**Multivendor e-commerce Dashboard**

### Submitted By

| **Student Name** | **Student ID** |
| --- | --- |
| Md. Esrat Hossan | 221-15-4757 |

**MINI LAB PROJECT REPORT**

This Report Presented in Partial Fulfillment of the course **in the Computer Science and Engineering Department**



### DAFFODIL INTERNATIONAL UNIVERSITY

**Dhaka, Bangladesh**

**December 14, 2024**

# Table of Contents

**Declaration** [**i**](#_heading=h.30j0zll)

**Course & Program Outcome** [**ii**](#_heading=h.3znysh7)

1. **Introduction** [**1**](#_heading=h.tyjcwt)
   1. Introduction [1](#_heading=h.1t3h5sf)
   2. Motivation [1](#_heading=h.2s8eyo1)
   3. Objectives [1](#_heading=h.3rdcrjn)
   4. Feasibility Study [1](#_heading=h.lnxbz9)
   5. Gap Analysis [1](#_heading=h.1ksv4uv)
   6. Project Outcome [1](#_heading=h.2jxsxqh)
2. **Proposed Methodology/Architecture** [**2**](#_heading=h.3j2qqm3)
   1. Requirement Analysis & Design Specification [2](#_heading=h.4i7ojhp)
      1. Overview [2](#_heading=h.1ci93xb)
      2. Proposed Methodology/ System Design [2](#_heading=h.2bn6wsx)
      3. UI Design [2](#_heading=h.3as4poj)
   2. Overall Project Plan [2](#_heading=h.49x2ik5)
3. **Implementation and Results** [**3**](#_heading=h.147n2zr)
   1. Implementation [3](#_heading=h.23ckvvd)
   2. Performance Analysis [3](#_heading=h.32hioqz)
   3. Results and Discussion [3](#_heading=h.2grqrue)
4. **Engineering Standards and Mapping** [**4**](#_heading=h.3fwokq0)
   1. Impact on Society, Environment and Sustainability [4](#_heading=h.4f1mdlm)
      1. Impact on Life [4](#_heading=h.19c6y18)
      2. Impact on Society & Environment [4](#_heading=h.28h4qwu)
      3. Ethical Aspects [4](#_heading=h.37m2jsg)
      4. Sustainability Plan [4](#_heading=h.46r0co2)
   2. Project Management and TeamWork [4](#_heading=h.111kx3o)
   3. Complex Engineering Problem [4](#_heading=h.206ipza)
      1. Mapping of Program Outcome [4](#_heading=h.2zbgiuw)
      2. Complex Problem Solving [4](#_heading=h.3ygebqi)
      3. Engineering Activities [5](#_heading=h.3cqmetx)

Table of Contents Table of Contents

1. **Conclusion** [**6**](#_heading=h.2r0uhxc)
   1. Summary [6](#_heading=h.3q5sasy)
   2. Limitation [6](#_heading=h.kgcv8k)
   3. Future Work [6](#_heading=h.43ky6rz)

**References** [**6**](#_heading=h.43ky6rz)

**Chapter 1**

# Introduction

Every chapter should start with 1-2 sentences on the outline of the chapter.

### Introduction

Multivendor platforms are leading the way in the fast-paced expansion of the e-commerce sector. These platforms give administrators a centralized dashboard to oversee operations while allowing several vendors to sign up, list, and sell their goods. The goal of the Multivendor E-commerce Dashboard is to offer a reliable solution that makes managing vendors easier, improves customer satisfaction, and expedites order processing.

### Motivation

* Multivendor systems meet the demands of cost-effectiveness, scalability, and flexibility as digital marketplaces grow.
* The lack of variety in product offerings in traditional single-vendor systems reduces client satisfaction.
* By lowering costs, improving accessibility, and boosting market competitiveness, resolving these issues will benefit all parties involved.

### Objectives

* Create and put into use a database schema for an e-commerce system with several vendors.
* Create a dashboard with order monitoring, product management, and vendor registration.
* Ensure scalability, security, and dependability in managing massive volumes of data.
* Put in place features for consumer feedback, payment processing, and cart management.

### Feasibility Study

**Technical viability**: PHP/Node.js is used for the backend and MySQL is used for database administration, guaranteeing scalability and compatibility.

**Operational Feasibility**: By enabling streamlined procedures for administrators, sellers, and customers, the dashboard satisfies operational criteria.

**Economic Feasibility**: When compared to outsourcing multivendor system development, this solution is more affordable.

### Gap Analysis

* Managing vendor scalability and preserving system performance under high load are problems for current platforms.
* The goal of this project is to create a multivendor management dashboard that is effective, lightweight, and adaptable.

### Project Outcome

A multivendor dashboard that is fully operational and includes essential components for order tracking, payment processing, and product management, Improved user experience for buyers and sellers alike.

**Chapter 2**

# Proposed Methodology/Architecture

Every chapter should start with 1-2 sentences on the outline of the chapter.

### Requirement Analysis & Design Specification

**Functional requirements**: Admin, Seller, and Customer are the user roles.

Modules include payment processing, inventory tracking, and product management.

**Non-Functional Requirements**: High performance, security, and scalability are non-functional requirements.

#### Overview

A modular architecture underpins the system design, with distinct modules for managing user, order, and product data.

