E-Commerce Order Management Database System

1. Introduction

This project focuses on building a complete E-Commerce Database Management System using PostgreSQL.
The goal is to design a well-structured relational database, insert real-world data, and demonstrate industry-level SQL skills including:

- Database design
- ER modeling
- SQL CRUD operations
- Joins & subqueries
- Advanced SQL (views, transactions, functions, triggers, procedures)
- Performance tuning

This project simulates how real online shopping platforms (Amazon, Flipkart etc) manage customers, orders, payments, and inventory.

2. Problem Statement

Online e-commerce systems handle thousands of customers, products, orders, and payments daily.
A well-designed database is required to:

- Store customer and product details
- Track orders and order items
- Process payments
- Maintain stock levels
- Prevent invalid operations
- Generate meaningful reports

This project solves the above requirements using PostgreSQL.

3. Technologies Used

- PostgreSQL 15 / pgAdmin
- PL/pgSQL for triggers, functions & stored procedures
- ERD tool: dbdiagram.io
- SQL concepts: DDL, DML, Joins, Views, Transactions, Functions, Procedures, Triggers, Indexing

Table of Contents

4. Database Design

5. Database Schema

6. Data Insertion

7. Indexes

8. Basic SQL Operations

9. Joins

10. Aggregate Functions

11. Sub queries

12. Advanced Joins

13. Views

14. Analytical Queries

15. Transactions

16. Stored Procedure

17. Functions

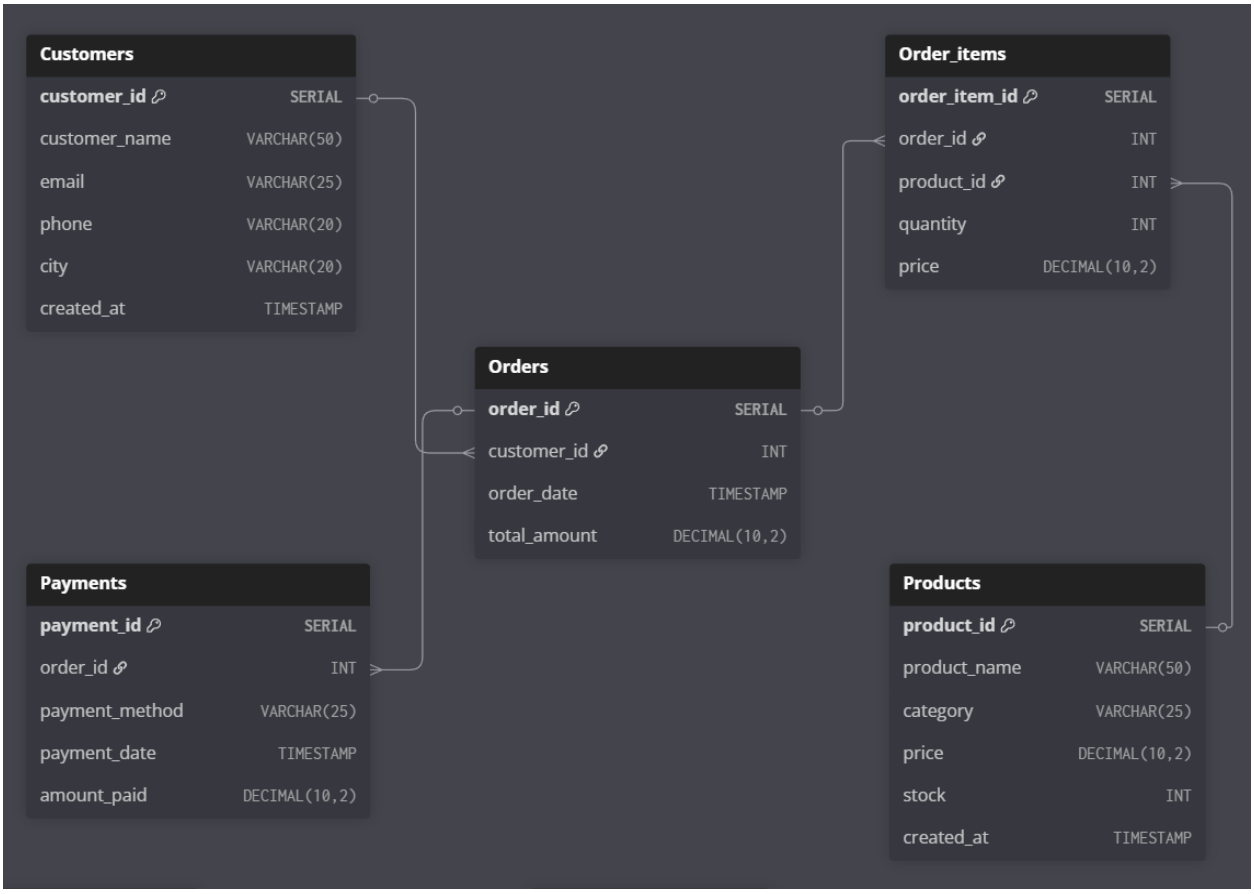
18. Triggers

19. Conclusion

20. Author Details

4. Database Design

4.1 Entity-Relationship Diagram



The ERD consists of five main entities:

1. Customers

Stores customer details.
Primary Key: customer_id
Used in the Orders table as FK.

2. Products

Contains product catalog information such as price and stock.
Primary Key: product_id
Mapped to Order_Items.

3. Orders

Each customer order is stored here.
Primary Key: order_id
Foreign Key: customer_id → Customers

4. Order_Items

Stores each product bought in an order.
Foreign Keys: order_id → Orders, product_id → Products
This creates a many-to-many relationship.

5. Payments

Stores payment details for each order.
Foreign Key: order_id → Orders
Relationship: 1 order = 1 payment

This structure supports scalability, consistency & real-world behavior.

5. Database Schema

5.1 Customers Table

```sql
CREATE TABLE Customers (
    customer_id SERIAL PRIMARY KEY,
    customer_name VARCHAR(50) NOT NULL,
    email VARCHAR(50) UNIQUE NOT NULL,
