit
accessories
Camera
DSLR
Photo
Search products...
Sort by
17.00% OFF
Breville Barista Express Espresso Machine. One Size. Brushed Stainless...
(20)
$249.00
kitchen
accessories
Add to Cart
10.00% OFF
Waterproof Knit Athletic Sneakers - Gray
(89)
$33.90
shoes
running shoes
footwear
Add to Cart
18.00% OFF
2 Slot Toaster - Black
(2197)
$18.99
toaster
kitchen
appliances
Add to Cart

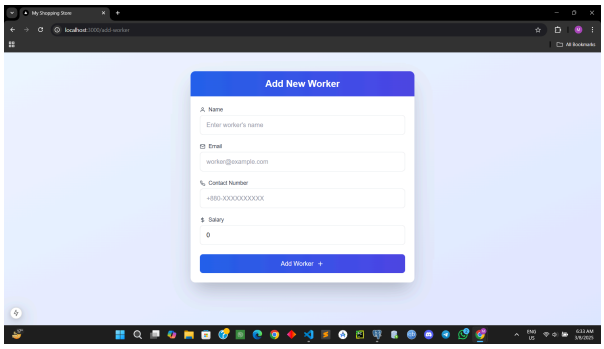

My Shopping Store
localhost:3000/Home
All Bookmarks
HOME
COLLECTION
ABOUT
CONTACT
Filters
Price Range
$0
$300
Categories
socks
basketballs
apparel
tshirts
sports
bathroom
mens
hoodies
sweaters
kitchen
cleaning
swimming
robe
swimsuit
accessories
Camera
DSLR
Photo
Search products...
Sort by
Price: Low to High
Price: High to Low
Highest Rating
10.00% OFF
Black and Gray Athletic Cotton Socks - 6 Pairs
(87)
$10.90
socks
sports
apparel
Add to Cart
8.00% OFF
Intermediate Size Basketball
(127)
$20.95
sports
basketballs
Add to Cart

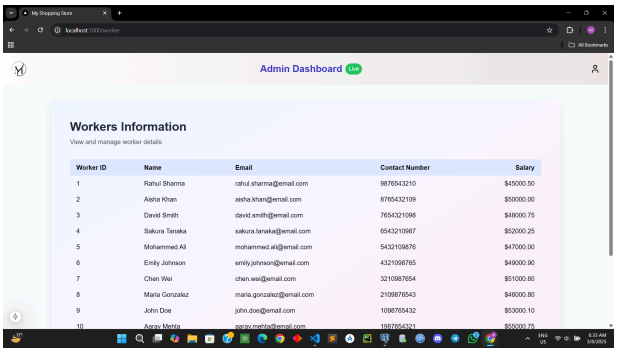

All Bookmarks
Rapid Boiling
Electric Glass and Steel Hot Tea Water Kettle - 1.7-Liter
(846 reviews)
$30.74
7.00% OFF
Keywords
water boiler
appliances
kitchen
Description
Experience premium quality and exceptional comfort with this versatile product. Crafted with meticulous attention to detail, it features:
• Premium materials for lasting durability
• Ergonomic design for maximum comfort
• Versatile functionality for everyday use
• Easy maintenance and care
• Modern aesthetic that complements any style
Perfect for both casual and professional settings, this product combines style with practicality. Each piece is carefully inspected to ensure it meets our high standards of quality.
Add to Cart
ENG INTL
7:12 AM 3/8/2025

Admin Panel

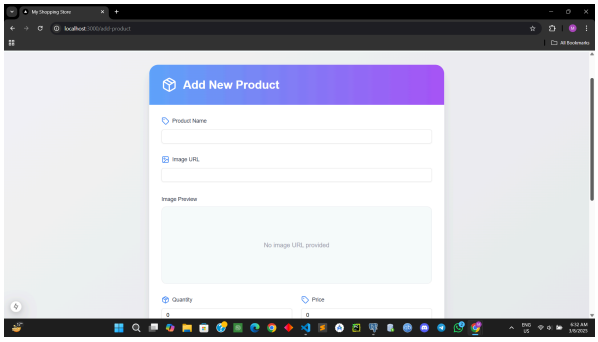

My Shopping Store
localhost:3000/admin
All Bookmarks
Admin Dashboard
Live
Dashboard Overview
Monitor your store's performance and manage operations
Add Product
Create new listings
Update Products
Modify existing items
Manage Users
View all users
Update Order Status
Change the status of orders
Create a New Worker Profile
Create a new worker profile
Manage Workers
View all workers
Total Products
34
Total Sales
$2600.45
ENG US
6:32 AM 3/8/2025