#### Proposed Methodology/ System Design

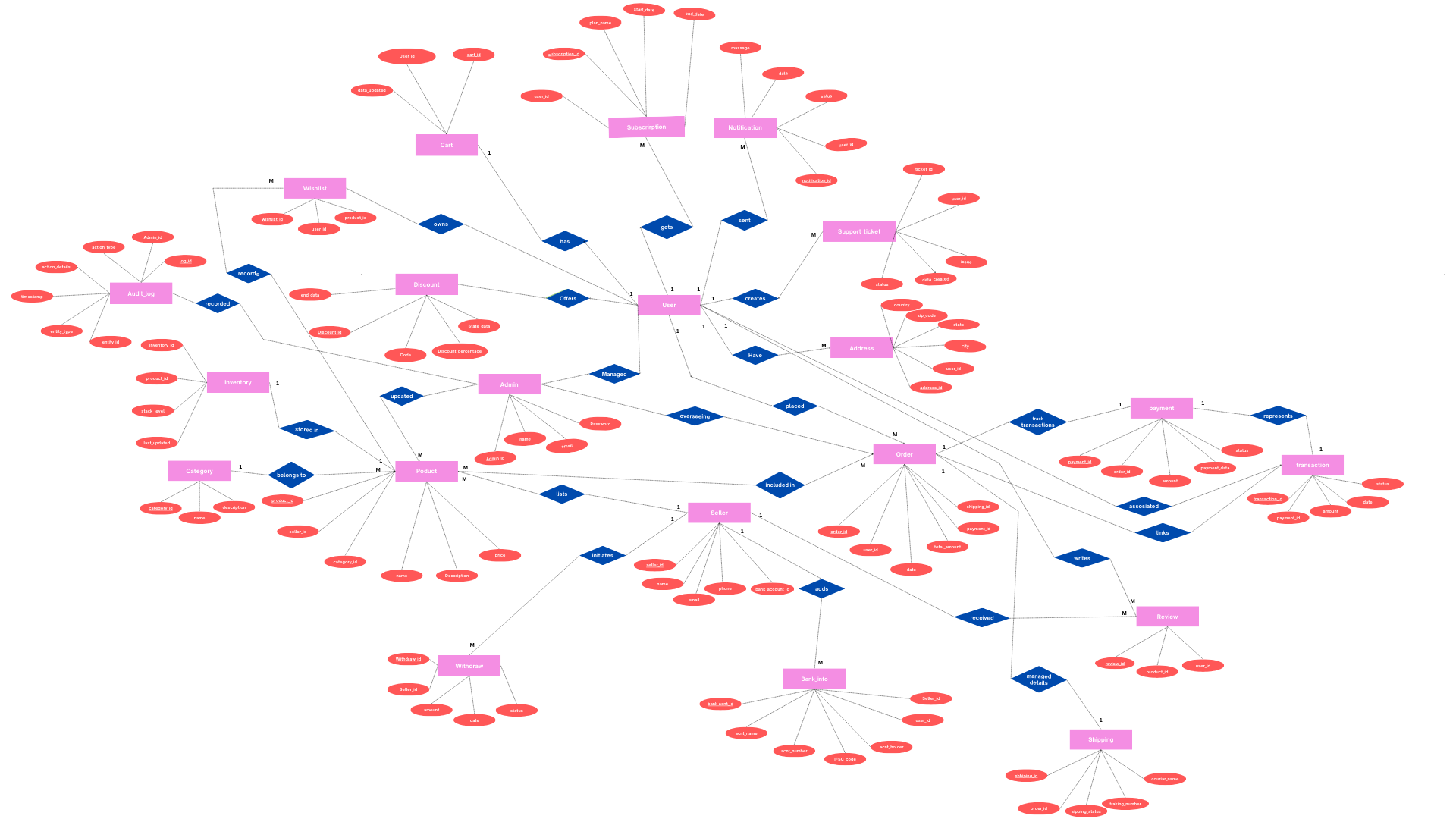


Figure 2.1: This is a ER diagram

<https://www.canva.com/design/DAGXOE8PytE/jWcziM-qWnBeQGf_TmXqvQ/edit>



Figure 2.2: Relational Schema

<https://www.canva.com/design/DAGY6vVsG5I/bIF1CUnEuD0eoiXXY1vqeA/view?utm_content=DAGY6vVsG5I&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=he95deef2bc>

#### UI Design

The design of our platform is shown below:



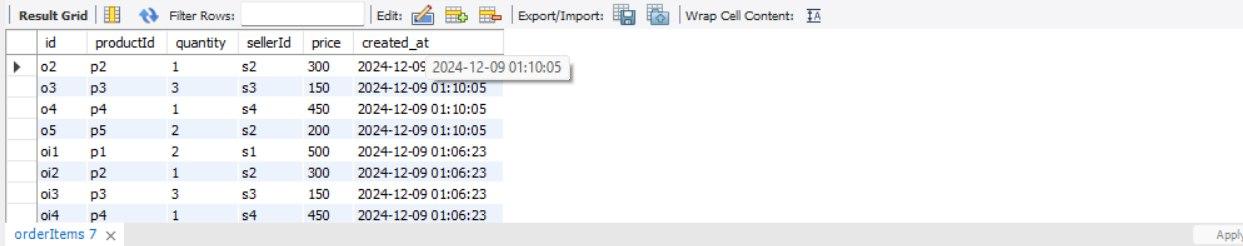
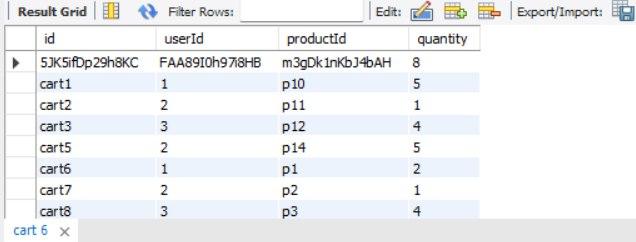


Fig 2.3 : Order and Orderitems tables



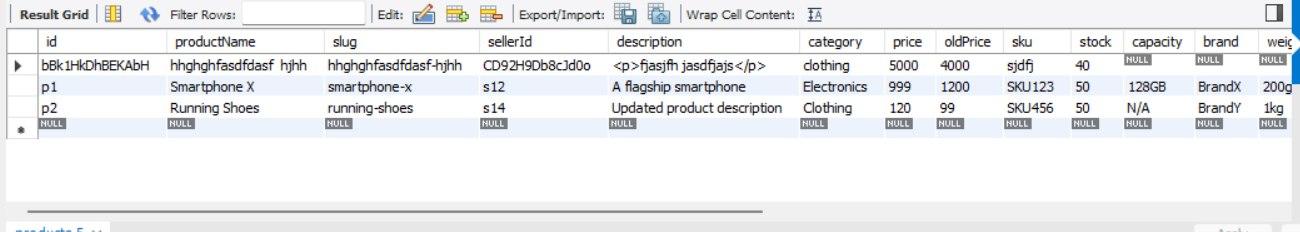
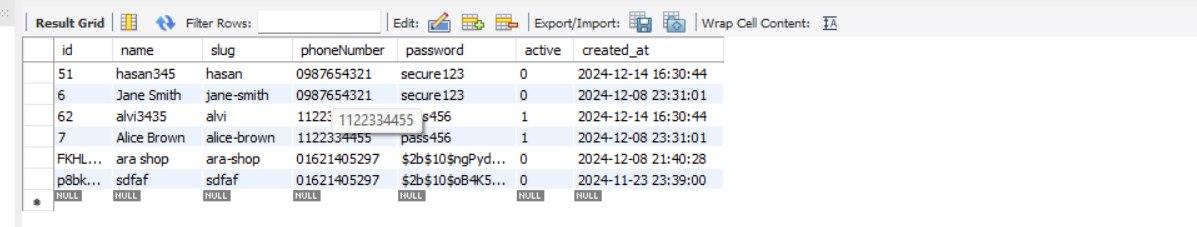
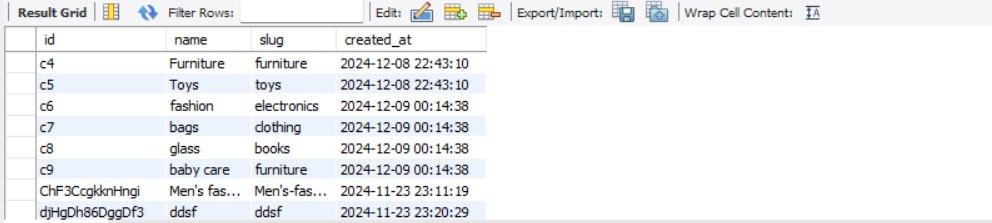


Fig 2.4 - Cart and Product tables



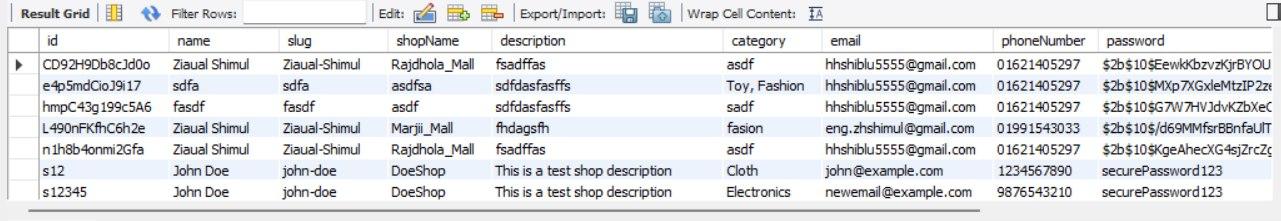


Fig 2.5 - Seller, Category and Bankinfo tables

### Overall Project Plan

Phase 1: Design and configuration of the database.