    phone VARCHAR(20),
    city VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

5.2 Products Table

```sql
CREATE TABLE Products (
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(50) NOT NULL,
    category VARCHAR(25),
    price DECIMAL(10,2) NOT NULL,
    stock INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

5.3 Orders Table

```sql
CREATE TABLE Orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INT NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10,2),

    FOREIGN KEY (customer_id)
        REFERENCES customers(customer_id)
        ON DELETE CASCADE
);
```

5.4 Order Items Table

```sql
CREATE TABLE Order_items (
    order_item_id SERIAL PRIMARY KEY,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,

    FOREIGN KEY (order_id)
        REFERENCES orders(order_id)
        ON DELETE CASCADE,

    FOREIGN KEY (product_id)
        REFERENCES products(product_id),

    UNIQUE(order_id, product_id)
);
```

5.5 Payments Table

```sql
CREATE TABLE Payments (
    payment_id SERIAL PRIMARY KEY,
    order_id INT UNIQUE NOT NULL,
    payment_method VARCHAR(25),
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    amount_paid DECIMAL(10,2),

    FOREIGN KEY (order_id)
        REFERENCES orders(order_id)
        ON DELETE CASCADE
);
```

6. Data Insertion

The dataset used in this project consists of randomly generated sample records for customers, products, orders, order_items and payments. These values were created purely for educational and demonstration purposes

6.1 Customers Table (30 customers)

```
SELECT * FROM Customers LIMIT 10;
```

customer_id [PK] integer	customer_name character varying (50)	email character varying (50)	phone character varying (20)	city character varying (20)	created_at timestamp without time zone
1	Amit Sharma	amit@example.com	9876543210	Delhi	2025-11-20 16:32:25.983397
2	Sneha Nair	sneha@example.com	9123456780	Kochi	2025-11-20 16:32:25.983397
3	Ravi Kumar	ravi@example.com	9988776655	Bangalore	2025-11-20 16:32:25.983397
4	Priya Singh	priya@example.com	9090909090	Mumbai	2025-11-20 16:32:25.983397
5	John Mathew	john@example.com	9876501234	Pune	2025-11-20 16:32:25.983397
6	Rahul Das	rahul@example.com	9811122233	Kolkata	2025-11-20 16:32:25.983397
7	Sara Thomas	sara@example.com	9800011112	Chennai	2025-11-20 16:32:25.983397
8	David Joseph	david@example.com	9900088887	Hyderabad	2025-11-20 16:32:25.983397
9	Kevin George	kevin@example.com	9812345678	Delhi	2025-11-20 16:32:25.983397
10	Latha Krishnan	latha@example.com	9900011122	Bangalore	2025-11-20 16:32:25.983397