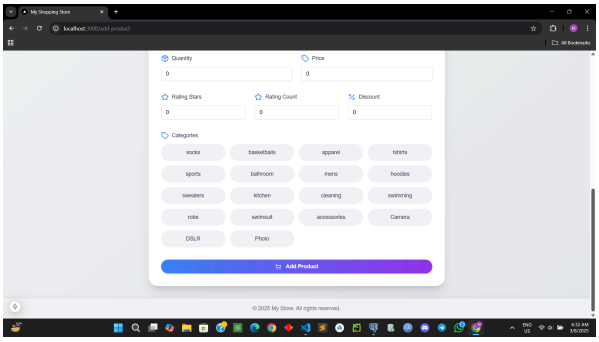
Add worker

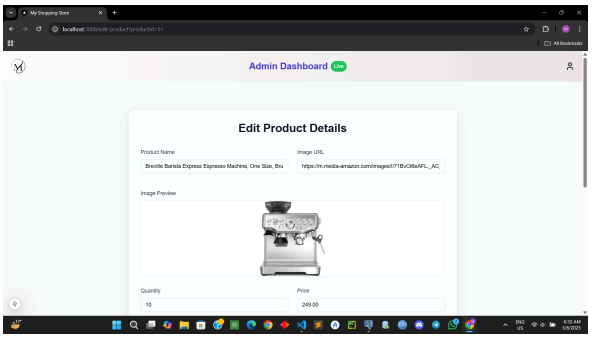


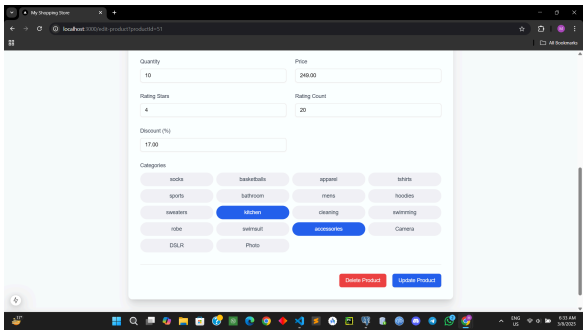


Add and Edit Product (Delete and Update)

