E-COMMERCE SYSTEM

Project Report

Submitted for CSD317: Introduction To Database Systems

Name of Instructor: Dr. Sonia Khetarpaul

Members:

Dhanya Girdhar – 2310110463, dg218

Aastha Malhotra- 2310110007, am565

Raashi Sharma- 2310110566, rs708

1. PROJECT OVERVIEW:

This eCommerce website is a full-stack web application developed using **PHP**, **MySQL (via XAMPP)**, **HTML**, **CSS**, and **JavaScript**. It allows users to register, browse products, manage their cart, place orders, and participate in interactive trivia quizzes. Sellers can list products, track orders, manage stock, and view insights on sales and customer behavior. The platform ensures a seamless shopping experience with dynamic front-end interactions and a robust, normalized backend database.

1. OBJECTIVES AND FUNCTIONS:

The objective of this eCommerce project is to develop a secure, user-friendly platform that enables seamless customer and seller interactions.

**1. Customer Dashboard:**

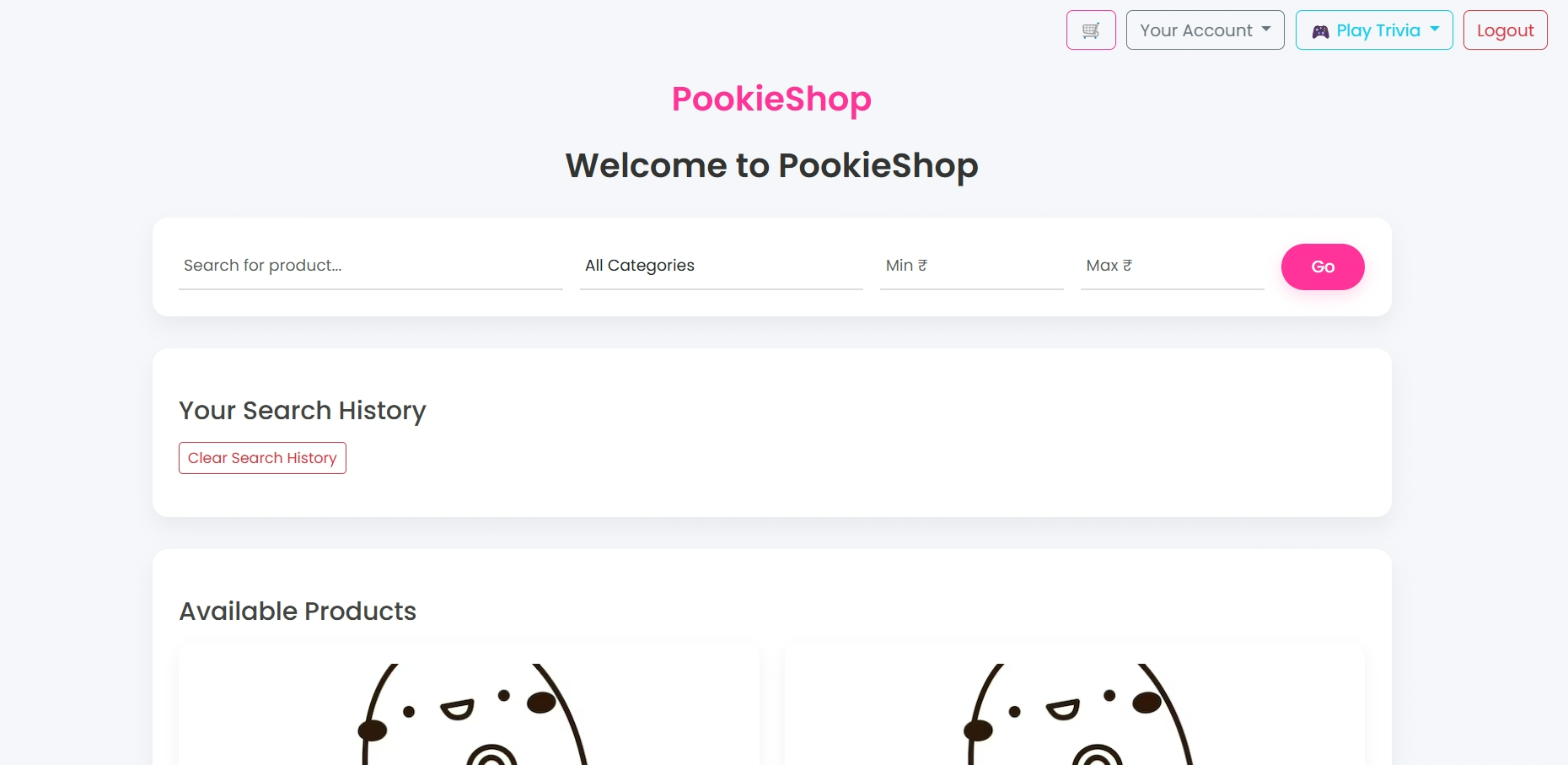
**Functions:**

1. **Register**: Allows a new customer to create an account by entering personal details (name, email, password, etc.).
2. **Login**: Authenticates a customer using their email and password.
3. **Customer dashboard**: displays all the products, search history
4. **Your cart**: displays all the products added to cart
5. **Your account:** displays options to view orders, order summary and return requests
6. **Your Order:** displays all the items/products order so far
7. **Your summary:** displays Total Orders, Total amount spent, Average Order Value and Most purchased product
8. **Your Return Request:** displays all the return request made and their status
9. **Play Trivia:** option to answer trivia question based on chosen theme and get discount on the total cart order
10. **Top Customer Discount:** customers get top customer discount if they are the top customer of any seller when they purchase any item from their inventory

**2.** **Seller Dashboard :**

**Functions:**

1. **Register**: Allows a new seller to create an account by entering their business details (name, email, password, etc.).
2. **Login**: Authenticates the seller using their credentials.
3. **View Inventory**: Displays a list of all products the seller is offering along with stock levels, prices, and other product details.
4. **Add New Product**: Allows the seller to add new products to their inventory, specifying product details like name, price, description, and stock level.
5. **Update Product**: Enables the seller to modify the details of an existing product, including price, stock quantity, or description.
6. **Delete Product**: Allows the seller to remove a product from their inventory.
7. **Track Sales**: Displays a summary of sales statistics, including revenue, quantity sold, and top-selling products.
8. **Manage Orders**: Displays the list of orders placed for the seller’s products, including customer name, order id, and total amount.
9. **Customer Reviews:** the sellers can view ratings and comments on the products along with customer name and date of review.
10. **Pending Return Request:** can approve/reject return requests.
11. **Low Stock Alerts:** give warning when the stock quantity is less than 5.
12. **Monthly Revenue Graph:** sellers can see their revenue generated each month.
13. WEBSITE OVERVIEW:
14. CUSTOMER DASHBOARD:



YOUR ORDERS:

**A screenshot of a chat

AI-generated content may be incorrect.**

YOUR ORDER SUMMARY:

**A screenshot of a computer

AI-generated content may be incorrect.**

YOUR CART:

**A screenshot of a computer

AI-generated content may be incorrect.**

YOUR RETURN REQUEST:

1. BEFORE SELLER APPROVES RETURN REQUEST

**A screenshot of a computer screen

AI-generated content may be incorrect.**

1. THE SELLER APPROVES THE RETURN REQUEST:

**A screenshot of a computer

AI-generated content may be incorrect.**

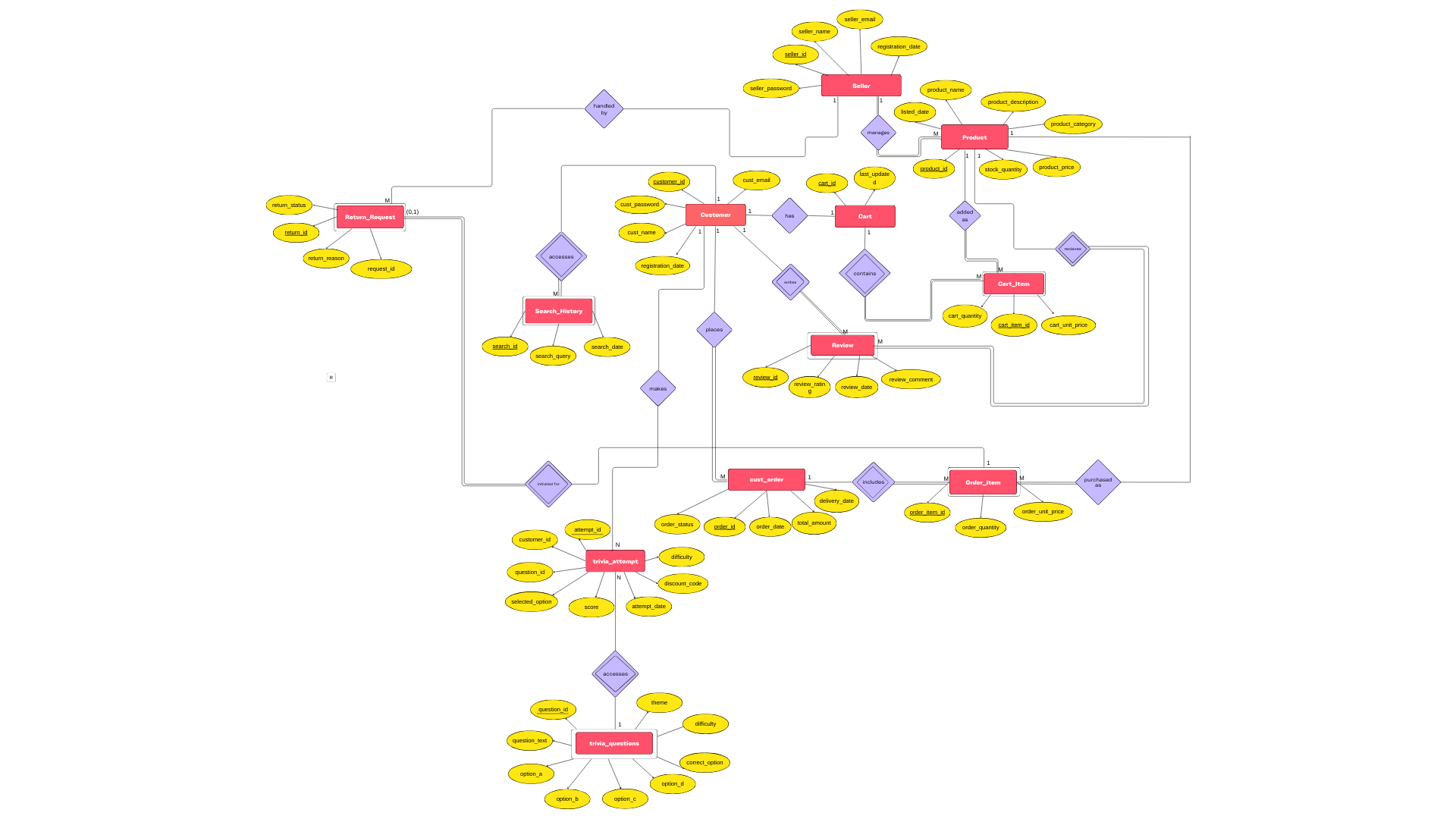
1. AFTER THE SELLER APPROVES THE RETURN REQUEST:

**A screenshot of a computer screen

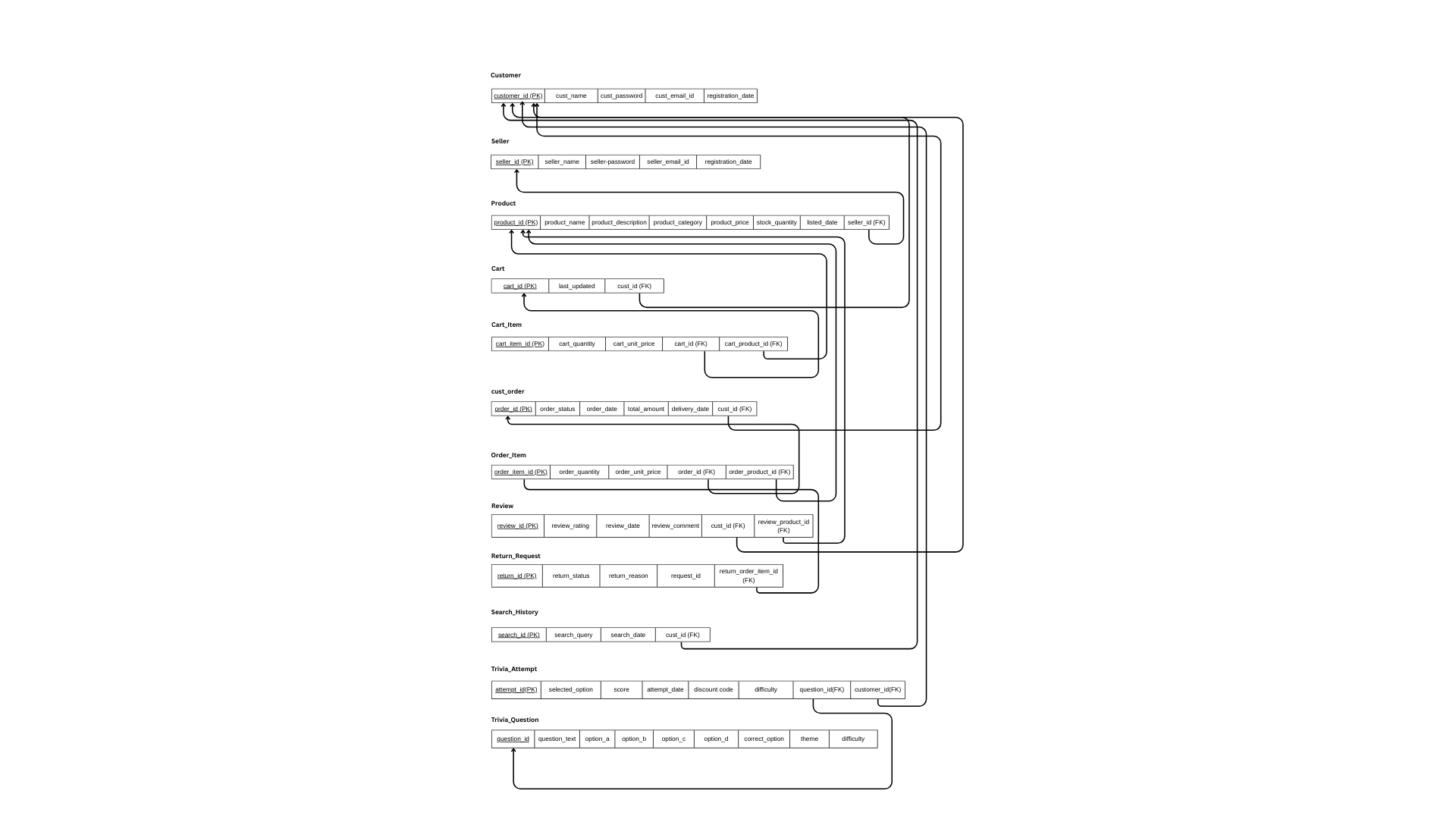
AI-generated content may be incorrect.**

1. SELLER DASHBOARD:  
     
   A screenshot of a computer

   AI-generated content may be incorrect.
2. ER DIAGRAM:



1. RELATION MODEL:



1. NORMALIZATION:

Assumptions Taken:

1. Email IDs are unique for all customers and sellers.
2. A customer can only have one cart.
3. There is one review per customer per product.
4. The prices of products do not change, they remain the same.
5. A product can appear only once in a cart. (quantity increases if a product is added multiple times).
6. A product can appear only once in an order. (quantity increases if a product is added multiple times).
7. A customer can have multiple orders.
8. An order will have one return request only.
9. A customer can have multiple search queries.

1. Customer Table:

* Attributes: customer\_id (PK), cust\_name, cust\_password, cust\_email, registration\_date
* Functional Dependencies:
* customer\_id →cust\_name, cust\_password, cust\_email, registration\_date
* cust\_email→ customer\_id, cust\_name, cust\_password, registration\_date
* Candidate Keys: customer\_id, cust\_email
* Highest Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

2. Seller Table:

* Attributes: seller\_id (PK), seller\_name, seller\_password, seller\_email, registration\_date
* Functional Dependencies:
* seller\_id → seller\_name, seller\_password, seller\_email, registration\_date
* seller\_email → seller\_id, seller\_name, seller\_password, registration\_date
* Candidate Keys: seller\_id, seller\_email
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

3. Product Table:

* Attributes: product\_id (PK), product\_name, product\_description, product\_category, product\_price, stock\_quantity, listed\_date, seller\_id (FK)
* Functional Dependencies:
* product\_id → product\_name, product\_description, product\_category, product\_price, stock\_quantity, listed\_date, seller\_id
* Candidate Keys: product\_id
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

4. Cart Table:

* Attributes: cart\_id (PK), last\_updated, cust\_id (FK)
* Functional Dependencies: o cart\_id → last\_updated, cust\_id o cust\_id → cart\_id
* Candidate Keys: cart\_id, cust\_id
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

5. Cart\_Item Table:

* Attributes: cart\_item\_id (PK), cart\_quantity, cart\_unit\_price, cart\_id (FK), cart\_product\_id (FK)
* Functional Dependencies:
* cart\_item\_id → cart\_quantity, cart\_unit\_price, cart\_id, cart\_product\_id
* (cart\_id, cart\_product\_id) → cart\_item\_id, cart\_quantity, cart\_unit\_price (assuming product can appear only once in a cart)
* Candidate Keys: cart\_item\_id, (cart\_id, cart\_product\_id)
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

6. Cust\_Order Table:

* Attributes: order\_id (PK), order\_status, order\_date, total\_amount, delivery\_date cust\_id (FK)
* Functional Dependencies:
* order\_id → order\_status, order\_date, total\_amount, cust\_id, delivery\_date
* Candidate Keys: order\_id
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

7. Order\_Item Table:

* Attributes: order\_item\_id (PK), order\_quantity, order\_unit\_price, order\_id (FK), order\_product\_id (FK)
* Functional Dependencies:
* order\_item\_id → order\_quantity, order\_unit\_price, order\_id, order\_product\_id
* (order\_id, order\_product\_id) → order\_item\_id, order\_quantity, order\_unit\_price
* Candidate Keys: order\_item\_id, (order\_id, order\_product\_id)
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

8. Review Table:

* Attributes: review\_id (PK), review\_rating, review\_date, review\_comment, cust\_id (FK), review\_product\_id (FK)
* Functional Dependencies:
* review\_id → rating, review\_date, review\_comment, cust\_id, review\_product\_id
* (cust\_id, review\_product\_id) → review\_id, review\_rating, review\_date, review\_comment
* Candidate Keys: review\_id, (cust\_id, review\_product\_id)
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