6.2 Products Table (20 products)

```
SELECT * FROM Products LIMIT 10;
```

product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	Legion 5 Laptop	Electronics	95000.00	20	2025-11-20 16:33:40.481077
2	iPhone 14	Electronics	120000.00	15	2025-11-20 16:33:40.481077
3	Samsung Monitor 27"	Electronics	23000.00	18	2025-11-20 16:33:40.481077
4	Wireless Mouse	Accessories	1200.00	80	2025-11-20 16:33:40.481077
5	Mechanical Keyboard	Accessories	4500.00	50	2025-11-20 16:33:40.481077
6	Boat Headphones	Accessories	1800.00	60	2025-11-20 16:33:40.481077
7	Logitech Webcam	Electronics	5000.00	25	2025-11-20 16:33:40.481077
8	Gaming Chair	Furniture	15000.00	10	2025-11-20 16:33:40.481077
9	Office Desk	Furniture	20000.00	12	2025-11-20 16:33:40.481077
10	Water Bottle	Home	300.00	150	2025-11-20 16:33:40.481077

6.3 Orders Table (80 orders)

```
SELECT * FROM Orders LIMIT 10;
```

order_id [PK] integer	customer_id integer	order_date timestamp without time zone	total_amount numeric (10,2)
1	24	2025-11-20 16:34:00.171934	117262.08
2	8	2025-11-20 16:34:00.171934	53220.00
3	2	2025-11-20 16:34:00.171934	41150.19
4	22	2025-11-20 16:34:00.171934	56045.76
5	6	2025-11-20 16:34:00.171934	37116.53
6	12	2025-11-20 16:34:00.171934	93393.03
7	8	2025-11-20 16:34:00.171934	116146.67
8	16	2025-11-20 16:34:00.171934	33322.42
9	7	2025-11-20 16:34:00.171934	89055.63
10	9	2025-11-20 16:34:00.171934	9023.10

6.4 Order items Table (200 order items)

```
SELECT * FROM Order_items LIMIT 10;
```

order_item_id [PK] integer	order_id integer	product_id integer	quantity integer	price numeric (10,2)
112	1	5	3	47374.75
113	1	7	4	10270.48
114	1	8	1	583.66
115	1	9	5	15992.36
116	1	11	3	26158.10
117	1	13	4	13514.24
118	1	18	5	2455.34
119	2	4	5	25506.73
120	2	10	4	46147.53
121	2	11	4	9303.57

6.5 Payments Table (80 Payments)

```
SELECT * FROM Payments LIMIT 10;
```

payment_id [PK] integer	order_id integer	payment_method character varying (25)	payment_date timestamp without time zone	amount_paid numeric (10,2)
1	1	Credit Card	2025-11-20 16:38:17.192372	117262.08
2	2	Credit Card	2025-11-20 16:38:17.192372	53220.00
3	3	Net Banking	2025-11-20 16:38:17.192372	41150.19
4	4	Net Banking	2025-11-20 16:38:17.192372	56045.76
5	5	Net Banking	2025-11-20 16:38:17.192372	37116.53
6	6	Credit Card	2025-11-20 16:38:17.192372	93393.03
7	7	UPI	2025-11-20 16:38:17.192372	116146.67
8	8	Credit Card	2025-11-20 16:38:17.192372	33322.42
9	9	Cash	2025-11-20 16:38:17.192372	89055.63
10	10	Cash	2025-11-20 16:38:17.192372	9023.10

7. Indexes

```
CREATE INDEX idx_orders_customer_id
ON Orders(customer_id);

CREATE INDEX idx_order_items_order_id
ON order_items(order_id);

CREATE INDEX idx_order_items_product_id
ON order_items(product_id);

CREATE INDEX idx_orders_order_date
ON orders(order_date);

CREATE INDEX idx_products_name
ON products(product_name);
```