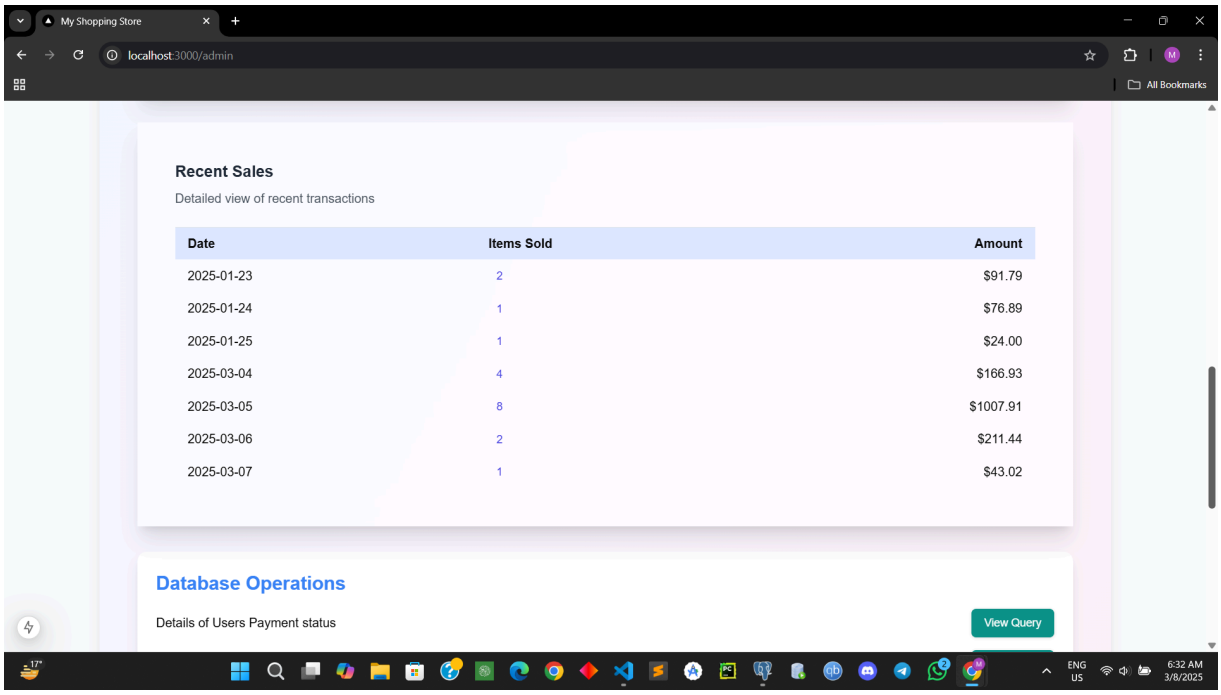


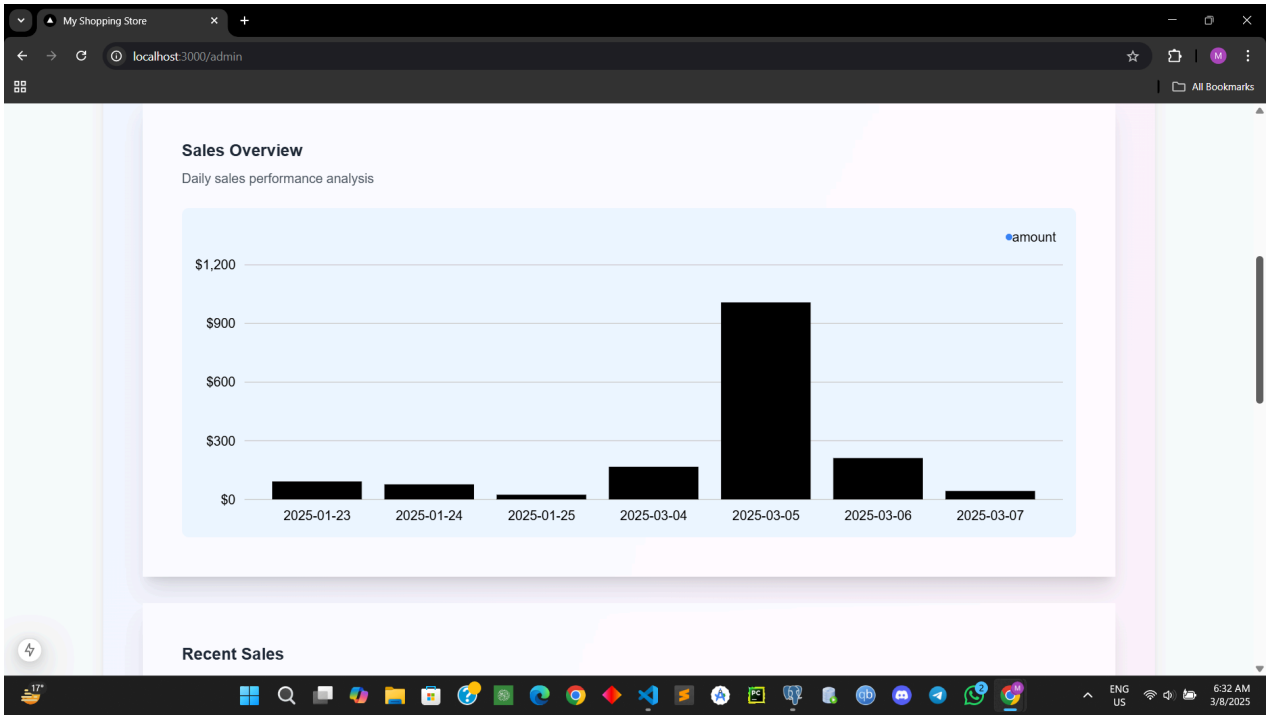




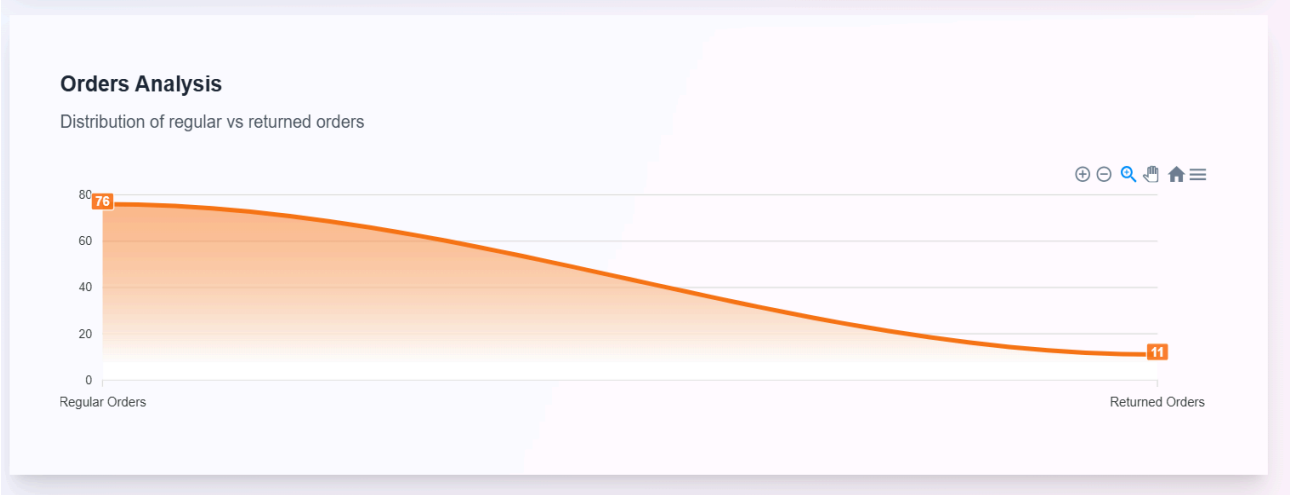


Overview of Recent Sales