Phase 2: Development of the backend.

Phase 3: Testing and frontend integration.

Phase 4: User testing and deployment.

**Chapter 3**

# Implementation and Results

Every chapter should start with 1-2 sentences on the outline of the chapter.

### Implementation

CREATE DATABASE e\_com;

USE e\_com;

CREATE TABLE products (

id VARCHAR(20) PRIMARY KEY,

productName VARCHAR(300) NOT NULL,

slug VARCHAR(350),

sellerId varchar(20) NOT NULL,

description TEXT,

category VARCHAR(100),

price int,

oldPrice int,

sku VARCHAR(100),

stock INT,

capacity VARCHAR(100),

brand VARCHAR(100),

weight VARCHAR(100),

dimensions VARCHAR(100),

sold\_out int default 0,

sizes JSON,

colors JSON,

keywords JSON,

active bool,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- DROP TABLE IF EXISTS products;

-- Drop Table products;

Create table category(

id VARCHAR(20) PRIMARY KEY,

name varchar(50) NOT NULL,

slug varchar(80)NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Create table users(

id VARCHAR(20) PRIMARY KEY,

name varchar(50) NOT NULL,

slug varchar(80)NOT NULL,

phoneNumber varchar(15),

password varchar(150),

active bool,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Create table sellers(

id VARCHAR(20) PRIMARY KEY,

name varchar(50) NOT NULL,

slug varchar(80)NOT NULL,

shopName varchar(15),

description varchar(500),

category varchar(50),

email varchar(40),

phoneNumber varchar(15),

password varchar(150),

active bool,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

Create table cart(

id varchar(20) primary key,

userId varchar(20) not null,

productId varchar(20) not null,

quantity int not null,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE wishes (

id VARCHAR(20) PRIMARY KEY,

userId VARCHAR(20) NOT NULL,

productId VARCHAR(20) NOT NULL,

added\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

create TABLE OrderItems (

id VARCHAR(255) PRIMARY KEY,

productId VARCHAR(255) NOT NULL ,

quantity INT NOT NULL,

sellerId VARCHAR(255) NOT NULL,

price int not null,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE orders (

id VARCHAR(14) PRIMARY KEY,

userId VARCHAR(255) NOT NULL,

totalPrice DECIMAL(10, 2) NOT NULL,

status ENUM('pending', 'confirmed', 'shipped', 'delivered', 'cancelled') DEFAULT 'pending',

address JSON NOT NULL,

createdAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

CREATE TABLE DiscountCoupon (

id INT AUTO\_INCREMENT PRIMARY KEY,

couponCode varchar(10),

productId varchar(10) NOT NULL,

discountPrice DECIMAL(10, 2) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

drop table DiscountCoupon;

CREATE TABLE Address (

id VARCHAR(100) PRIMARY KEY,

userId VARCHAR(20) NOT NULL,

division VARCHAR(100) NOT NULL,

district VARCHAR(100) NOT NULL,

upazilla VARCHAR(100) NOT NULL,

post\_code VARCHAR(20) NOT NULL,

area varchar(60),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Drop Table Address;

CREATE TABLE ProductReview (

id INT AUTO\_INCREMENT PRIMARY KEY,

productId varchar(10) NOT NULL,

review VARCHAR(200),

rating INT CHECK (rating BETWEEN 1 AND 5),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

drop table ProductReview;

CREATE TABLE SellerReview (

id INT AUTO\_INCREMENT PRIMARY KEY,

sellerId varchar(10) NOT NULL,

review VARCHAR(200),

rating INT CHECK (rating BETWEEN 1 AND 5),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- drop table SellerReview;

CREATE TABLE Transactions (

id INT AUTO\_INCREMENT PRIMARY KEY,

amount DECIMAL(10, 2) NOT NULL,

orderId varchar(20) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- drop table Transactions;

CREATE TABLE BankInformation (

id INT AUTO\_INCREMENT PRIMARY KEY,

sellerId varchar(20) NOT NULL, -- Foreign key to reference the user

bankName VARCHAR(100) NOT NULL,

accountNumber VARCHAR(50) NOT NULL UNIQUE,

accountHolderName VARCHAR(100) NOT NULL,

branchName VARCHAR(100),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- drop table BankInformation;

CREATE TABLE Subscription (

id INT AUTO\_INCREMENT PRIMARY KEY,

email VARCHAR(255) NOT NULL UNIQUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE Withdraw (

id INT AUTO\_INCREMENT PRIMARY KEY,

sellerId INT NOT NULL, -- Reference to the seller

bankInfoId INT NOT NULL, -- Reference to the bank information

amount DECIMAL(10, 2) NOT NULL, -- Amount to be withdrawn

status ENUM('pending', 'approved', 'rejected') DEFAULT 'pending', -- Status of the withdrawal

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE Support (

id INT AUTO\_INCREMENT PRIMARY KEY,

userId VARCHAR(20) NOT NULL,

issue TEXT NOT NULL,

status VARCHAR(50) NOT NULL DEFAULT 'Open',

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE Inventory (

id INT AUTO\_INCREMENT PRIMARY KEY,

productId VARCHAR(20) NOT NULL,

productId VARCHAR(100) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

-- users ------------------------------------users query -------

select \* from users;

SELECT \* FROM users LIMIT 100;

INSERT INTO users (id, slug, name, phoneNumber, password, active)

VALUES ('12345', 'john-doe', 'John Doe', '1234567890', 'securePassword', false);

INSERT INTO users (id, slug, name, phoneNumber, password, active) -- insert users

VALUES

('4', 'shiblu', 'shiblu12', '1234567890', 'password123', true),

('5', 'hasan', 'hasan345', '0987654321', 'secure123', false),

('6', 'alvi', 'alvi3435', '1122334455', 'pass456', true);

SELECT -- growth user per month

DATE\_FORMAT(MIN(created\_at), '%b') AS month,

COUNT(\*) AS users

FROM users

GROUP BY DATE\_FORMAT(created\_at, '%Y-%m')

ORDER BY MIN(created\_at);

UPDATE users -- taggle active status

SET active = CASE

WHEN active = 1 THEN 0

ELSE 1

END

WHERE id = "1";

DELETE FROM users WHERE id ="AeLl45480l2AHd" ;

-- sellers query --------------------------------------- sellers query

INSERT INTO sellers (

id, name, slug, shopName, description, category, email, phoneNumber, password, active

) VALUES

(

's2', ' hasan', 'hasan232', 'AliceStyle', 'Stylish clothing for all seasons', 'Cloth',

'a@example.com', '9876543210', 'password123', true

),

(

's3', 'fardus alam', 'fardus-alam324', 'SmithGadgets', 'Affordable electronics and gadgets', 'Electronics',

'robert@example.com', '4567891230', 'password456', false

),

(

's4', 'Emma', 'emma123', 'EmmaJewels', 'Unique handmade jewelry', 'Accessories',

'emma@example.com', '7891234560', 'password789', true

);

SELECT \* FROM sellers;

SELECT \* FROM sellers limit 10;