8. Basic SQL Operations

```
SELECT * FROM Customers WHERE city = 'Kochi';
```

	customer_id [PK] integer	customer_name character varying (50)	email character varying (50)	phone character varying (20)	city character varying (20)	created_at timestamp without time zone
1	2	Sneha Nair	sneha@example.com	9123456780	Kochi	2025-11-20 16:32:25.983397
2	14	Anu Ramesh	anu@example.com	9833344455	Kochi	2025-11-20 16:32:25.983397
3	24	Meera S	meera@example.com	9877701122	Kochi	2025-11-20 16:32:25.983397

```
SELECT * FROM Products WHERE price > 10000 ORDER BY price DESC;
```

	product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	2	iPhone 14	Electronics	120000.00	15	2025-11-20 16:33:40.481077
2	1	Legion 5 Laptop	Electronics	95000.00	20	2025-11-20 16:33:40.481077
3	14	Projector X2	Electronics	35000.00	8	2025-11-20 16:33:40.481077
4	3	Samsung Monitor 27"	Electronics	23000.00	18	2025-11-20 16:33:40.481077
5	9	Office Desk	Furniture	20000.00	12	2025-11-20 16:33:40.481077
6	13	Standing Desk	Furniture	18000.00	15	2025-11-20 16:33:40.481077
7	8	Gaming Chair	Furniture	15000.00	10	2025-11-20 16:33:40.481077

```
SELECT * FROM Customers WHERE customer_name ILIKE 'A%';
```

	customer_id [PK] integer	customer_name character varying (50)	email character varying (50)	phone character varying (20)	city character varying (20)	created_at timestamp without time zone
1	1	Amit Sharma	amit@example.com	9876543210	Delhi	2025-11-20 16:32:25.983397
2	14	Anu Ramesh	anu@example.com	9833344455	Kochi	2025-11-20 16:32:25.983397
3	17	Aswin K	aswin@example.com	9003344556	Chennai	2025-11-20 16:32:25.983397
4	19	Aarav Sen	aarav@example.com	9556677889	Delhi	2025-11-20 16:32:25.983397
5	26	Ajay G	ajay@example.com	9112233445	Mumbai	2025-11-20 16:32:25.983397

```
SELECT DISTINCT city FROM customers;
```

	city character varying (20)
1	Kochi
2	Mumbai
3	Pune
4	Delhi
5	Chennai
6	Hyderabad
7	Bangalore
8	Kolkata

```
SELECT * FROM Products WHERE stock IS NOT NULL LIMIT 5;
```

	product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	1	Legion 5 Laptop	Electronics	95000.00	20	2025-11-20 16:33:40.481077
2	2	iPhone 14	Electronics	120000.00	15	2025-11-20 16:33:40.481077
3	3	Samsung Monitor 27"	Electronics	23000.00	18	2025-11-20 16:33:40.481077
4	4	Wireless Mouse	Accessories	1200.00	80	2025-11-20 16:33:40.481077
5	6	Boat Headphones	Accessories	1800.00	60	2025-11-20 16:33:40.481077

9. Joins

```sql
--LEFT JOIN: Display Customers and Their Orders
SELECT c.customer_name, o.order_id, o.total_amount
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.order_id
ORDER BY total_amount DESC LIMIT 10;
```

	customer_name character varying (50)	order_id integer	total_amount numeric (10,2)
1	Kavya S	27	107311.51
2	Amit Sharma	1	105535.87
3	Sara Thomas	7	104532.00
4	Neeraj Kumar	23	96179.72
5	Aarav Sen	19	87325.60
6	Rahul Das	6	84053.73
7	Kevin George	9	80150.07
8	Deepak R	30	69929.00
9	Vishal Roy	22	59995.74
10	Priya Singh	4	50441.18

```sql
--INNER JOIN: Display Order Items With Product Details
SELECT oi.order_id, p.product_name, oi.quantity
FROM order_items oi
JOIN products p ON oi.product_id = p.product_id
ORDER BY quantity DESC LIMIT 10;
```