9. Return\_Request Table:

* Attributes: return\_id (PK), return\_status, return\_reason, request\_id, return\_order\_item\_id (FK)
* Functional Dependencies:
* return\_id → return\_status, return\_reason, request\_id, return\_order\_item\_id
* return\_order\_item\_id → return\_id
* Candidate Keys: return\_id, return\_order\_item\_id
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

10. Search\_History Table:

* Attributes: search\_id (PK), search\_query, search\_date, cust\_id (FK)
* Functional Dependencies:
* search\_id → search\_query, search\_date, cust\_id
* Candidate Keys: search\_id
* Normal Form: BCNF (no partial dependencies, no transitive dependencies, all determinants are candidate keys)

11. trivia\_questions Table:

* Attributes: question\_id (PK), question\_text, option\_a, option\_b, option\_c, option\_d, correct\_option, difficulty, theme
* Functional Dependencies:
* question\_id → question\_text, option\_a, option\_b, option\_c, option\_d, correct\_option, difficulty, theme
* Candidate Keys: question\_id
* Normal Form: BCNF (All determinants are candidate keys, no partial or transitive dependencies)

12. trivia\_attempts Table:

* Attributes: attempt\_id (PK), customer\_id (FK), question\_id (FK), selected\_option, score, attempt\_date, discount\_code, difficulty
* Functional Dependencies:
* attempt\_id → customer\_id, question\_id, selected\_option, score, attempt\_date, discount\_code, difficulty
* (customer\_id, question\_id, attempt\_date) → attempt\_id, selected\_option, score, discount\_code, difficulty
* Candidate Keys:
* attempt\_id
* (customer\_id, question\_id, attempt\_date) : a customer can only attempt a specific question once per day
* Normal Form: 3NF, but possibly not BCNF

1. SQL QUERIES:

1. TABLES:

CREATE TABLE `cart` (

`cart\_id` primary key int(11) NOT NULL auto\_increment

`last\_updated` date DEFAULT curdate(),

`cust\_id` int(11) NOT NULL

);

ALTER TABLE `cart`

ADD CONSTRAINT `cart\_ibfk\_1` FOREIGN KEY (`cust\_id`) REFERENCES `customer` (`customer\_id`) ON DELETE CASCADE;

CREATE TABLE `cart\_item` (

`cart\_item\_id` primary key int(11) NOT NULL auto\_increment

`cart\_id` int(11) NOT NULL,

`cart\_product\_id` int(11) NOT NULL,

`cart\_quantity` int(11) NOT NULL CHECK (`cart\_quantity` > 0),

`cart\_unit\_price` decimal(10,2) NOT NULL CHECK (`cart\_unit\_price` >= 0)

);

ALTER TABLE `cart\_item`

ADD CONSTRAINT `cart\_item\_ibfk\_1` FOREIGN KEY (`cart\_id`) REFERENCES `cart` (`cart\_id`) ON DELETE CASCADE,

ADD CONSTRAINT `cart\_item\_ibfk\_2` FOREIGN KEY (`cart\_product\_id`) REFERENCES `product` (`product\_id`) ON DELETE CASCADE;

CREATE TABLE `customer` (

`customer\_id` primary key int(11) NOT NULL auto\_increment

`cust\_name` varchar(255) NOT NULL,

`cust\_email` varchar(255) NOT NULL,

`cust\_password` varchar(255) NOT NULL,

`registration\_date` date DEFAULT curdate()

);

CREATE TABLE `cust\_order` (

`order\_id` primary key int(11) NOT NULL auto\_increment

`cust\_id` int(11) NOT NULL,

`order\_date` date DEFAULT curdate(),

`total\_amount` decimal(10,2) NOT NULL CHECK (`total\_amount` >= 0),

`order\_status` enum('Pending','Shipped','Delivered','Cancelled') NOT NULL DEFAULT 'Pending',

`delivery\_date` date DEFAULT NULL

);

ALTER TABLE `cust\_order`

ADD CONSTRAINT `cust\_order\_ibfk\_1` FOREIGN KEY (`cust\_id`) REFERENCES `customer` (`customer\_id`) ON DELETE CASCADE;

CREATE TABLE `order\_item` (

`order\_item\_id` primary key int(11) NOT NULL auto\_increment

`order\_id` int(11) NOT NULL,

`order\_product\_id` int(11) NOT NULL,

`order\_quantity` int(11) NOT NULL CHECK (`order\_quantity` > 0),

`order\_unit\_price` decimal(10,2) NOT NULL CHECK (`order\_unit\_price` >= 0)

);

ALTER TABLE `order\_item`

ADD CONSTRAINT `order\_item\_ibfk\_1` FOREIGN KEY (`order\_id`) REFERENCES `cust\_order` (`order\_id`) ON DELETE CASCADE,

ADD CONSTRAINT `order\_item\_ibfk\_2` FOREIGN KEY (`order\_product\_id`) REFERENCES `product` (`product\_id`) ON DELETE CASCADE;

CREATE TABLE `product` (

`product\_id` primary key int(11) NOT NULL auto\_increment

`product\_name` varchar(255) NOT NULL,

`product\_description` text DEFAULT NULL,

`product\_category` varchar(100) NOT NULL,

`product\_price` decimal(10,2) NOT NULL CHECK (`product\_price` > 0),

`stock\_quantity` int(11) NOT NULL CHECK (`stock\_quantity` >= 0),

`listed\_date` date DEFAULT curdate(),

`seller\_id` int(11) NOT NULL

);

ALTER TABLE `product`

ADD CONSTRAINT `product\_ibfk\_1` FOREIGN KEY (`seller\_id`) REFERENCES `seller` (`seller\_id`) ON DELETE CASCADE;

CREATE TABLE `return\_request` (

`return\_id` primary key int(11) NOT NULL auto\_increment

`return\_order\_item\_id` int(11) NOT NULL,

`request\_id` varchar(50) NOT NULL,

`return\_reason` text NOT NULL,

`return\_status` enum('Pending','Approved','Rejected') DEFAULT 'Pending'

);

ALTER TABLE `return\_request`

ADD CONSTRAINT `return\_request\_ibfk\_1` FOREIGN KEY (`return\_order\_item\_id`) REFERENCES `order\_item` (`order\_item\_id`) ON DELETE CASCADE;