in graph form per day

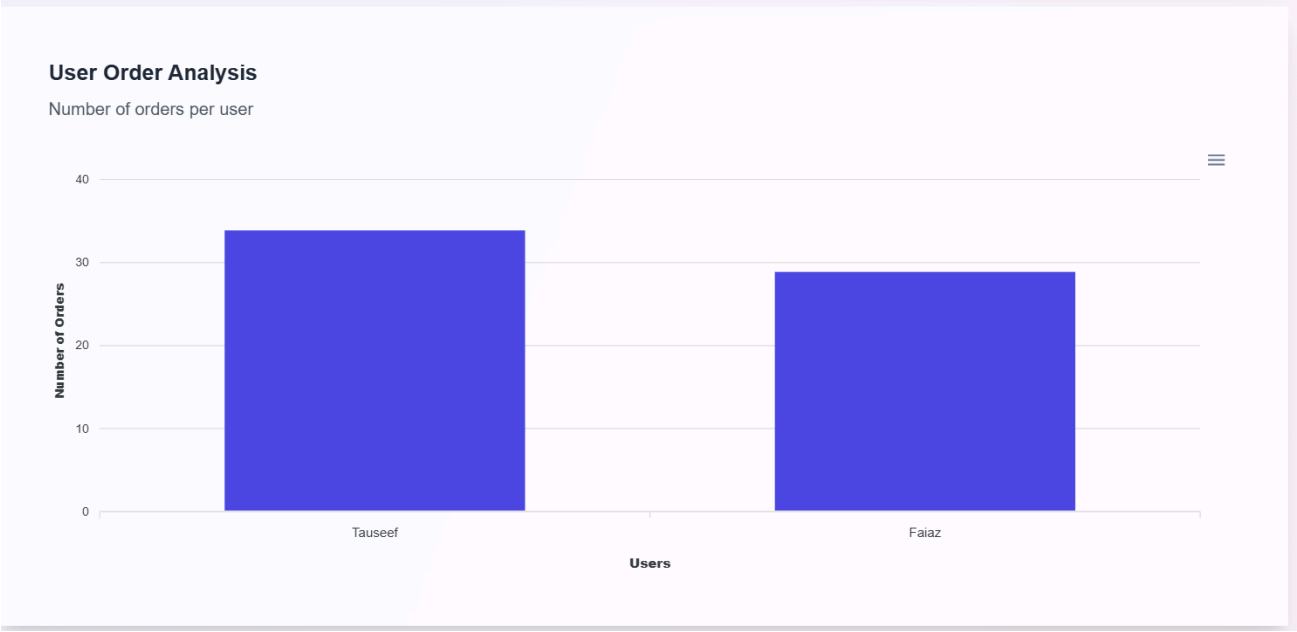




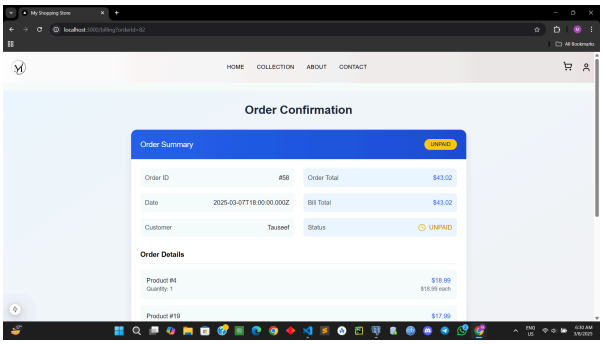
Distribution of regular vs returned orders

