# Database Foundations for Business Analytics

E-commerce Product Recommendation System

# Group 14

**Adharsha Velen**(axg230033@utdallas.edu)

**Arwin Kumar Ravi**(axr220249@utdallas.edu)

**Barath Kumar Dhanasekar**([bxd220033@utdallas.edu](mailto:bxd220033@utdallas.edu))

**Jesse Jackson Yajjala** (jxy230003@utdallas.edu)

**Mahadevan Ramanan**(mxr230040@utdallas.edu)

**Prathamesh P Nagraj**([pxn230011@utdallas.edu](mailto:pxn230011@utdallas.edu))

**Vijay Refkin Puvvala** (vxp230008@utdallas.edu)

**Abstract**

This project report talks about the implementation phase of the E-Commerce Product Recommendation System, with the aim of providing user with recommendations, and creating triggers and stored procedures that are responsible for recording new data. The document commences with an overview of the logic design and modelling processes. Section 1 introduces the project, while Section 2 presents the relational schema. Section 3 talks about the triggers and stored procedures created, and the logic behind the same. Section 4 talks about how we can use SQL queries to analyse the dataset. The report concludes with a concise summary, briefly summarizing the database system created.

**table of contents**

**abstract**………………………………………………………………………………………………...2

**table of contents** ……………………...3

1. Introduction.……………………............................................................................................4

2. Relational Schema ……………………...4

3. Functions, Triggers and Stored Procedures ……………………...5

4. SQL Queries For Data Analysis ……………………...8

5. Conclusion ……………………...8

1. **INTRODUCTION:**

This project seeks to create an E-Commerce Product Recommendation System designed to enrich the shopping experience for users on an e-commerce platform. Our primary objective is to address the challenge of enhancing user engagement, boosting sales, and elevating the shopping experience through personalized product recommendations. This approach aims to stimulate revenue growth, reinforce user loyalty, and establish a competitive edge for the e-commerce platform. We also aim to manage the inventory, by keeping track of the product availability.

1. **RELATIONAL SCHEMA:**

A screenshot of a computer

Description automatically generated

Fig 2.1 – RELATIONAL SCHEMA

1. **FUNCTIONS, TRIGGERS AND STORED PROCEDURES:**
2. **Function for placing an order:** Function that takes different prompts and adds data to the orders table.

CREATE FUNCTION CreateOrder()

RETURNS INT

BEGIN

DECLARE userid INT;

DECLARE InproductID INT;

DECLARE quantity INT;

DECLARE totalcost DECIMAL(10,2);

DECLARE continueOrder BOOLEAN DEFAULT TRUE;

WHILE continueOrder DO

SET continueOrder = FALSE;

SET userid = EXISTS (SELECT \* FROM Customers WHERE customerid = INPUT('Enter your customer ID: '));

IF NOT user THEN

SELECT 'Invalid user ID!';

CONTINUE;

END IF;

SET InproductID = INPUT('Enter product ID (or leave blank to finish): ');

IF InproductID IS NULL THEN

LEAVE;

END IF;

SET InproductID = CONVERT\_INT(InproductID);

IF NOT EXISTS (SELECT \* FROM Products WHERE ProductID = InproductID) THEN

SELECT 'Invalid product ID!';

CONTINUE;

END IF;

SET price = (SELECT price FROM Products WHERE ProductID = productID);

-- Quantity and total cost

SET quantity = CONVERT\_INT(INPUT('Enter quantity: '));

SET totalcost = price \* quantity;

SELECT 'Total cost for this item:', totalCost;

-- Confirmation and order details

IF UPPER(INPUT('Do you want to add this item (y/N)? ')) = 'Y' THEN

INSERT INTO Orders (OrderID, CustomerId, ProductID, Quantity, Cost, OrderDate)

VALUES ((SELECT MAX(OrderID) + 1 FROM Orders), Customerid, InproductID, quantity, totalCost, SYSDATE);

-- Trigger stock update and order processing

CALL trigger\_placeorder();

SET continueOrder = True;

END IF;

END WHILE;

RETURN 0; -- Order successfully completed

END;

'

-- Return latest order ID or 0

RETURN IF lastOrderID IS NOT NULL THEN lastOrderID + 1 ELSE 0 END;

END;

1. **Trigger: Trigger\_placeorder():** This trigger is set to run after every insert into the orders table.   
     
   CREATE TRIGGER trigger\_placeorder

AFTER INSERT ON Orders

FOR EACH ROW

BEGIN

-- Update product stock

UPDATE Products

SET Stock = Stock - NEW.Quantity

WHERE ProductID = NEW.ProductID;

CALL ProcessOrders();

END;

1. **Stored Procedure - ProcessOrders:** This stored procedure processes the order, and displays the new order ID.

CREATE PROCEDURE ProcessOrders(newOrderID INT)

BEGIN

SELECT 'Order placed successfully! Order ID:', newOrderID;

END;

1. **Trigger – trigger\_AddProduct:** This trigger is set to run every time a new product has been added to the products table.  
     