CREATE TABLE `review` (

`review\_id` primary key int(11) NOT NULL auto\_increment

`cust\_id` int(11) NOT NULL,

`review\_product\_id` int(11) NOT NULL,

`review\_rating` int(11) NOT NULL CHECK (`review\_rating` between 1 and 5),

`review\_date` date DEFAULT curdate(),

`review\_comment` text DEFAULT NULL

);

ALTER TABLE `review`

ADD CONSTRAINT `review\_ibfk\_1` FOREIGN KEY (`cust\_id`) REFERENCES `customer` (`customer\_id`) ON DELETE CASCADE,

ADD CONSTRAINT `review\_ibfk\_2` FOREIGN KEY (`review\_product\_id`) REFERENCES `product` (`product\_id`) ON DELETE CASCADE;

CREATE TABLE `search\_history` (

`search\_id` primary key int(11) NOT NULL auto\_increment

`cust\_id` int(11) NOT NULL,

`search\_query` varchar(255) NOT NULL,

`search\_date` date DEFAULT curdate()

);

ALTER TABLE `search\_history`

ADD CONSTRAINT `search\_history\_ibfk\_1` FOREIGN KEY (`cust\_id`) REFERENCES `customer` (`customer\_id`) ON DELETE CASCADE;

CREATE TABLE `seller` (

`seller\_id` primary key int(11) NOT NULL auto\_increment

`seller\_name` varchar(255) NOT NULL,

`seller\_email` varchar(255) NOT NULL,

`seller\_password` varchar(255) NOT NULL,

`registration\_date` date DEFAULT curdate()

);

CREATE TABLE `trivia\_attempts` (

`attempt\_id` primary key int(11) NOT NULL auto\_increment

`customer\_id` int(11) NOT NULL,

`question\_id` int(11) NOT NULL,

`selected\_option` enum('A','B','C','D') DEFAULT NULL,

`score` int(11) DEFAULT 0,

`attempt\_date` date DEFAULT curdate(),

`discount\_code` varchar(20) DEFAULT NULL,

`difficulty` enum('Easy','Medium','Hard') DEFAULT NULL

);

ALTER TABLE `trivia\_attempts`

ADD CONSTRAINT `trivia\_attempts\_ibfk\_1` FOREIGN KEY (`customer\_id`) REFERENCES `customer` (`customer\_id`),

ADD CONSTRAINT `trivia\_attempts\_ibfk\_2` FOREIGN KEY (`question\_id`) REFERENCES `trivia\_questions` (`question\_id`);

CREATE TABLE `trivia\_questions` (

`question\_id` primary key int(11) NOT NULL auto\_increment

`question\_text` text NOT NULL,

`option\_a` varchar(255) DEFAULT NULL,

`option\_b` varchar(255) DEFAULT NULL,

`option\_c` varchar(255) DEFAULT NULL,

`option\_d` varchar(255) DEFAULT NULL,

`correct\_option` enum('A','B','C','D') NOT NULL,

`difficulty` enum('Easy','Medium','Hard') NOT NULL,

`theme` varchar(100) NOT NULL

);

2. E\_COMMERCE.PHP( THE CUSTOMER DASHBOARD):

1. Fetch Distinct Product Categories

SELECT DISTINCT product\_category FROM Product;

1. Dynamic Product Search with Optional Filters

Base query:

SELECT \* FROM Product WHERE 1=1

Optional appended conditions (based on user input):

AND product\_name LIKE :search\_query

AND product\_category = :category

AND product\_price >= :min\_price

AND product\_price <= :max\_price

1. Insert User Search into Search History

INSERT INTO Search\_History (cust\_id, search\_query)

VALUES (:cust\_id, :search\_query);

1. Fetch User's Search History

SELECT search\_query, search\_date

FROM Search\_History

WHERE cust\_id = :cust\_id

ORDER BY search\_date DESC;

3. ORDERS.PHP ( DISPLAYS CUSTOMER’S ORDERS ) :

1. Fetch Orders for the Current User

SELECT

o.order\_id,

o.order\_date,

o.delivery\_date,

o.total\_amount,

o.order\_status,

p.product\_name,

p.product\_price AS original\_price,

oi.order\_item\_id,

oi.order\_quantity,

oi.order\_unit\_price,

oi.order\_product\_id

FROM cust\_order o

JOIN order\_item oi ON o.order\_id = oi.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

WHERE o.cust\_id = ?

ORDER BY o.order\_date DESC;

1. Fetch order\_item\_ids for User's Orders

SELECT order\_item\_id

FROM order\_item

WHERE order\_id IN (?, ?, ?, ...);

1. Fetch Returned Order Items

SELECT return\_order\_item\_id

FROM return\_request

WHERE return\_order\_item\_id IN (?, ?, ?, ...);

1. Fetch Products Already Reviewed by the User

SELECT review\_product\_id

FROM review

WHERE cust\_id = ?;

4. ORDERS\_SUMMARY.PHP (DISPLAYS CUSTOMER’S ORDER

SUMMARY):

1. Fetch Total Orders, Total Spent, and Average Order Value

SELECT

COUNT(\*) AS total\_orders,

COALESCE(SUM(total\_amount), 0) AS total\_spent,

ROUND(AVG(total\_amount), 2) AS avg\_order\_value

FROM cust\_order

WHERE cust\_id = ?;

1. Fetch Total Amount of the Most Recent Order

SELECT total\_amount

FROM cust\_order

WHERE cust\_id = ?

ORDER BY order\_date DESC

LIMIT 1;

1. Fetch Most Frequently Bought Product

SELECT p.product\_name, SUM(oi.order\_quantity) AS qty

FROM order\_item oi

JOIN cust\_order co ON oi.order\_id = co.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

WHERE co.cust\_id = ?

GROUP BY oi.order\_product\_id

ORDER BY qty DESC

LIMIT 1;

5. RETURN\_REQUESTS (DISPLAYS CUSTOMER’S RETURN

REQUESTS):

Fetch Return Requests for the Logged-in Customer

SELECT

rr.return\_id,

rr.request\_id,

rr.return\_reason,

rr.return\_status,

co.order\_date,

co.total\_amount,

p.product\_name

FROM return\_request rr

JOIN order\_item oi ON rr.return\_order\_item\_id = oi.order\_item\_id

JOIN cust\_order co ON oi.order\_id = co.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

WHERE co.cust\_id = ?