SELECT \* FROM sellers -- category seller

WHERE category = 'Electronics';

UPDATE sellers -- taggle active status

SET active = CASE

WHEN active = 1 THEN 0

ELSE 1

END

WHERE id = "s1";

SELECT \* FROM sellers -- select seller by id

WHERE id = 's12345';

SELECT \* FROM sellers -- last 30 days sellers

WHERE created\_at >= NOW() - INTERVAL 30 DAY;

SELECT category, COUNT(\*) AS active\_sellers -- active seller count by category

FROM sellers

WHERE active = true

GROUP BY category;

UPDATE sellers

SET email = 'newemail@example.com', phoneNumber = '9876543210'

WHERE id = 's12345';

SELECT \* FROM sellers -- Paginated Sellers List

ORDER BY created\_at DESC

LIMIT 10 OFFSET 0;

SELECT \* FROM sellers -- find sellers by email

WHERE email = 'john@example.com';

SELECT \* FROM sellers -- search seller use like

WHERE name LIKE '%Doe%' OR shopName LIKE '%Shop%';

SELECT -- growth sellers per month

DATE\_FORMAT(MIN(created\_at), '%b') AS month,

COUNT(\*) AS sellers

FROM sellers

GROUP BY DATE\_FORMAT(created\_at, '%Y-%m')

ORDER BY MIN(created\_at);

SELECT \* FROM sellers

WHERE description LIKE '%test%';

DELETE FROM sellers

WHERE id = 's12345';

-- category query -------------------------------- category query--------

INSERT INTO category (id, name, slug) VALUES

('c6', 'fashion', 'electronics'),

('c7', 'bags', 'clothing'),

('c8', 'glass', 'books'),

('c9', 'baby care', 'furniture');

SELECT \* FROM category; -- select all

UPDATE category -- Update a category's name

SET name = 'Apparel', slug = 'apparel'

WHERE id = 'c2';

SELECT COUNT(\*) AS total\_category

FROM category;

SELECT \* FROM category

ORDER BY created\_at DESC;

SELECT \* FROM category -- use like

WHERE slug LIKE '%ele%';

DELETE FROM category -- find category by books

WHERE id = 'c5';

SELECT \* FROM category -- find category by books

WHERE name = 'Books';

-- product query -------------------------------product query-------------

INSERT INTO products (

id, productName, slug, sellerId, description, category, price, oldPrice, sku,

stock, capacity, brand, weight, dimensions, sold\_out, sizes, colors, keywords, active

) VALUES

('p1', 'Smartphone X', 'smartphone-x', 's12', 'A flagship smartphone', 'Electronics', 999, 1200, 'SKU123',

50, '128GB', 'BrandX', '200g', '150x75x8mm', 10,

JSON\_ARRAY('128GB', '256GB'), JSON\_ARRAY('Black', 'Silver'), JSON\_ARRAY('smartphone', 'electronics'), true),

('p2', 'Running Shoes', 'running-shoes', 's14', 'Comfortable running shoes', 'Clothing', 79, 99, 'SKU456',

200, 'N/A', 'BrandY', '1kg', '30x10x10cm', 5,

JSON\_ARRAY('7', '8', '9'), JSON\_ARRAY('Red', 'Blue'), JSON\_ARRAY('shoes', 'sports'), true),

('p3', 'Fiction Book', 'fiction-book', 's13', 'An engaging fiction novel', 'Books', 15, 20, 'SKU789',

100, 'N/A', 'BrandZ', '300g', '20x15x3cm', 2,

JSON\_ARRAY('Paperback', 'Hardcover'), JSON\_ARRAY('N/A'), JSON\_ARRAY('fiction', 'books'), true);

SELECT \* FROM products; -- all products

SELECT -- all products with seller details

p.id AS product\_id,

p.productName,

p.slug AS product\_slug,

s.id AS seller\_id,

s.name AS seller\_name,

p.category,

p.price,

p.stock

FROM

products p

JOIN

sellers s

ON

p.sellerId = s.id;

SELECT -- Find products by category and their seller names

p.productName,

s.name AS seller\_name

FROM

products p

JOIN

sellers s

ON

p.sellerId = s.id

WHERE

p.category = 'Electronics';

SELECT -- Count total products sold by each seller

s.id AS seller\_id,

s.name AS seller\_name,

COUNT(p.id) AS total\_products

FROM

sellers s

LEFT JOIN

products p

ON

s.id = p.sellerId

GROUP BY

s.id, s.name;

SELECT -- products with low stock and their seller details

p.productName,

p.stock,

s.name AS seller\_name

FROM

products p

JOIN

sellers s

ON

p.sellerId = s.id

WHERE

p.stock < 500;

SELECT -- active products and their seller details

p.productName,

s.name AS seller\_name,

p.price

FROM

products p

JOIN

sellers s

ON

p.sellerId = s.id

WHERE

p.active = true;

SELECT -- products sorted by creation date

p.id,

p.productName,

p.created\_at,

s.name AS seller\_name

FROM

products p

JOIN

sellers s

ON

p.sellerId = s.id

ORDER BY

p.created\_at DESC;

SELECT -- filter product

p.productName,

p.price,

s.name AS seller\_name

FROM

products p

JOIN

sellers s ON p.sellerId = s.id

WHERE

p.category = 'Electronics'

AND p.price BETWEEN 5 AND 5000;

SELECT -- Count Total Products Sold by Each Seller

s.name AS seller\_name,

COUNT(p.id) AS total\_products

FROM

sellers s

LEFT JOIN

products p ON s.id = p.sellerId

GROUP BY

s.id, s.name;

UPDATE products -- Update a product's price

SET price = 69

WHERE id = 'p2';

UPDATE products

SET

price = 120,

stock = 50,

description = 'Updated product description'

WHERE

id = 'p2';

SELECT -- Products with Low Stock Alerts

p.productName,

p.stock,

s.name AS seller\_name

FROM

products p

JOIN

sellers s ON p.sellerId = s.id

WHERE

p.stock < 100;

SELECT -- Search Products by Keywords

id,

productName,

price

FROM

products

WHERE

description LIKE '%Wireless%' OR

productName LIKE '%s%';

DELETE FROM products -- delete products

WHERE id = 'p3';

DELETE FROM products

WHERE id IN ('p1', 'p2', 'p3');

-- cart query --------------------------cart query

INSERT INTO cart (id, userId, productId, quantity)

VALUES

('cart1', '1', 'p10', 2),

('cart2', '2', 'p11', 1),

('cart3', '3', 'p12', 4);

SELECT -- All Cart Items for a Specific User

c.id,

c.quantity,

p.productName,

p.price,

s.name AS seller\_name

FROM

cart c

JOIN

products p ON c.productId = p.id

JOIN

sellers s ON p.sellerId = s.id

WHERE

c.userId = '1';

UPDATE cart -- Update Quantity of a Product in the Cart

SET quantity = 5

WHERE id = 'cart1';

DELETE FROM cart -- Delete a Specific Product from the Cart

WHERE id = 'cart1';

DELETE FROM cart -- Delete users Cart

WHERE userId = '1';

SELECT -- Cart Summary with Total Cost for a User

c.userId,

SUM(p.price \* c.quantity) AS total\_cost

FROM

cart c

JOIN

products p ON c.productId = p.id

WHERE

c.userId = '1'

GROUP BY

c.userId;