   CREATE TRIGGER trigger\_AddProduct

AFTER INSERT ON Products

FOR EACH ROW

BEGIN

DECLARE ProductName VARCHAR(255);

DECLARE stock\_quantity INT;

DECLARE ProductID INT ;

DECLARE Price DECIMAL(10,2);

DECLARE CategoryID INT ;

DECLARE CategoryName VARCHAR(255);

SET ProductName = NEW.ProductName;

SET stock\_quantity = NEW.stock\_quantity;

SET ProductID = NEW.ProductID;

SET Price - NEW.Price ;

SET CategoryID = NEW.CategoryID;

SET CategoryName = NEW.CategoryName;

CALL AddProduct(ProductName, stock\_quantity, ProductID,Price, CategoryID, CategoryName);

END;

1. **Stored Procedure – Add\_Product:** Set of commands that help in inserting data into the products table.  
   CREATE PROCEDURE AddProduct(  
       IN p\_ProductName VARCHAR(255),  
       IN p\_stock\_quantity INT,  
       IN p\_ProductID INT,  
       IN p\_Price DECIMAL(10,2),  
       IN p\_CategoryID INT,  
       IN p\_CategoryName VARCHAR(255)  
   )  
   BEGIN  
       -- Insert into Orders table  
       INSERT INTO products (ProductID, Quantity,p\_stock\_quantity, Cost)  
       VALUES (p\_ProductName, p\_ProductID,p\_stock\_quantity,p\_Price);  
       -- Update stock size in Products table  
       UPDATE Products  
       SET StockSize = StockSize + p\_stock\_quantity  
       WHERE ProductID = p\_ProductID;  
   END
2. **Trigger: AfterInsertRecommendation:** This trigger is used to run the stored procedure for calculating the recommendations score for each user, for a particular product.  
     
   DELIMITER //

CREATE TRIGGER AfterInsertRecommendation  
AFTER INSERT  
ON interaction FOR EACH ROW  
BEGIN  
    -- Call the stored procedure to calculate and update RecommendationScore  
    CALL CalculateRecommendationScore(NEW.User\_ID, NEW.Product\_ID);  
END //  
DELIMITER ;

1. **Stored Procedure – CalculateRecommendationScore:** This stored procedure is used to calculate the recommendation score for each user.  
   CREATE DEFINER=`root`@`localhost` PROCEDURE `CalculateRecommendationScore`(IN p\_UserID INT, IN p\_ProductID INT)  
   BEGIN  
       DECLARE mean\_score DECIMAL(5, 2);  
       DECLARE interaction\_id\_value INT;

    -- Calculate the mean of interaction weightages for the specified User\_ID and Product\_ID  
    SELECT AVG(IT.Weightage) INTO mean\_score  
    FROM Interaction I  
    JOIN Interaction\_Type IT ON I.Interaction\_Type\_ID = IT.Interaction\_ID  
    WHERE I.User\_ID = p\_UserID AND I.Product\_ID = p\_ProductID;

    -- Get the Interaction\_ID associated with the latest interaction for the specified User\_ID and Product\_ID  
    SELECT Interaction\_ID INTO interaction\_id\_value  
    FROM Interaction  
    WHERE User\_ID = p\_UserID AND Product\_ID = p\_ProductID  
    ORDER BY TimeStamp DESC  
    LIMIT 1;

    -- Update the Recommendation Table with the calculated RecommendationScore  
    UPDATE recommendationtable  
    SET RecommendationScore = mean\_score, Interaction\_ID = interaction\_id\_value  
    WHERE User\_ID = p\_UserID AND Product\_ID = p\_ProductID;

    -- If there is no existing record, insert a new record  
    IF ROW\_COUNT() = 0 THEN  
        INSERT INTO recommendationtable (User\_ID, Product\_ID, RecommendationScore, Interaction\_ID)  
        VALUES (p\_UserID, p\_ProductID, mean\_score, interaction\_id\_value);  
    END IF;

END

1. **SQL QUERIES FOR ANALYSING DATASET:**

Analysing the dataset will help us in understanding trends and making key business decisions. We have created 10 queries, which are as follows:

1. **Details of top five customers with maximum expenditure:** This query will help identifying the top 5 customers, in terms of expenditure. The idea behind this query is to provide rewards for customers that purchase heavily in order to encourage them to purchase more.  
     