	order_id integer	product_name character varying (50)	quantity integer
1	39	Mechanical Keyboard	10
2	7	Boat Headphones	5
3	2	Wireless Mouse	5
4	5	USB-C Cable	5
5	4	Bluetooth Speaker	5
6	1	Portable SSD 1TB	5
7	3	Desk Lamp	5
8	1	Office Desk	5
9	5	Gaming Chair	5
10	6	Gaming Chair	5

```sql
--FULL JOIN: Display All Customers + All Orders
SELECT c.customer_name, o.order_id, o.total_amount
FROM Customers c
FULL JOIN orders o ON c.customer_id = o.customer_id LIMIT 10;
```

	customer_name character varying (50)	order_id integer	total_amount numeric (10,2)
1	Amit Sharma	81	5000.00
2	Sneha Nair	3	41150.19
3	Ravi Kumar	14	46517.11
4	Ravi Kumar	28	49901.00
5	Ravi Kumar	73	41381.36
6	Priya Singh	24	503.06
7	John Mathew	37	107941.86
8	Rahul Das	5	33404.88
9	Rahul Das	31	72325.17
10	Rahul Das	33	101881.69

10. Aggregate Functions

```sql
--Count products by category
SELECT category, COUNT(*) AS total_products
FROM products GROUP BY category;
```

	category character varying (25)	total_products bigint
1	Furniture	3
2	Electronics	7
3	Accessories	6
4	Home	2
5	Stationery	2

```sql
--Category with avg price above 10000
SELECT category, AVG(price) FROM products
GROUP BY category HAVING AVG(price) > 10000;
```

	category character varying (25)	avg numeric
1	Furniture	17666.666666666667
2	Electronics	42142.857142857143

```sql
--Total revenue from all sales
SELECT SUM(price * quantity) AS Total_revenue
FROM order_items;
```

	total_revenue numeric
1	12748043.55

```sql
--Cheapest and costliest products
SELECT MIN(price) AS cheapest, MAX(price) AS most_expensive
FROM products;
```

	cheapest numeric	most_expensive numeric
1	120.00	120000.00

```sql
--Orders per city
SELECT c.city, COUNT(o.order_id) AS total_orders
FROM Customers c LEFT JOIN orders o
ON c.customer_id = o.customer_id
GROUP BY c.city ORDER BY total_orders DESC;
```

	city character varying (20)	total_orders bigint
1	Kolkata	14
2	Bangalore	13
3	Chennai	12
4	Mumbai	12
5	Delhi	11
6	Hyderabad	8
7	Pune	6
8	Kochi	5

11. Sub queries

```
--Products above Average price
SELECT * FROM Products WHERE price > (SELECT AVG(price) FROM Products);
```

	product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	1	Legion 5 Laptop	Electronics	95000.00	20	2025-11-20 16:33:40.481077
2	2	iPhone 14	Electronics	120000.00	15	2025-11-20 16:33:40.481077
3	3	Samsung Monitor 27"	Electronics	23000.00	18	2025-11-20 16:33:40.481077
4	9	Office Desk	Furniture	20000.00	12	2025-11-20 16:33:40.481077
5	14	Projector X2	Electronics	35000.00	8	2025-11-20 16:33:40.481077

```
--Customers who never ordered
SELECT * FROM Customers
WHERE customer_id NOT IN (SELECT customer_id FROM Orders);
```

	customer_id [PK] integer	customer_name character varying (50)	email character varying (50)	phone character varying (20)	city character varying (20)	created_at timestamp without time zone
1	17	Aswin K	aswin@example.com	9003344556	Chennai	2025-11-20 16:32:25.983397

```
--Product with avg price comparison
SELECT product_name, price, (SELECT AVG(price) FROM Products) AS avg_price
FROM Products;
```

	product_name character varying (50)	price numeric (10,2)	avg_price numeric
1	Legion 5 Laptop	95000.00	18063.500000000000
2	iPhone 14	120000.00	18063.500000000000
3	Samsung Monitor 27"	23000.00	18063.500000000000
4	Wireless Mouse	1200.00	18063.500000000000
5	Boat Headphones	1800.00	18063.500000000000
6	Logitech Webcam	5000.00	18063.500000000000
7	Gaming Chair	15000.00	18063.500000000000
8	Office Desk	20000.00	18063.500000000000
9	Water Bottle	300.00	18063.500000000000
10	Notebook Set	150.00	18063.500000000000

```
-- 10% Discount to all those who spent more than 50000
UPDATE Orders SET total_amount = total_amount * 0.90
WHERE customer_id IN (SELECT customer_id FROM Orders
                      GROUP BY customer_id HAVING SUM(total_amount) > 50000);
```

```
UPDATE 75

Query returned successfully in 161 msec.
```