SELECT -- Cart Details with User Information

c.id,

u.name AS user\_name,

p.productName,

c.quantity,

p.price

FROM

cart c

JOIN

users u ON c.userId = u.id

JOIN

products p ON c.productId = p.id;

SELECT -- Detailed Cart Information with Prices, Quantities, and Seller Info

c.id,

p.productName,

p.price,

c.quantity,

s.name AS seller\_name

FROM

cart c

JOIN

products p ON c.productId = p.id

JOIN

sellers s ON p.sellerId = s.id;

SELECT -- Products in Cart with Discount Information

p.productName,

p.price,

p.oldPrice,

c.quantity

FROM

cart c

JOIN

products p ON c.productId = p.id

WHERE

c.userId = '1';

SELECT -- Cart Quantities by Product

p.productName,

SUM(c.quantity) AS total\_quantity

FROM

cart c

JOIN

products p ON c.productId = p.id

GROUP BY

p.productName;

-- wishes query --------------------------------------wishes query

-- Insert 4-5 sample wishlist items

INSERT INTO wishes (id, userId, productId)

VALUES

('wish1', '1', 'p1'),

('wish2', '2', 'p2'),

('wish3', '3', 'p3');

SELECT -- All Cart Items for a Specific User

w.id,

p.productName,

p.price,

s.name AS seller\_name

FROM

wishes w

JOIN

products p ON w.productId = p.id

JOIN

sellers s ON p.sellerId = s.id

WHERE

w.userId = '1';

DELETE FROM wishes -- Delete users Cart

WHERE userId = '1';

-- order query --------------------- order query-------------

INSERT INTO OrderItems (id, productId, quantity, sellerId, price)

VALUES

('o1', 'p1', 2, 's1', 500),

('o2', 'p2', 1, 's2', 300),

('o3', 'p3', 3, 's3', 150),

('o4', 'p4', 1, 's4', 450),

('o5', 'p5', 2, 's2', 200);

INSERT INTO orders (id, userId, totalPrice, status, address)

VALUES

('o1', '1', 1200.00, 'pending', '{"city": "New York", "zip": "10001"}'),

('o2', '1', 800.00, 'confirmed', '{"city": "Los Angeles", "zip": "90001"}'),

('o3', '2', 1500.00, 'shipped', '{"city": "Chicago", "zip": "60614"}'),

('o4', '3', 500.00, 'cancelled', '{"city": "Houston", "zip": "77002"}'),

('o5', '2', 1000.00, 'delivered', '{"city": "San Francisco", "zip": "94107"}');

-- Join orders and OrderItems to get detailed information

SELECT

o.id AS order\_id,

o.userId ,

o.totalPrice ,

o.status ,

oi.productId ,

oi.quantity ,

oi.price ,

o.address

FROM

orders AS o

JOIN

OrderItems AS oi ON o.id = oi.id;

SELECT -- Calculate the total order price

o.id AS order\_id,

SUM(oi.quantity \* oi.price) AS order\_total

FROM

orders AS o

JOIN

OrderItems AS oi ON o.id = oi.id

GROUP BY

o.id;

SELECT -- Fetch orders with status 'delivered'

o.id,

o.userId ,

o.totalPrice ,

o.status

FROM

orders AS o

WHERE

o.status = 'delivered';

UPDATE orders -- Update an Order Status

SET status = 'shipped', updatedAt = CURRENT\_TIMESTAMP

WHERE id = 'o1';

SELECT -- Count orders grouped by status

status, COUNT(\*) AS order\_count

FROM

orders

GROUP BY

status;

SELECT -- Find Orders Containing Specific Products

o.id ,

o.userId,

o.totalPrice,

oi.productId,

oi.quantity

FROM

orders AS o

JOIN

OrderItems AS oi ON o.id = oi.id

WHERE

oi.productId = 'p3';

SELECT -- Fetch orders placed in the last 7 days

id AS order\_id,

userId AS customer\_id,

totalPrice AS order\_total,

status,

createdAt AS order\_date

FROM

orders

WHERE

createdAt >= DATE\_SUB(CURRENT\_DATE, INTERVAL 7 DAY);

SELECT -- Calculate total revenue from all delivered orders

SUM(totalPrice) AS revenue

FROM

orders

WHERE

status = 'delivered';

UPDATE orders -- Update order's total price

SET totalPrice = 1500.00, updatedAt = CURRENT\_TIMESTAMP

WHERE id = 'o2';

SELECT -- Detailed Statistics about OrderItems

productId,

SUM(quantity) AS total\_sold\_quantity,

SUM(quantity \* price) AS total\_sales\_value

FROM

OrderItems

GROUP BY

productId;

SELECT -- Get Order Information

o.id ,

o.userId ,

o.totalPrice ,

s.name AS seller\_name,

oi.productId ,

oi.quantity

FROM

orders AS o

JOIN

OrderItems AS oi ON o.id = oi.id

JOIN

sellers AS s ON oi.sellerId = s.id;

SELECT -- Get top-selling products

productId,

SUM(quantity) AS total\_quantity\_sold

FROM

OrderItems

GROUP BY

productId

ORDER BY

total\_quantity\_sold DESC;

SELECT -- View Full Address Information

o.id ,

o.address

FROM

orders AS o;

DELETE FROM OrderItems -- order Items delete

WHERE id = 'o1';

DELETE FROM orders WHERE id = 'o1'; -- orders delete by id

-- transactions query ----------------- Transaction query

INSERT INTO Transactions (amount, orderId)

VALUES

(100.50, 'O1'),

(200.75, 'O2'),

(150.00, 'O3');

SELECT -- order and transactions join query

t.id ,

t.amount ,

t.created\_at,

o.id ,

o.userId,

o.totalPrice

FROM

Transactions t

JOIN

orders o ON t.orderId = o.id;

SELECT

SUM(t.amount) AS totalPrice

FROM

Transactions t

WHERE

orderId = 'O1';

DELETE FROM Transactions

WHERE id = 1;

UPDATE Transactions

SET amount = 250.00

WHERE id = 2;

SELECT COUNT(\*) AS TotalTransactions

FROM Transactions;

SELECT -- join query ransactions for a Specific User

t.id,

t.amount,

t.created\_at,

o.id ,

o.userId ,

o.totalPrice

FROM

Transactions t

JOIN

orders o ON t.orderId = o.id

WHERE

o.userId = '1';

-- address query address query ------------------

-- Insert sample address for a user

INSERT INTO Address (id, userId, division, district, upazilla, post\_code, area)

VALUES ('a1', '1', 'Dhaka', 'Dhaka District', 'Gulshan', '1212', 'Gulshan Avenue');

-- Insert sample address for a seller

INSERT INTO Address (id, userId, division, district, upazilla, post\_code, area)

VALUES ('a2', 's1', 'Chittagong', 'Chittagong District', 'Patenga', '4203', 'Patenga Sea Road');

SELECT \* FROM Address;

SELECT \* -- To find all addresses for a specific userId

FROM Address

WHERE userId = '1';

SELECT \* -- To find all addresses for a division

FROM Address

WHERE division = 'Dhaka';

SELECT COUNT(\*) AS TotalAddresses -- total address

FROM Address;

UPDATE Address -- update area by userId

SET area = 'New Gulshan Avenue'

WHERE id = '1';