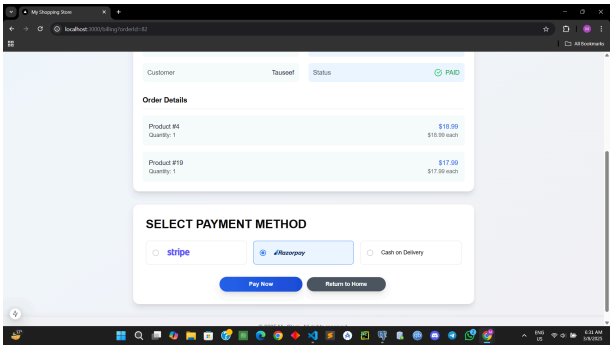


Overview of no of orders per user

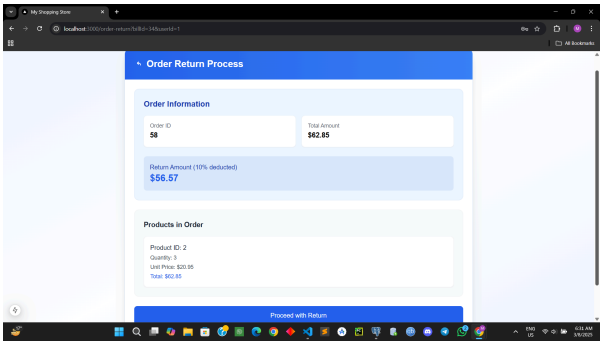


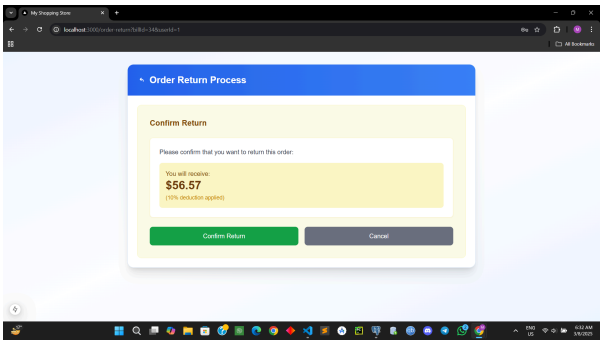
Order Confirmation

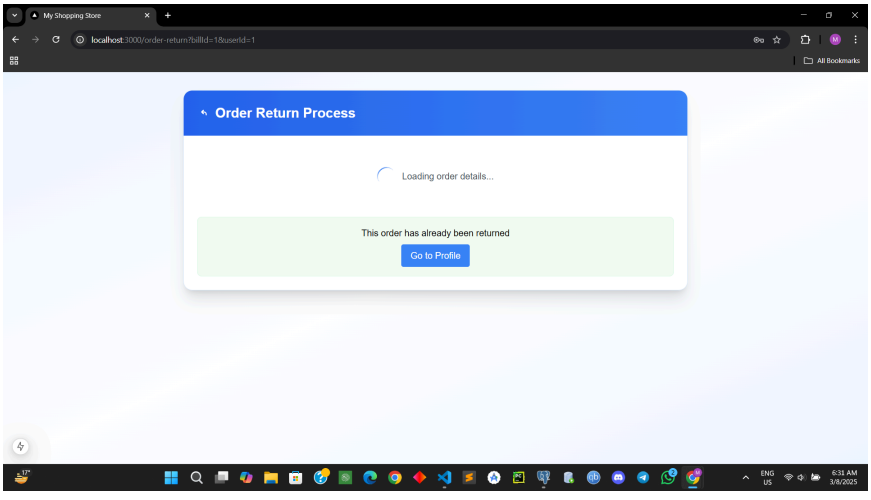




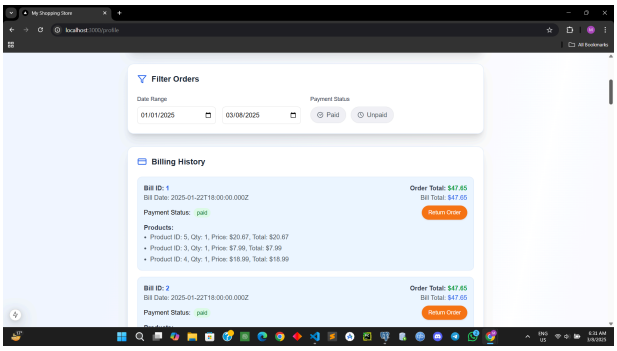
Order Return Process

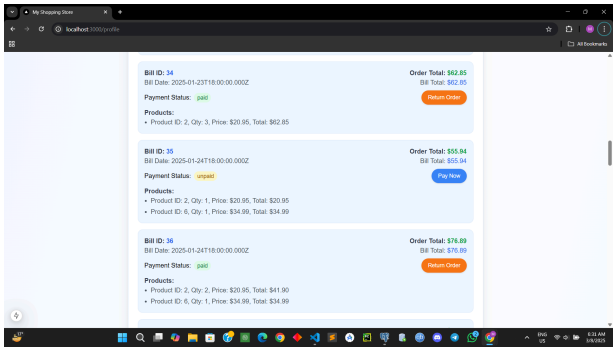




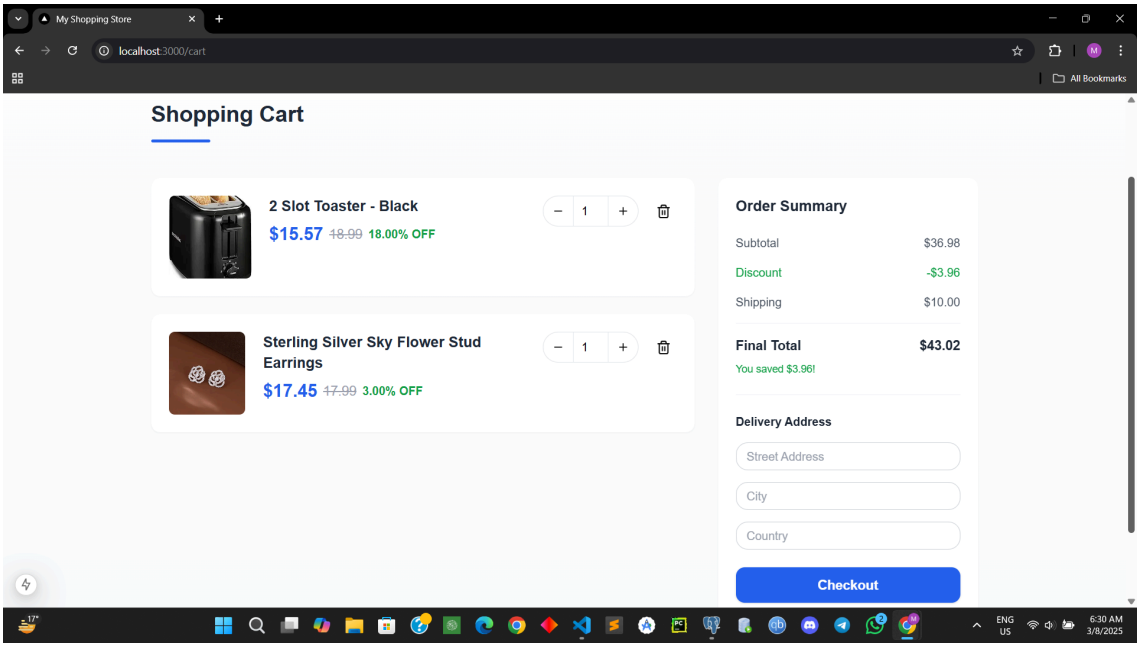


Bill Details

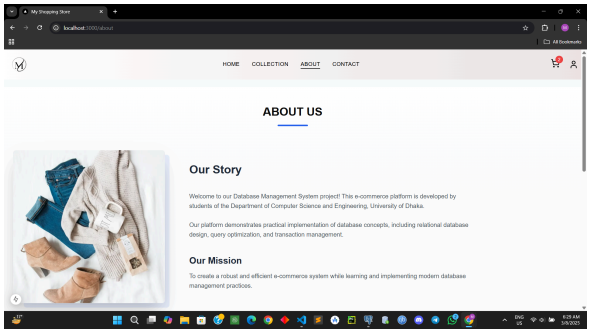


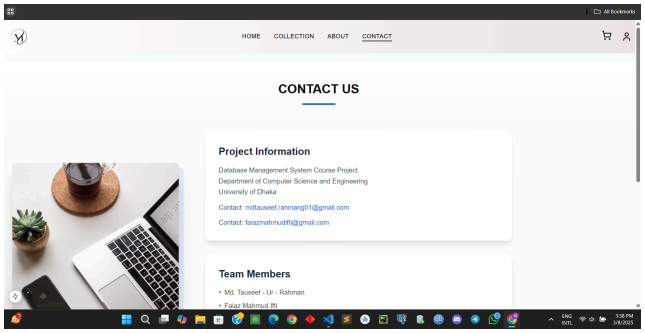


Shopping Cart page



About Us and Contact Us pages





10. List of non-trivial FDs:

Users Table :

- user_id → (user_name, user_email, user_password, user_contact_no, is_admin)
- user_email → (user_id, user_name, user_password, user_contact_no, is_admin)

Product Table :

- prod_id → (prod_name, prod_image, prod_quantity, prod_price, rating_stars, rating_count, prod_discount, prod_keywords)

Orders Table :

- order_id → (order_date, user_id, delivery_address, total_amt, order_status)

Shopping_cart Table :

- cart_id → (prod_id, user_id)

Order_details Table :

- (order_id, prod_id) → (prod_qty, prod_price, prod_total_price)

Bill_details Table :

- bill_id → (bill_date, user_id, order_id, user_name, order_total_price, bill_total_price, pay_status)
- user_id → user_name
- order_id → (user_id , user_name, order_total_price)

Order_return Table :

- order_return_id → (order_id, user_id, return_date, prod_id, return_amount)
- order_id → user_id

Worker_Table :