12. Advanced Joins

```sql
--CROSS JOIN: each product with each Category
SELECT p.product_name, c.category FROM products p
CROSS JOIN (SELECT DISTINCT category FROM products) c;
```

	product_name character varying (50)	category character varying (25)
1	Legion 5 Laptop	Furniture
2	Legion 5 Laptop	Electronics
3	Legion 5 Laptop	Accessories
4	Legion 5 Laptop	Home
5	Legion 5 Laptop	Stationery
6	iPhone 14	Furniture
7	iPhone 14	Electronics
8	iPhone 14	Accessories
9	iPhone 14	Home
10	iPhone 14	Stationery
11	Samsung Monitor 27"	Furniture
12	Samsung Monitor 27"	Electronics

```sql
--UNION ALL: Categorise with "High" and "Low" with total_amount as 50000
SELECT order_id, customer_id, total_amount, 'High' AS Spend
FROM Orders WHERE total_amount > 50000
UNION ALL
SELECT order_id, customer_id,total_amount, 'Low' AS Spend
FROM Orders WHERE total_amount < 50000
ORDER BY order_id;
```

	order_id integer	customer_id integer	total_amount numeric (10,2)	spend text
1	1	24	94982.28	High
2	2	8	43108.20	Low
3	3	2	41150.19	Low
4	4	22	45397.06	Low
5	5	6	30064.39	Low
6	6	12	75648.36	High
7	7	8	94078.80	High
8	8	16	26991.16	Low
9	9	7	72135.06	High
10	10	9	9023.10	Low

13. Views

```sql
-- High value orders
CREATE VIEW high_value_orders AS
SELECT order_id, customer_id, total_amount FROM Orders
WHERE total_amount > 50000 ORDER BY total_amount DESC;

SELECT * FROM high_value_orders LIMIT 5;
```

	order_id integer	customer_id integer	total_amount numeric (10,2)
1	37	5	97147.67
2	58	22	96838.95
3	27	11	96580.36
4	1	24	94982.28
5	7	8	94078.80

14. Analytical Queries

```sql
-- Product Category wise Sales
SELECT p.category, SUM(oi.quantity * oi.price ) AS total_sales
FROM Products p JOIN order_items oi
ON p.product_id = oi.product_id
GROUP BY p.category ORDER BY total_sales DESC;
```

	category character varying (25)	total_sales numeric
1	Accessories	4306414.41
2	Electronics	4149085.62
3	Furniture	1669532.73
4	Home	1404208.67
5	Stationery	1218802.12

```sql
-- Top 5 customers by spending
SELECT c.customer_name, SUM(o.total_amount) AS total_spent
FROM Customers c JOIN Orders o
ON c.customer_id = o.customer_id
GROUP BY c.customer_name ORDER BY total_spent DESC LIMIT 5;
```

	customer_name character varying (50)	total_spent numeric
1	Vishal Roy	373546.18
2	Rahul Das	347478.82
3	Sajid Ali	251547.75
4	Latha Krishnan	229676.81
5	David Joseph	229180.02

15. Transactions

```
--Safe order placement
BEGIN;
UPDATE products SET stock = stock - 2 WHERE product_id = 12;
INSERT INTO Orders (customer_id, order_date, total_amount)
VALUES (1, CURRENT_DATE, 5000);
SELECT * FROM products WHERE product_id = 12;
--no issues, stock not negative. Hence Commit
COMMIT;
```

16. Stored Procedure

```
--Stored Procedure to update stock
CREATE OR REPLACE PROCEDURE restock_product2 (p_product_id INT, p_quantity INT)
AS $$
BEGIN
    UPDATE products SET stock = stock + p_quantity
    WHERE product_id = p_product_id;
    COMMIT;
END;
$$ LANGUAGE plpgsql
SELECT * FROM products WHERE product_id = 8;
CALL restock_product2(8,10);
SELECT * FROM products WHERE product_id = 8;
```

	product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	8	Gaming Chair	Furniture	15000.00	10	2025-11-20 16:33:40.481077

```
CALL

Query returned successfully in 169 msec.
```

	product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	8	Gaming Chair	Furniture	15000.00	20	2025-11-20 16:33:40.481077