DELETE FROM Address WHERE id = 'a1'; -- address delete by id

-- subcription query ------------------------- subcription query

INSERT INTO Subscription (email)

VALUES ('hasan@example.com');

SELECT \*

FROM Subscription;

SELECT \*

FROM Subscription

WHERE email = 'hasan@example.com';

UPDATE Subscription

SET email = 'email@example.com'

WHERE id = 1;

SELECT COUNT(\*) AS TotalSubscriptions -- count total subcription

FROM Subscription;

SELECT \* -- subscriptions in descending

FROM Subscription

ORDER BY created\_at DESC;

SELECT \* -- after a specific date

FROM Subscription

WHERE created\_at > '2023-01-01';

-- bankinformation -------------------------bankinformation

INSERT INTO BankInformation (sellerId, bankName, accountNumber, accountHolderName, branchName)

VALUES

('s1', 'ABC Bank', '1234567890', 'alvi', 'Main Branch'),

('s2', 'XYZ Bank', '0987654321', 'hasan', 'Central Branch'),

('s3', 'PQR Bank', '1122334455', 'alip', 'ashulia Branch'),

('s4', 'LMN Bank', '6677889900', 'tanjid', 'City Branch');

SELECT \*

FROM BankInformation;

SELECT \*

FROM BankInformation

WHERE sellerId = "s1";

SELECT COUNT(\*) AS TotalBankInformation

FROM BankInformation;

SELECT -- join query

b.id ,

sellers.name,

b.bankName,

b.accountNumber,

b.accountHolderName,

b.branchName,

b.created\_at

FROM

BankInformation b

JOIN

sellers ON b.sellerId = sellers.id;

SELECT

b.id ,

s.name,

b.bankName,

b.accountNumber,

b.accountHolderName,

b.branchName,

b.created\_at

FROM

BankInformation b

JOIN

sellers s ON b.sellerId = s.id

WHERE

s.id = 's2';

-- all review query ----------------all review query

INSERT INTO ProductReview (productId, review, rating)

VALUES

('p1', 'Great product, highly recommended!', 5),

('p2', 'Not bad, could use some improvement.', 3),

('p3', 'Terrible quality, do not buy!', 1);

INSERT INTO SellerReview (sellerId, review, rating)

VALUES

('s1', 'Fast shipping and great service!', 5),

('s2', 'Seller was responsive but the product quality was poor.', 2),

('s3', 'Amazing seller, highly professional!', 5);

SELECT \*

FROM ProductReview;

SELECT \*

FROM SellerReview;

SELECT \*

FROM ProductReview

WHERE productId = 'p1';

SELECT \*

FROM SellerReview

WHERE sellerId = 's2';

SELECT \*

FROM ProductReview

WHERE rating > 4;

SELECT \*

FROM SellerReview

WHERE rating > 4;

UPDATE ProductReview

SET rating = 4

WHERE id = 2;

DELETE FROM ProductReview

WHERE id = 3;

SELECT COUNT(\*) AS TotalReviews

FROM ProductReview

WHERE productId = 'p1';

SELECT COUNT(\*) AS TotalReviews

FROM SellerReview

WHERE sellerId = 's1';

SELECT AVG(rating) AS AverageRating

FROM ProductReview

WHERE productId = 'p2';

SELECT AVG(rating) AS AverageRating

FROM SellerReview

WHERE sellerId = 's1';

SELECT -- join query

p.productId,

p.review AS ProductReview,

p.rating AS ProductRating,

s.sellerId,

s.review AS SellerReview,

s.rating AS SellerRating

FROM

ProductReview p

JOIN

products pr on p.productId = pr.id

JOIN

SellerReview s ON pr.sellerId = s.sellerId;

-- discount coupon query here ---------------------------

INSERT INTO DiscountCoupon (couponCode, productId, discountPrice)

VALUES

('DISCOUNT10', 'P1', 10),

('SUMMER20', 'P2', 20),

('WINTER30', 'P3', 30);

SELECT -- join query

d.id ,

d.couponCode,

d.discountPrice,

p.productName,

p.price,

p.sku,

p.brand,

p.category

FROM

DiscountCoupon d

JOIN

Products p ON d.productId = p.id;

SELECT

d.id ,

d.couponCode,

d.discountPrice

FROM

DiscountCoupon d

WHERE

productId = 'P1';

UPDATE DiscountCoupon

SET discountPrice = 15.00

WHERE couponCode = 'DISCOUNT10';

DELETE FROM DiscountCoupon

WHERE id = 1;

SELECT -- Products Eligible for Discounts

p.id ,

p.productName,

p.price ,

d.couponCode,

d.discountPrice

FROM

Products p

JOIN

DiscountCoupon d ON p.id = d.productId

WHERE

p.active = true;

SELECT COUNT(\*) AS TotalCoupons

FROM DiscountCoupon

WHERE productId = 'P2';

SELECT -- Calculate Final Price After Discount

p.productName,

p.price ,

d.discountPrice,

(p.price - d.discountPrice) AS FinalPrice

FROM

DiscountCoupon d

JOIN

Products p ON d.productId = p.id;

-- support query ---------support query---------

INSERT INTO Support (userId, issue, status)

VALUES

('1', 'Issue with logging into the account', 'Open'),

('2', 'Payment not processed successfully', 'In Progress'),

('3', 'Product returned but refund not received', 'Resolved');

SELECT \* FROM Support;

SELECT COUNT(\*) AS OpenIssues

FROM Support

WHERE status = 'Resolved';

DELETE FROM Support

WHERE id = 2;

SELECT -- user info

Support.id ,

Support.userId ,

Support.issue ,

Support.status ,

Support.created\_at,

users.name,

users.phoneNumber

FROM

Support

LEFT JOIN

users ON Support.userId = users.id;

SELECT \* FROM Support

WHERE userId = '2';

UPDATE Support

SET status = 'Resolved'

WHERE id = 1;

-- inventory query --- inventory

INSERT INTO Inventory (productId, stack\_level)

VALUES

('P1', 'Low'),

('P2', 'Medium'),

('P3', 'High');

SELECT \* FROM Inventory;

SELECT \*

FROM Inventory

WHERE created\_at >= NOW() - INTERVAL 7 DAY;

SELECT

i.id ,

i.productId,

p.productName,

i.stack\_level ,

i.created\_at

FROM

Inventory i

JOIN

Products p ON i.productId = p.id;

SELECT \*

FROM Inventory

WHERE updated\_at > created\_at;

UPDATE Inventory

SET stack\_level = 'Medium'

WHERE productId = 'P1';

### Performance Analysis

Queries:

1. List all products along with their seller's name

SELECT

products.productName,

sellers.name AS sellerName

FROM

products

INNER JOIN

sellers

ON

products.sellerId = sellers.id;

1. List all sellers and their products (including sellers with no products)

SELECT

sellers.name AS sellerName,

products.productName

FROM

sellers

LEFT JOIN

products

ON

sellers.id = products.sellerId;

1. List all products and their respective seller information (including products with no seller info)

SELECT

products.productName,

sellers.name AS sellerName

FROM

products

RIGHT JOIN

sellers

ON

products.sellerId = sellers.id;

1. Retrieve order details with products and user information

SELECT

orders.id AS orderId,

users.name AS userName,

products.productName,

OrderItems.quantity

FROM

orders

INNER JOIN

users

ON

orders.userId = users.id

INNER JOIN

OrderItems

ON

orders.id = OrderItems.id

INNER JOIN

products

ON

OrderItems.productId = products.id;