- worker_id → (worker_name, worker_email, worker_contact_no, worker_salary)
- worker_email → (worker_id, worker_name, worker_contact_no, worker_salary)

11. proof that the schemas are in desired normal forms

Conditions for 1NF:
All the schemas should -

- Have primary keys
- Have no multi-valued attributes (i.e. all columns have atomic values)

All schemas in our database have primary keys and no multi-valued attribute. So all are in 1NF.

Conditions for 2NF :

1. Table must be in 1NF.
2. Table must not contain any partial dependencies i.e. all non-prime attributes must be fully dependent on the candidate key as a whole

Conditions for 3NF :

1. Table must be in 2NF.
2. Table must not contain any transitive dependencies i.e. all non-prime attributes must depend on primary key (or prime attributes)

Conditions for BCNF :

1. Table must be in **3NF**.
2. **For every functional dependency (FD) X → Y, X must be a superkey** (i.e., X must be either a candidate key or a superset of a candidate key).

Users table :

2NF check :

- since user_id is the primary key and determines all attributes so it is in 2NF.

3NF check:

- The candidate keys are {user_id} and {user_email}
- Every **non-prime attribute** (user_name, user_email, etc.) **depends only on the primary key** (user_id), so no transitive dependency exists..

BCNF check:

- user_id is a **candidate key**.
- user_email is **also a candidate key**.