   SELECT c.customerid, c.name, c.email, c.age, c.gender, c.address, SUM(o.price) AS total\_purchases  
   FROM customer c  
   JOIN orders o ON c.customerid = o.user\_id  
   GROUP BY c.customerid, c.name, c.email, c.age, c.gender, c.address  
   ORDER BY total\_purchases DESC  
   LIMIT 5;

A screenshot of a computer

Description automatically generated

1. **Details of the top 5 customers with maximum interactions:** This query will help us identifying top 5 customers, in terms of maximum interactions. This query joins customer table with the interaction table and counts the maximum interactions for each customer.  
   SELECT c.customerid, c.name, c.email, c.age, c.gender, c.address,  
   COUNT(i.interaction\_id) AS total\_interactions  
   FROM customer c  
   JOIN interaction i ON c.customerid = i.user\_id  
   GROUP BY c.customerid, c.name, c.email, c.age, c.gender, c.address  
   ORDER BY total\_interactions DESC  
   LIMIT 5;

A screenshot of a computer

Description automatically generated

1. **Checking if any customer exists without any purchase:** This query will check for customer without any purchase. The main aim of this query is to check whether there is any customer without any purchase.  
   SELECT c.customerid, c.name, c.email, c.age, c.gender, c.address  
   FROM customer c  
   LEFT JOIN orders o ON c.customerid = o.user\_id  
   WHERE o.order\_id IS NULL;

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

1. **Finding the most popular product:** This query will check for the most popular products by considering the product that has been purchased more.SELECT p.product\_id, p.productname, c.categoryname, b.brandsname, p.price, p.inventory, SUM(o.quantity) AS total\_quantity\_sold FROM orders o  
   JOIN products p ON o.product\_id = p.product\_id  
   JOIN category c ON p.category\_id = c.category\_id  
   JOIN brands b ON p.brands\_id = b.brands\_id  
   GROUP BY p.product\_id, p.productname, c.categoryname, b.brandsname, p.price, p.inventory  
   ORDER BY total\_quantity\_sold DESC  
   LIMIT 5;

A screenshot of a computer

Description automatically generated

1. **Based on each category top 5 sales:** In this query, our goal is to identify the products sold the most across each category.  
   WITH RankedProducts AS (  
   SELECT c.category\_id, c.categoryname, p.product\_id, p.productname, o.quantity, o.price, ROW\_NUMBER() OVER (PARTITION BY c.category\_id ORDER BY o.quantity DESC) AS rank\_within\_category  
   FROM orders o  
   JOIN products p ON o.product\_id = p.product\_id  
   JOIN category c ON p.category\_id = c.category\_id  
   )  
   SELECT product\_id, productname, category\_id, categoryname, quantity, price  
   FROM RankedProducts  
   WHERE rank\_within\_category <= 5;

**A screenshot of a computer

Description automatically generated**

1. **Identifying the hour at which maximum interactions were done:** The goal of this query is to identify the hour at which users interacted with the e-commerce platform the most. It will help in understanding the user traffic.   
   SELECT EXTRACT(HOUR FROM i\_date) AS interaction\_hour,COUNT(\*) AS total\_interactions FROM Interaction  
   GROUP BY interaction\_hour  
   ORDER BY total\_interactions DESC   
   LIMIT 1;  
   A screenshot of a computer

   Description automatically generated
2. **Identifying products with inventory size less than 200:** This query will identify products that have less than 200 units in the inventory. This output will help in planning the inventory management for products.  
   SELECT p.product\_id, p.productname, c.categoryname, b.brandsname, p.price, p.inventory  
   FROM products p  
   JOIN category c ON p.category\_id = c.category\_id  
   JOIN brands b ON p.brands\_id = b.brands\_id  
   WHERE p.inventory < 200  
   ORDER BY Inventory DESC;  
   A screenshot of a computer

   Description automatically generated
3. **Identifying top 3 brands based on sales:** This query will help identify the top 3 brands based on their sales. It will help in determining the top performing brands.  
   SELECT b.brands\_id, b.brandsname, SUM(o.price) AS total\_sales  
   FROM orders o  
   JOIN products p ON o.product\_id = p.product\_id  
   JOIN brands b ON p.brands\_id = b.brands\_id  
   GROUP BY b.brands\_id, b.brandsname  
   ORDER BY total\_sales DESC  
   LIMIT 3;  
   A screenshot of a computer

   Description automatically generated
4. **Revenue per gender per category:** This shows the revenue that is coming from each gender, based on categories. This query will help in understanding user behaviour, based on their gender.  
   SELECT c.gender, cat.categoryname, SUM(o.price) AS Total\_Revenue  
   FROM orders o  
   JOIN products p ON o.product\_id = p.product\_id  
   JOIN category cat ON p.category\_id = cat.category\_id  
   JOIN brands b ON p.brands\_id = b.brands\_id  
   JOIN customer c ON o.user\_id = c.customerid  
   GROUP BY c.gender, cat.categoryname  
   ORDER BY c.gender, Total\_Revenue DESC;  
   A screenshot of a chat

   Description automatically generated

**5. Conclusion**

In this report, we have explained about how we have used SQL to create functions, stored procedures and triggers that will help in the functioning of a database in an E-Commerce Recommendation platform. We have also created some SQL queries, which can be used for data analysis which in turn can be used to make important business decisions.