ORDER BY rr.return\_id DESC;

6.LEADERBOARD.PHP (DISPLAYS TRIVIA LEADERBOARD):

SELECT c.cust\_name, SUM(t.score) AS total\_score

FROM trivia\_attempts t

JOIN customer c ON c.customer\_id = t.customer\_id

GROUP BY t.customer\_id

ORDER BY total\_score DESC

LIMIT 10;

7. CART.PHP (DISPLAYS PRODUCTS ADDED TO CART):

1. Fetch Cart Items with Stock Info and Seller ID:

SELECT ci.cart\_item\_id, p.product\_name, ci.cart\_quantity, ci.cart\_unit\_price, p.stock\_quantity, p.seller\_id

FROM Cart\_Item ci

JOIN Product p ON ci.cart\_product\_id = p.product\_id

JOIN Cart c ON ci.cart\_id = c.cart\_id

WHERE c.cust\_id = ?

1. Get Top Customer per Seller:

SELECT o.seller\_id, o.cust\_id

FROM (

SELECT p.seller\_id, co.cust\_id, SUM(coi.order\_quantity \* coi.order\_unit\_price) AS total\_spent

FROM cust\_order co

JOIN order\_item coi ON co.order\_id = coi.order\_id

JOIN product p ON p.product\_id = coi.order\_product\_id

GROUP BY p.seller\_id, co.cust\_id

) o

INNER JOIN (

SELECT seller\_id, MAX(total\_spent) AS max\_spent

FROM (

SELECT p.seller\_id, co.cust\_id, SUM(coi.order\_quantity \* coi.order\_unit\_price) AS total\_spent

FROM cust\_order co

JOIN order\_item coi ON co.order\_id = coi.order\_id

JOIN product p ON p.product\_id = coi.order\_product\_id

GROUP BY p.seller\_id, co.cust\_id

) AS totals

GROUP BY seller\_id

) top ON o.seller\_id = top.seller\_id AND o.total\_spent = top.max\_spent

8. SellerDashboard.PHP (THE SELLER DASHBOARD):

1. Update Stock and Notify Customers (if stock goes from 0 → positive):

SELECT stock\_quantity FROM product WHERE product\_id = ? AND seller\_id = ?

UPDATE product SET stock\_quantity = ? WHERE product\_id = ? AND seller\_id = ?

UPDATE cart\_item

SET cart\_quantity = 1

WHERE cart\_product\_id = :product\_id AND cart\_quantity = 0

1. Fetch All Products for the Seller:

SELECT \* FROM product WHERE seller\_id = ?

1. Top-Selling Product:

SELECT p.product\_name, SUM(oi.order\_quantity) AS total\_sales

FROM order\_item oi

JOIN product p ON oi.order\_product\_id = p.product\_id

JOIN cust\_order o ON o.order\_id = oi.order\_id

WHERE p.seller\_id = ?

GROUP BY p.product\_id

ORDER BY total\_sales DESC

LIMIT 1

1. Top Customer for the Seller:

SELECT o.cust\_id, cu.cust\_name, SUM(oi.order\_quantity \* oi.order\_unit\_price) AS total\_spent

FROM order\_item oi

JOIN cust\_order o ON oi.order\_id = o.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

JOIN customer cu ON o.cust\_id = cu.customer\_id

WHERE p.seller\_id = ?

GROUP BY o.cust\_id

ORDER BY total\_spent DESC

LIMIT 1

1. Total Unique Customers:

SELECT COUNT(DISTINCT o.cust\_id) AS total\_customers

FROM cust\_order o

JOIN order\_item oi ON o.order\_id = oi.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

WHERE p.seller\_id = ?

1. Total Transactions & Revenue:

SELECT COUNT(DISTINCT o.order\_id) AS total\_transactions,

SUM(oi.order\_quantity \* oi.order\_unit\_price) AS total\_spent

FROM order\_item oi

JOIN cust\_order o ON oi.order\_id = o.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

WHERE p.seller\_id = ?

1. Recent Transactions (Last 5 Orders):

SELECT o.order\_id, cu.cust\_name, SUM(oi.order\_quantity \* oi.order\_unit\_price) AS total\_spent

FROM order\_item oi

JOIN cust\_order o ON oi.order\_id = o.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

JOIN customer cu ON o.cust\_id = cu.customer\_id

WHERE p.seller\_id = ?

GROUP BY o.order\_id

ORDER BY o.order\_date DESC

LIMIT 5

1. Monthly Revenue Chart:

SELECT

DATE\_FORMAT(co.order\_date, '%Y-%m') AS month,

SUM(oi.order\_quantity \* oi.order\_unit\_price) AS total\_revenue

FROM cust\_order co

JOIN order\_item oi ON co.order\_id = oi.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

WHERE p.seller\_id = :seller\_id

GROUP BY month

ORDER BY month;

1. Sales Summary (Units Sold & Revenue):

SELECT

COUNT(DISTINCT o.order\_id) AS total\_orders,

SUM(oi.order\_quantity) AS total\_units\_sold,

SUM(oi.order\_quantity \* oi.order\_unit\_price) AS total\_revenue

FROM order\_item oi

JOIN product p ON oi.order\_product\_id = p.product\_id

JOIN cust\_order o ON o.order\_id = oi.order\_id

WHERE p.seller\_id = ?

1. Product Reviews for Seller:

SELECT

r.review\_id,

r.review\_product\_id,

p.product\_name,

r.review\_rating,

r.review\_date,

r.review\_comment,

c.cust\_name

FROM review r

JOIN product p ON r.review\_product\_id = p.product\_id

JOIN customer c ON r.cust\_id = c.customer\_id

WHERE p.seller\_id = ?

ORDER BY r.review\_date DESC;

1. Pending Return Requests:

SELECT rr.\*, cu.cust\_name, p.product\_name

FROM return\_request rr

JOIN order\_item oi ON rr.return\_order\_item\_id = oi.order\_item\_id

JOIN cust\_order o ON oi.order\_id = o.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

JOIN customer cu ON o.cust\_id = cu.customer\_id

WHERE rr.return\_status = 'Pending' AND p.seller\_id = ?

GROUP BY rr.return\_id;

9. ADD\_PRODUCT.PHP (SELLER INSERTS NEW PRODUCT):

Insert New Product Query

INSERT INTO product (

product\_name,

product\_category,

product\_description,

product\_price,

stock\_quantity,

listed\_date,

seller\_id

)

VALUES (?, ?, ?, ?, ?, CURDATE(), ?);

1. UPDATING\_PRODUCT.PHP (SELLER UPDATES PRODUCT):

Update Product Details Query

UPDATE product

SET

product\_name = ?,

product\_price = ?,

stock\_quantity = ?