#### 5. List all users and their orders (including users without orders)

SELECT

users.name AS userName,

orders.id AS orderId,

orders.totalPrice

FROM

users

LEFT JOIN

orders

ON

users.id = orders.userId;

1. Retrieve product reviews with product names

SELECT

products.productName,

ProductReview.review,

ProductReview.rating

FROM

ProductReview

INNER JOIN

products

ON

ProductReview.productId = products.id;

1. List all order items with their corresponding seller and product information

SELECT

OrderItems.id AS orderItemId,

products.productName,

sellers.name AS sellerName,

OrderItems.quantity

FROM

OrderItems

INNER JOIN

products

ON

OrderItems.productId = products.id

INNER JOIN

sellers

ON

OrderItems.sellerId = sellers.id;

1. Find all products and their reviews (including products without reviews)

SELECT

products.productName,

ProductReview.review,

ProductReview.rating

FROM

products

LEFT JOIN

ProductReview

ON

products.id = ProductReview.productId;

1. List all transactions along with their corresponding order details

SELECT

Transactions.id AS transactionId,

Transactions.amount,

orders.id AS orderId,

orders.totalPrice

FROM

Transactions

INNER JOIN

orders

ON

Transactions.orderId = orders.id;

1. Retrieve all sellers and their reviews (including sellers with no reviews)

SELECT

sellers.name AS sellerName,

SellerReview.review,

SellerReview.rating

FROM

sellers

LEFT JOIN

SellerReview

ON

sellers.id = SellerReview.sellerId;

11. calculate final result after discount

SELECT

p.productName,

p.price ,

d.discountPrice,

(p.price - d.discountPrice) AS FinalPrice

FROM

DiscountCoupon d

JOIN

Products p ON d.productId = p.id;

12. show all products and discountInformation which Products Eligible for Discounts.

SELECT

p.id ,

p.productName,

p.price ,

d.couponCode,

d.discountPrice

FROM

Products p

JOIN

DiscountCoupon d ON p.id = d.productId

WHERE

p.active = true;

13. count totalCoupons for productid p2

SELECT COUNT(\*) AS TotalCoupons

FROM DiscountCoupon

group by productId

WHERE productId = 'P2';

14. select all products along with products review and seller review ;

SELECT

p.productId,

p.review AS ProductReview,

p.rating AS ProductRating,

s.sellerId,

s.review AS SellerReview,

s.rating AS SellerRating

FROM

ProductReview p

JOIN

products pr on p.productId = pr.id

JOIN

SellerReview s ON pr.sellerId = s.sellerId;

15. select avg sellerRating;

SELECT AVG(rating) AS AverageRating

FROM SellerReview

group by sellerId

WHERE sellerId = 's1';

16. select bank information along with seller name where selerId =s1;

SELECT

b.id ,

s.name,

b.bankName,

b.accountNumber,

b.branchName,

b.created\_at

FROM

BankInformation b

JOIN

sellers s ON b.sellerId = s.id

WHERE

s.id = 's2';

17. select all products timestamp is within the last 7 days;

SELECT \*

FROM Products

WHERE created\_at >= NOW() - INTERVAL 7 DAY;

18. find products which name like sa

SELECT

id,

productName,

price

FROM

products

WHERE

productName LIKE '%sa%';

19. taggle active status for sellers where id s1 ;

UPDATE sellers

SET active = CASE

WHEN active = 1 THEN 0

ELSE 1

END

WHERE id = "s1";

20. update price stock and description for p2 product

UPDATE products

SET

price = 2000,

stock = 50,

description = 'Updated product description'

WHERE

id = 'p2';

21. delete multiple products

DELETE FROM products

WHERE id IN ('p1', 'p2', 'p3');

22. How can you find all products sold by a specific seller?

SELECT \*

FROM products

WHERE sellerId = 'seller001';

23. Which users have items in their cart, and what are those items?

SELECT u.id AS user\_id, u.name AS user\_name, p.productName, c.quantity

FROM users u

JOIN cart c ON u.id = c.userId

JOIN products p ON c.productId = p.id;

24. How can you determine the total revenue generated by a specific seller?

SELECT s.id AS seller\_id, s.name AS seller\_name, SUM(oi.price \* oi.quantity) AS total\_revenue

FROM sellers s

JOIN products p ON s.id = p.sellerId

JOIN OrderItems oi ON p.id = oi.productId

WHERE s.id = 'seller001'

GROUP BY s.id, s.name;

25. What are the details of the orders placed by a specific user?

SELECT o.id AS order\_id, o.totalPrice, oi.productId, p.productName, oi.quantity

FROM orders o

JOIN OrderItems oi ON o.id = oi.id

JOIN products p ON oi.productId = p.id

WHERE o.userId = 'user001';

26. How can you find the top 5 best-selling products?

SELECT p.id AS product\_id, p.productName, SUM(oi.quantity) AS total\_sold

FROM products p

JOIN OrderItems oi ON p.id = oi.productId

GROUP BY p.id, p.productName

ORDER BY total\_sold DESC

LIMIT 5;

27. Which sellers have products that are out of stock?  
 SELECT DISTINCT s.id AS seller\_id, s.name AS seller\_name

FROM sellers s

JOIN products p ON s.id = p.sellerId

WHERE p.stock = 0;

28. What are the average ratings for each product?

SELECT p.id AS product\_id, p.productName, AVG(pr.rating) AS avg\_rating

FROM products p

LEFT JOIN ProductReview pr ON p.id = pr.productId

GROUP BY p.id, p.productName;

29. How can you identify orders shipped to a specific district?

SELECT o.id AS order\_id, o.totalPrice, a.district, u.name AS user\_name

FROM orders o

JOIN Address a ON JSON\_EXTRACT(o.address, '$.id') = a.id

JOIN users u ON o.userId = u.id

WHERE a.district = 'Dhaka';

30. What discounts are available for a specific product?

SELECT d.couponCode, d.discountPrice

FROM DiscountCoupon d

WHERE d.productId = 101;

31. Which users have made the most transactions?

SELECT u.id AS user\_id, u.name AS user\_name, COUNT(t.id) AS total\_transactions

FROM users u

JOIN orders o ON u.id = o.userId

JOIN Transactions t ON o.id = t.orderId

GROUP BY u.id, u.name

ORDER BY total\_transactions DESC

LIMIT 3;

32. SELECT products.id, products.productName, sellers.name AS sellerName

FROM products

INNER JOIN sellers ON products.sellerId = sellers.id

WHERE products.price > 100;

33. SELECT category.name AS categoryName, COUNT(products.id) AS productCount

FROM category

LEFT JOIN products ON category.id = products.category

GROUP BY category.name;

34. SELECT products.productName, users.name AS userName, cart.quantity

FROM products

RIGHT JOIN cart ON products.id = cart.productId

LEFT JOIN users ON cart.userId = users.id;

35. SELECT users.name AS userName, products.productName

FROM users

CROSS JOIN products

WHERE products.active = TRUE;

36. SELECT orders.id AS orderId, OrderItems.productId, OrderItems.quantity

FROM orders

INNER JOIN OrderItems ON orders.id = OrderItems.id

WHERE orders.totalPrice > 500;