- Since all FDs have **only candidate keys on the left side**, **BCNF holds**.

Product table :

2NF check :

- No partial dependencies since prod_id is the primary key so it is in 2NF.

3NF check:

- All **non-prime attributes** depend **only on prod_id**. So No transitive dependencies exist.

BCNF check:

- prod_id is a **candidate key**.
- All attributes depend **directly on prod_id**.
- No violations of BCNF rule.

Worker table :

2NF check :

- No partial dependencies since worker_id is the primary key so it is in 2NF.

3NF check:

- Every **non-prime attribute** depends **only on the candidate keys** (worker_id or worker_email). So no transitive dependencies exist on this table.

BCNF check :

- Both worker_id and worker_email are **candidate keys**.
- Since all FDs have only candidate keys on the left, no violations.

Shopping_cart table :

2NF check :

- No partial dependencies since cart_id is the primary key.

3NF check :

- The non-prime attributes are prod_id and user_id, both of which are foreign keys.
- These attributes **directly depend on the primary key (cart_id)**.

BCNF check :

- The two candidate keys are: **cart_id** and **(user_id, prod_id)** (an user can only add a product once)
- The functional dependency **cart_id → prod_id, user_id** is fine because cart_id is a candidate key (superkey).
- The functional dependency **(user_id, prod_id) → cart_id** is also fine because (user_id, prod_id) is a candidate key (superkey).