WHERE

product\_id = ?

AND seller\_id = ?;

11. handleReturnRequest.PHP (APPROVING/REJECTING

RETURN REQUEST) :

Update Return Request Status

UPDATE return\_request

SET return\_status = ?

WHERE return\_id = ?;

12. DELETE\_PRODUCT.PHP (SELLER DELETED PRODUCTS):

Delete Product Query

DELETE FROM product WHERE product\_id = ? AND seller\_id = ?;

13. SellerAuth.PHP (SELLER AUTHORIZATION AND

REGISTRATION) :

1. Seller Login Query

SELECT \* FROM seller WHERE seller\_email = :email;

1. Seller Registration Query

INSERT INTO seller (seller\_name, seller\_email, seller\_password, registration\_id)

VALUES (?, ?, ?, ?);

14. CustomerAuth.PHP (CUSTOMER AUTHORIZATION AND REGISTRATION) :

1. Login Query

SELECT \* FROM customer WHERE cust\_email = ?;

1. Check if Email Exists

SELECT customer\_id FROM customer WHERE cust\_email = ?;

1. Register New Customer

INSERT INTO customer (cust\_name, cust\_email, cust\_password, registration\_date)

VALUES (?, ?, ?, NOW());

15. clear\_search\_history.php:

This ajax handler is a simple handler that clears all search history for the current logged-in user by passing the following query:

DELETE FROM Search\_History WHERE cust\_id = ?

16. handle\_return\_request.php:

This ajax handler handles the return requests whenever the current logged in customer requests return on a particular order item.

SQL Queries:

SELECT oi.\*, o.order\_status

FROM order\_item oi

JOIN cust\_order o ON oi.order\_id = o.order\_id

WHERE oi.order\_item\_id = ? AND o.order\_status = 'Delivered'

SELECT \* FROM return\_request WHERE return\_order\_item\_id = ?

INSERT INTO return\_request (return\_order\_item\_id, request\_id, return\_reason) VALUES (?, ?, ?)

Logic Implemented:

* First it makes sure the order in which the order item which is initiated for request belongs to is delivered, only then the return request can be generated. This is done by a basic JOIN query.
* To prevent duplicate return requests, it is checked whether a return request already exists for the current order it
* em.
* If not, using an insert query, a new return request is added to the database.

17.  loginAjax.php:

This ajax handler is used for customer login authorization. Directly CustomerAuth.php handles the authorization.

Additional SQL Query:

UPDATE cust\_order SET order\_status = 'Delivered' WHERE cust\_id = ? AND delivery\_date <= CURDATE() AND order\_status = 'Pending'

Logic Implementation:

* Customer credentials entered are validated using CustomerAuth.php
* If valid, starts a session and stores user information and updates the order\_status as ‘Delivered’ for pending orders if delivery date has passed.

18. place\_order.php:

This ajax handler is used for handling the entire ‘place order’ process in cart.php

SQL Queries:

SELECT cart\_id FROM Cart WHERE cust\_id =?

SELECT cart\_product\_id, cart\_quantity, cart\_unit\_price FROM Cart\_Item WHERE cart\_id = ?

SELECT \* FROM trivia\_attempts WHERE customer\_id = ? AND discount\_code = ? AND attempt\_date = CURDATE() AND score = 1

INSERT INTO cust\_order (cust\_id, order\_date, total\_amount, order\_status, delivery\_date) VALUES (?, NOW(), ?, 'Pending', ?)

INSERT INTO order\_item (order\_id, order\_product\_id, order\_quantity, order\_unit\_price) VALUES (?, ?, ?, ?)

UPDATE product SET stock\_quantity = stock\_quantity - ? WHERE product\_id = ? AND stock\_quantity >= ?

SELECT seller\_id, product\_name FROM product WHERE product\_id = ?

SELECT o.cust\_id FROM cust\_order o

JOIN order\_item oi ON o.order\_id = oi.order\_id

JOIN product p ON oi.order\_product\_id = p.product\_id

WHERE p.seller\_id = ?

GROUP BY o.cust\_id

ORDER BY SUM(oi.order\_quantity \* oi.order\_unit\_price) DESC LIMIT 1

DELETE FROM Cart\_Item WHERE cart\_id = ?

Logic Implementation:

* The cart\_id of the logged in customer is retrieved, from which the cart items present in the cart along with unit price and quantity is retrieved.
* The discount code (from the trivia attempts table) entered by the customer is validated.
* A new order record is inserted into the cust\_order table, and each cart item is added as order item.
* The stock quantity of the purchased product is reduced in the seller’s inventory by the quantity of the product purchased.
* Seller information for each product is retrieved (for seller specific discounts). Top customer for each seller is identified using a JOIN query. The cust\_order, order\_item and product table is joined and the top customer for that seller is retrieved by using ORDER BY DESC LIMIT 1. A loyalty discount system is implemented where the biggest spender for each seller gets additional discounts.
* The cart is emptied after an order is placed.

In this ajax handler, **transactions** are used to ensure that the entire order placement process either completes successfully or fails completely without partial changes to the database. This is crucial for maintaining data integrity in complex operations like order processing.

Transactions Implementation:

try {

    $conn->beginTransaction();

    // database operations

    $conn->commit();

    // Success response

} catch (Exception $e) {

    $conn->rollBack();

    // Error response

}

Multiple database operations are ran through the SQL queries mentioned above within the transaction itself. If all operations succeed, the transaction is committed, making all changes permanent in the database. Otherwise, the transaction is rolled back to the start.Through transaction logic, we can ensure that either the order is fully processed or not processed at all.

18. sellerLoginAjax.php:

This ajax handler handles seller authentication. Direct authentication is handled by the sellerAuth class.

Logic Implementation:

* Validates seller login credentials using SellerAuth class
* If valid, starts a session and stores seller information
* Returns JSON response indicating success or failure

19. sellerRegisterAjax.php

This ajax handler handles seller registration of new sellers.

SQL Query:

INSERT INTO seller (seller\_name, seller\_email, seller\_password, registration\_id) VALUES (:name, :email, :password, :registration)

Logic Implementation:

Collects seller details and and inserts new seller information in the database.