37. SELECT sellers.name AS sellerName, products.productName

FROM sellers

LEFT JOIN products ON sellers.id = products.sellerId AND products.category = 'Electronics';

38. SELECT SellerReview.review, SellerReview.rating, sellers.name AS sellerName

FROM SellerReview

INNER JOIN sellers ON SellerReview.sellerId = sellers.id

WHERE SellerReview.rating > 3;

39. SELECT users.name AS userName, Address.division, Address.district

FROM users

RIGHT JOIN Address ON users.id = Address.userId;

40. SELECT products.productName, DiscountCoupon.couponCode, DiscountCoupon.discountPrice

FROM products

LEFT JOIN DiscountCoupon ON products.id = DiscountCoupon.productId;

41. SELECT orders.id AS orderId, users.name AS userName, orders.totalPrice

FROM orders

INNER JOIN users ON orders.userId = users.id

WHERE orders.status = 'delivered';

42. Show the names and IDs of all the vendors registered on the platform.

SELECT

vendor\_id,

vendor\_name

FROM vendors;

43. Display the list of products for a specific vendor, including product names, prices, and stock quantities.

SELECT

product\_name, price, stock\_quantity

FROM products

WHERE

vendor\_id = ?;

44. Calculate the total sales or revenue generated by each vendor on the platform.

SELECT

p.vendor\_id, v.vendor\_name, SUM(o.quantity \* p.price) AS total\_revenue

FROM orders o

JOIN products p ON o.product\_id = p.product\_id

JOIN vendors v ON p.vendor\_id = v.vendor\_id

GROUP BY p.vendor\_id, v.vendor\_name;

45. Show the total number of orders placed for a specific vendor, along with order details.

SELECT

o.order\_id, o.customer\_id, o.order\_date, SUM(o.quantity) AS total\_quantity

FROM orders o

JOIN products p ON o.product\_id = p.product\_id

WHERE p.vendor\_id = ?

GROUP BY o.order\_id, o.customer\_id, o.order\_date;

46. Display the order history for a specific customer, showing all the orders they’ve placed.

SELECT

o.order\_id, o.order\_date, p.product\_name, o.quantity, (o.quantity \* p.price) AS total\_amount

FROM orders o

JOIN products p ON o.product\_id = p.product\_id

WHERE o.customer\_id = ?;

47. Calculate the total revenue earned by the platform from all vendors combined.

SELECT

SUM(o.quantity \* p.price) AS total\_platform\_revenue

FROM orders o

JOIN products p ON o.product\_id = p.product\_id;

48. List the top-selling products based on the highest number of units sold..............create sql qury

SELECT

p.product\_id, p.product\_name, SUM(o.quantity) AS total\_units\_sold

FROM orders o

JOIN products p ON o.product\_id = p.product\_id

GROUP BY p.product\_id, p.product\_name

ORDER BY total\_units\_sold DESC

LIMIT 10;

49. Identify products with low stock levels and display them, helping vendors manage their inventory

SELECT

product\_id, product\_name, stock\_quantity

FROM products

WHERE stock\_quantity < 10;

50. Show the last 10 orders placed on the platform, including customer details, items purchased, and order amounts.

SELECT

o.order\_id, o.order\_date, c.customer\_name, p.product\_name, o.quantity, (o.quantity \* p.price) AS total\_amount

FROM orders o

JOIN customers c ON o.customer\_id = c.customer\_id

JOIN products p ON o.product\_id = p.product\_id

ORDER BY o.order\_date DESC

LIMIT 10;

### Results and Discussion

Successfully put in place a dashboard with functions including order tracking, real-time notifications, and product listings.

Determine what needs to be improved, like adding a recommendation engine.

**Chapter 4**

# Engineering Standards and Mapping

Every chapter should start with 1-2 sentences on the outline of the chapter.

### Impact on Society, Environment and Sustainability

#### Impact on Life

Better revenue streams for vendors and easier access to a variety of products for consumers.

#### Impact on Society & Environment

Encourages internet trade and lessens reliance on physical marketplaces.

#### Ethical Aspects

* Make sure user data is handled securely.
* Put anti-fraud procedures in place to protect the integrity of the platform.

#### Sustainability Plan

* Scalability can be ensured by using cloud-based technologies.
* Update frequently to satisfy consumer requests.

### Project Management and Teamwork

This multivendor e-commerce dashboard project was completed successfully thanks in large part to efficient project management and teamwork. To guarantee the project's timely and effective completion, the procedure comprised clearly identifying responsibilities, phases, and deliverables. An outline of the dynamics of teamwork and project management can be found below.

A well-organized and useful multivendor e-commerce system was the outcome of the collaboration and project management techniques. The group produced a scalable and user-friendly solution while effectively resolving obstacles and meeting deadlines.

### Complex Engineering Problem

#### Mapping of Program Outcome

This section assesses the degree to which the project's elements support the declared Program Outcomes (POs):

Table 4.1: Justification of Program Outcomes

| **PO’s** | **Justification** |
| --- | --- |
| PO1 | The application of theoretical knowledge in a practical setting is demonstrated by developing a working e-commerce system with efficient database queries. |
| PO2 | Critical analysis is demonstrated by addressing scalability and performance under high demand. |
| PO3 | Creating a dashboard and modular architecture requires organized design and resolving competing criteria. |

#### Complex Problem Solving

In this section, provide a mapping with problem solving categories. For each mapping add subsections to put rationale (Use Table [4.2).](#_heading=h.2dlolyb) For P1, you need to put another mapping with

Chapter 4. Engineering Standards and Mapping 4.3. Complex Engineering Problem

Knowledge profile and rational thereof.

| **EP1**  Depth of Knowledge | **EP2**  Range of Conflicting Requirements | **EP3**  Depth of Analysis | **EP4**  Familiarity of Issues | **EP5**  Extent of Applicable Codes | **EP6**  Extent  Of Stakeholder Involvement | **EP7**  Inter- dependence |
| --- | --- | --- | --- | --- | --- | --- |
| ✔ | ✔ | ✘ | ✔ | ✘ | ✔ | ✔ |

Table 4.2: Mapping with complex problem solving.

#### Engineering Activities

In this section, provide a mapping with engineering activities. For each mapping add subsections to put rationale (Use Table [4.3).](#_heading=h.1rvwp1q)

| **EA1**  Range of resources | **EA2**  Level of Interaction | **EA3**  Innovation | **EA4**  Consequences for society and  environment | **EA5**  Familiarity |
| --- | --- | --- | --- | --- |
| ✔ | ✘ | ✔ | ✔ | ✘ |

Table 4.3: Mapping with complex engineering activities.

**Chapter 5**

# Conclusion

Every chapter should start with 1-2 sentences on the outline of the chapter.

### Summary

In order to increase platform efficiency, the project created a strong multivendor e-commerce dashboard that effectively addressed vendor scalability, simplified order management, and improved functionality for administrators, sellers, and customers.

### Limitation

* Limited use of AI-based suggestions and advanced analytics.
* More optimization is needed to handle very huge datasets.

### Future Work

In the future we used machine learning techniques to give people tailored product recommendations. To improve platform accessibility and guarantee a flawless user experience on smartphones and tablets, develop a mobile application.

# References

[1] Jon Kleinberg and Eva Tardos. Algorithm Design. Pearson Education India, 2006.

[2] Martin Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.

[3] Canva. "Design Platform for Visual Content." Accessed December 2024.<https://www.canva.com>