Orders table :

2NF check :

- No partial dependencies since order_id is the primary key

3NF check :

- The non-prime attributes **directly depend on the primary key** (**order_id**).

BCNF check :

- order_id is a **candidate key**.
- No other attribute determines order_id.
- All FDs have order_id (a superkey) on the left side.

Order_details table :

2NF check:

- The composite key (order_id, prod_id) uniquely determines all other attributes.

3NF check :

- Every **non-prime attribute** depends **only on the composite primary key** {order_id, prod_id}.

BCNF check :

- {order_id, prod_id} is the **candidate key**.
- All non-key attributes depend **fully** on {order_id, prod_id}.
- No violations of BCNF.

Order_return table :

2NF check :

- No partial dependencies since order_return_id is the primary key.

3NF check :

- All **non-prime attributes** depend **only on** order_return_id. So No transitive dependencies exist.

BCNF check :

- order_return_id is a **candidate key**.
- No other dependencies exist that violate BCNF.

Bill_details table :

2NF check :

- No partial dependencies since bill_id is the primary key

3NF check :

- **Issue:** user_name depends on user_id, **not directly on bill_id**.
- Since user_name is a **non-prime attribute** and depends on another **non-prime attribute** (user_id), we have a **transitive dependency**.

BCNF check :

- The first FD is fine (bill_id is a candidate key).
- The second FD suggests order_id determines non-key attributes **but order_id is not a superkey**

So, in our database

- ➢ **all schemas are in second normal form.**
- ➢ **except bill_details schema , all the rest seven schemas are in 3NF as well as in BCNF**

12. Front End designing tools and techniques, advantages and disadvantages of the implementation

Here's a detailed breakdown of the front-end designing tools and techniques we've used, along with their advantages and disadvantages:

Front-End Designing Tools and Techniques

1. Next.js 14 (React)

- **Technique**: A React-based framework for server-side rendering (SSR), static site generation (SSG), and client-side rendering (CSR).
- **Advantages**:
    - Improved performance with SSR and SSG.
    - Built-in routing and API routes simplify development.
    - Automatic code splitting for faster page loads.
    - SEO-friendly due to server-side rendering.
    - Supports TypeScript out of the box.
- **Disadvantages**:
    - The steeper learning curve for beginners compared to plain React.
    - Overhead for small projects (might be overkill).
    - Requires understanding of server-side concepts.

2. TypeScript

- **Technique**: A statically typed superset of JavaScript.
- **Advantages**:
    - Catches errors at compile time, reducing runtime bugs.
    - Improves code readability and maintainability.
    - Enhances developer productivity with better IDE support (e.g., autocompletion).
    - Easier to refactor code.
- **Disadvantages**:
    - Adds complexity to the development process.
    - Requires additional setup and configuration.
    - Can be overkill for small projects.

3. Tailwind CSS

- **Technique**: A utility-first CSS framework.
- **Advantages**:
    - Rapid prototyping with pre-defined utility classes.
    - Highly customizable with a tailwind.config.js file.
    - Reduces the need for writing custom CSS.
    - Encourages consistency in design.
- **Disadvantages**:
    - Can lead to cluttered HTML with many utility classes.
    - The steeper learning curve for developers unfamiliar with utility-first CSS.
    - Larger CSS file size if not purged properly.

4. Tremor (for Charts and Analytics)

- **Technique**: A library for building charts and analytics dashboards.
- **Advantages**:
    - Easy to integrate with React and Next.js.
    - Pre-built components for common chart types (e.g., bar, line, pie).
    - Customizable and responsive.
- **Disadvantages**:
    - Limited compared to more mature libraries like D3.js or Chart.js.
    - May not support advanced charting requirements.

5. Lucid Icons

- **Technique**: A lightweight icon library.
- **Advantages**:
    - Easy to use and integrate.
    - Highly customizable (size, color, etc.).
    - Lightweight and performant.
- **Disadvantages**:
    - Limited icon set compared to larger libraries like FontAwesome.
    - Requires manual updates for new icons.

13. Conclusion of the work

Working on this **Shop Management System** project has been a great learning experience. Throughout the project, we faced many challenges, such as designing a well-structured database, writing efficient queries, and ensuring smooth connections between different parts of the system.

We learned how to manage user data, handle orders and payments, and update product inventory. We also improved our skills in **database design, query optimization, and problem-solving**.

This project helped us understand how real-world systems work and how important databases are in managing business operations. Overall, it was a rewarding journey that gave us valuable knowledge for future projects.