20. signupAjax.php:

This ajax handler handles customer registration of new sellers. It doesn’t have any sql queries as it is directly handler by CustomerAuth.php

Logic Implementation:

* Validates that all required fields are provided
* Uses CustomerAuth class to register the customer after validating that the email is not already in use.

21. submit\_review.php:

This ajax handler handles product review submission on order items.

SQL Queries:

SELECT \* FROM review WHERE cust\_id = ? AND review\_product\_id = ?

INSERT INTO review (cust\_id, review\_product\_id, review\_rating, review\_comment) VALUES (?, ?, ?, ?)

Logic Implementation:

* Firstly, it is checked if the currently logged in customer has already reviewed the product.
* If not, it inserts a new record in the reviews table.

22.  triviaAjax.php:

This ajax handler implements an interactive trivia game with discounts for correct answers.

SQL Queries:

SELECT \* FROM trivia\_attempts WHERE customer\_id = ? AND attempt\_date = CURDATE()

SELECT \* FROM trivia\_questions WHERE theme = ? AND difficulty = ? ORDER BY RAND() LIMIT 1

SELECT \* FROM trivia\_questions WHERE question\_id = ?

INSERT INTO trivia\_attempts (customer\_id, question\_id, attempt\_date, score, discount\_code, difficulty) VALUES (?, ?, CURDATE(), ?, ?, ?)

SELECT \* FROM trivia\_attempts WHERE customer\_id = ? AND discount\_code = ? AND attempt\_date = CURDATE() AND score = 1

SELECT c.cust\_name, SUM(score) AS total\_score

FROM trivia\_attempts t

JOIN customer c ON t.customer\_id = c.customer\_id

GROUP BY t.customer\_id

ORDER BY total\_score DESC LIMIT 10

Logic Implementation:

* A customer is only allowed to play trivia once a day, that is checked.
* A random question of the chosen theme and difficulty is selected from trivia\_questions table.
* Records the user's trivia attempt with results (used for maintaining leaderboard).
* For the correct answer to the question, a discount code is generated, which is validated during checkout.
* Using a basic JOIN query on trivia\_attemps and customer tables, the top 10 customers are retrieved on the basis of their total scores for the leaderboard.

23. cartAjax.php

This Ajax Handler handles the functionality of adding and removing items from your cart.

SQL Queries:

SELECT stock\_quantity FROM Products WHERE product\_id = :product\_id

SELECT cart\_id FROM Cart WHERE cust\_id = :cust\_id

INSERT INTO Cart (cust\_id) VALUES (:cust\_id)

SELECT \* FROM cart\_item WHERE cart\_id = :cart\_id AND cart\_product\_id = :product\_id

UPDATE cart\_item SET cart\_quantity = cart\_quantity + 1 WHERE cart\_item\_id = :cart\_item\_id

SELECT product\_price FROM Products WHERE product\_id = :product\_id

INSERT INTO cart\_item (cart\_id, cart\_product\_id, cart\_quantity, cart\_unit\_price) VALUES (:cart\_id, :product\_id, 1, :price)

DELETE FROM cart\_item WHERE cart\_item\_id = :cart\_item\_id

UPDATE cart\_item SET cart\_quantity = :qty WHERE cart\_item\_id = :cart\_item\_id

Logic Implemented:

* Stock quantity of a particular product is checked to make sure that an “out of stock” item isn’t added to cart.
* Since cart and customer have a 1:1 relationship, the ajax checks if there exists a cart that is referred to a customer. In case a cart doesn’t exist for a particular customer, it is added into the database which is referred to that customer.
* No duplicate entries of cart\_item are allowed in the cart, if the same product is added multiple times in the cart, the quantity is updated. A new item is added with quantity 1.
* The price of the product added in the cart is retrieved from the product table in the database.
* When an item is removed from the cart, the entry is deleted from the cart\_item table.
* The quantity of the cart item can be increased/decreased directly in the cart as well.

24. searchAjax.php

This ajax handler handles adding products to cart directly from search results.

SQL Queries:

SELECT cart\_id FROM Cart WHERE cust\_id = :cust\_id

INSERT INTO Cart (cust\_id) VALUES (:cust\_id)

SELECT cart\_item\_id, cart\_quantity

FROM cart\_item

WHERE cart\_id = :cart\_id AND cart\_product\_id = :product\_id

UPDATE cart\_item

SET cart\_quantity = :qty

WHERE cart\_item\_id = :item\_id

INSERT INTO cart\_item (cart\_id, cart\_product\_id, cart\_quantity, cart\_unit\_price)

VALUES (:cart\_id, :product\_id, 1, (SELECT product\_price FROM Product WHERE product\_id = :product\_id))

Logic Implementation:

The cart is retrieved for the logged-in customer. If a cart doesn’t exist for the particular customer a new record will be created.

If the product exists in the cart already, its quantity is increased when the item is added. If the item does not exist, a new record is created.

It is similar to the cartAjax.php but this only handles adding and not removing.

1. LIMITATIONS:
2. **No Real-Time Payment Integration**

The system does not currently support real-time transaction processing (e.g., via Razorpay, Stripe, or PayPal). As a result, all purchases and return requests are handled as simulated actions rather than actual financial transactions.

1. **Incomplete Return Workflow**

Although the return request functionality is implemented, approved returns are not automatically reflected in the inventory. This may lead to **inaccurate stock levels** over time and requires manual stock adjustments by the seller.

1. **Limited Role Management**

The system differentiates between customers and sellers, but lacks support for additional roles such as delivery personnel, admin users, or customer support agents.

1. **No Notification System**

Users and sellers do not receive email or SMS notifications for order confirmations, return status updates, or stock alerts, which limits the interactive experience.

1. REFERENCES:
2. PHP DOCUMENTATION: <https://www.php.net/docs.php>
3. PHP MANUAL: <https://www.php.net/manual/en/index.php>
4. PHP W3SCHOOLS TUTORIALS: <https://www.w3schools.com/php/>
5. SAMPLE FULL-STACK PROJECT FOR REFERENCE: <https://youtube.com/playlist?list=PLJ4-ETiGBrdOZ4kvbzNGidD26M24BLImM&si=qKjqZTM5PqBpR1DD>
6. XAMPP TUTORIAL: <https://www.tpointtech.com/xampp>
7. JAVASCRIPT: DOCUMENTATION: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
8. HTML/CSS: <https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics>, <https://www.w3schools.com/HTML/html_css.asp>