17. Functions

```sql
--Function to get total order price
CREATE OR REPLACE FUNCTION get_order_total(p_order_id INT)
RETURNS INT AS $$
DECLARE
    total_amt INT;
BEGIN
    SELECT SUM(quantity * price) INTO total_amt
    FROM order_items WHERE order_id = p_order_id;
    RETURN total_amt;
END;
$$ LANGUAGE plpgsql

SELECT get_order_total(15);
```

	get_order_total integer
1	322245

```sql
--Function to get customer total spending
CREATE OR REPLACE FUNCTION customer_total_spent(p_customer_id INT)
RETURNS INT AS $$
DECLARE
    total_amt INT;
BEGIN
    SELECT SUM(total_amount) INTO total_amt
    FROM Orders WHERE customer_id = p_customer_id;
    RETURN total_amt;
END;
$$ LANGUAGE plpgsql

SELECT customer_total_spent(25);
```

	customer_total_spent integer
1	62936

```sql
--Function to check the total revenue by a product
CREATE OR REPLACE FUNCTION total_by_product(p_product_name VARCHAR(50))
RETURNS INT AS $$
DECLARE
    total_amt INT;
BEGIN
    SELECT SUM(o.quantity * o.price) INTO total_amt
    FROM order_items o JOIN Products p ON p.product_id = o.product_id
    WHERE p.product_name ILIKE P_product_name;
    RETURN total_amt;
END;
$$ LANGUAGE plpgsql

SELECT total_by_product('Gaming Chair');
```

	total_by_product integer
1	360739

18. Triggers

```sql
--Trigger to reduce stock when an item is ordered
CREATE OR REPLACE FUNCTION reduce_stock()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Products SET stock = stock - NEW.quantity
    WHERE product_id = NEW.product_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql

CREATE TRIGGER trg_reduce_stock
AFTER INSERT ON Order_items
FOR EACH ROW
EXECUTE FUNCTION reduce_stock();

SELECT * FROM Products WHERE Product_id = 7;
INSERT INTO Order_items (order_id, product_id, quantity, price)
VALUES (25,7,10,5000);
SELECT * FROM Products WHERE Product_id = 7;
```

	product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	7	Logitech Webcam	Electronics	5000.00	25	2025-11-20 16:33:40.481077

```
INSERT 0 1

Query returned successfully in 300 msec.
```

	product_id [PK] integer	product_name character varying (50)	category character varying (25)	price numeric (10,2)	stock integer	created_at timestamp without time zone
1	7	Logitech Webcam	Electronics	5000.00	15	2025-11-20 16:33:40.481077

```sql
--Trigger to prevent negative stock
CREATE OR REPLACE FUNCTION prevent_neg_stock()
RETURNS TRIGGER AS $$
DECLARE
    stock_bal INT;
BEGIN
    SELECT stock INTO stock_bal FROM products
    WHERE product_id = NEW.product_id;

    IF stock_bal < NEW.quantity THEN
        RAISE EXCEPTION 'Insufficient Stock';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_neg_stk
BEFORE INSERT ON order_items
FOR EACH ROW
EXECUTE FUNCTION prevent_neg_stock();

SELECT * FROM ordeR_items;
INSERT INTO order_items(order_id, product_id,quantity,price)
VALUES (40,5,100,2000);
```

```
ERROR:  Insufficient Stock
CONTEXT:  PL/pgSQL function prevent_neg_stock() line 9 at RAISE
```

19. Conclusion

This project successfully demonstrates the complete process of designing and implementing a fully functional e-commerce database system using PostgreSQL. From database design, normalization, and ER modeling to advanced SQL operations, stored programs, triggers, performance tuning, and transactions, the system covers all major concepts required for modern database development. The project showcases the practical application of SQL in real-world scenarios and reflects a strong understanding of relational database principles.

20. Author

Jeswin K Reji
Aspiring Data Scientist
November 2025
jeswinkr7@gmail.com
LinkedIn GitHub