**Sql code(E-Commerce Recommendation Engine)**

-- Analysis Started

-- 1. Select all records from the Products table:

SELECT \* FROM Products;

-- 2. Filter products by category 'Electronics':

SELECT \* FROM Products WHERE category = 'Electronics';

-- 3. Sort products by price in descending order:

SELECT \* FROM Products ORDER BY price DESC;

-- 4. Count the number of interactions:

SELECT COUNT(\*) AS interaction\_count FROM Interactions;

-- 5. Calculate the total purchase amount for each user:

SELECT user\_id, SUM(total\_amount) AS total\_purchase\_amount

FROM Past\_Purchases

GROUP BY user\_id;

-- 6. Retrieve the oldest purchase date:

SELECT MIN(purchase\_date) AS oldest\_purchase FROM Past\_Purchases;

-- 7. Join Products and Interactions to get product details with interaction type:

SELECT p.\*, i.interaction\_type

FROM Products p

JOIN Interactions i ON p.product\_id = i.product\_id;

-- 8. Subquery to find products with more than 10 interactions:

SELECT \* FROM Products

WHERE product\_id IN (SELECT product\_id FROM Interactions GROUP BY product\_id HAVING COUNT(\*) > 10);

-- Run the inner query separately to verify if it returns the expected counts of interactions per product:

SELECT product\_id, COUNT(\*) AS interaction\_count

FROM Interactions

GROUP BY product\_id;

-- 9. Update product price for a specific product:

UPDATE Products SET price = 1500 WHERE product\_id = 'P001';

-- 10. Delete an interaction record:

DELETE FROM Interactions WHERE interaction\_id = 5;

-- 11. Retrieve the top 5 users with the highest total purchase amount:

SELECT user\_id, SUM(total\_amount) AS total\_purchase\_amount

FROM Past\_Purchases

GROUP BY user\_id

ORDER BY total\_purchase\_amount DESC

LIMIT 5;

-- 12. Count the number of unique brands in the Products table:

SELECT COUNT(DISTINCT brand) AS unique\_brands FROM Products;

-- 13. Window function to rank products by price within each category:

SELECT product\_id, product\_name, category, price,

RANK() OVER (PARTITION BY category ORDER BY price) AS price\_rank

FROM Products;

-- 14. Common Table Expression (CTE) to find the average price of products:

WITH AvgPrice AS (

SELECT AVG(price) AS average\_price FROM Products

)

SELECT \* FROM Products WHERE price > (SELECT average\_price FROM AvgPrice);

-- 14. Create an index on the user\_id column of the Past\_Purchases table:

CREATE INDEX idx\_user\_id ON Past\_Purchases(user\_id);

-- 15. Retrieve the product with the highest total purchase amount:

SELECT product\_id, SUM(total\_amount) AS total\_purchase\_amount

FROM Past\_Purchases

GROUP BY product\_id

ORDER BY total\_purchase\_amount DESC

LIMIT 1;

-- 16. Create a view to show interactions with product details:

CREATE VIEW InteractionDetails AS

SELECT i.\*, p.product\_name, p.category, p.brand

FROM Interactions i

JOIN Products p ON i.product\_id = p.product\_id;

-- 17. Rollback a transaction if an error occurs while updating interactions:

START TRANSACTION;

UPDATE Interactions SET interaction\_type = 'Click' WHERE interaction\_id = 10;

SAVEPOINT before\_commit;

UPDATE Interactions SET interaction\_type = 'View' WHERE interaction\_id = 11;

ROLLBACK TO before\_commit;

COMMIT;

-- 18. Count the number of interactions per product

SELECT product\_id, COUNT(\*) AS interaction\_count

FROM Interactions

GROUP BY product\_id;

-- 19. List top N most popular products based on interactions

SELECT product\_id, COUNT(\*) AS interaction\_count

FROM Interactions

GROUP BY product\_id

ORDER BY interaction\_count DESC

LIMIT 3;

-- 20. Retrieve product details along with user interactions:

SELECT p.\*, i.interaction\_type, i.user\_id

FROM Products p

INNER JOIN Interactions i ON p.product\_id = i.product\_id;

-- 21. Find products with no interactions:

SELECT p.\*

FROM Products p

LEFT JOIN Interactions i ON p.product\_id = i.product\_id

WHERE i.product\_id IS NULL;

-- 22. Rank products by price within each category using window functions:

SELECT product\_id, product\_name, category, price,

RANK() OVER (PARTITION BY category ORDER BY price) AS price\_rank

FROM Products;

-- 24. Calculate the cumulative sum of total purchases by user:

SELECT user\_id, purchase\_id, total\_amount,

SUM(total\_amount) OVER (PARTITION BY user\_id ORDER BY purchase\_id) AS cumulative\_sum

FROM Past\_Purchases;

-- 25. Find products purchased more than once:

SELECT \*

FROM Products

WHERE product\_id IN (

SELECT product\_id

FROM Past\_Purchases

GROUP BY product\_id

HAVING COUNT(\*) > 1

);

-- 26. Retrieve interactions for products with prices above the average price:

SELECT \*

FROM Interactions

WHERE product\_id IN (

SELECT product\_id

FROM Products

WHERE price > (SELECT AVG(price) FROM Products)

);

-- 27. Create a CTE to calculate average product price by category:

WITH AvgPriceByCategory AS (

SELECT category, AVG(price) AS avg\_price

FROM Products

GROUP BY category

)

SELECT p.product\_id, p.product\_name, p.category, p.price, c.avg\_price

FROM Products p

JOIN AvgPriceByCategory c ON p.category = c.category;

-- 28. Use a CTE to find the top 3 users with the highest total purchase amounts:

WITH UserPurchaseTotals AS (

SELECT user\_id, SUM(total\_amount) AS total\_purchase\_amount

FROM Past\_Purchases

GROUP BY user\_id

)

SELECT user\_id, total\_purchase\_amount

FROM UserPurchaseTotals

ORDER BY total\_purchase\_amount DESC

LIMIT 3;

-- 29. Calculate the percentage contribution of each product to the total sales amount:

WITH ProductSales AS (

SELECT product\_id, SUM(total\_amount) AS total\_sales

FROM Past\_Purchases

GROUP BY product\_id

)

SELECT p.product\_id, p.product\_name, p.price, ps.total\_sales,

(ps.total\_sales / (SELECT SUM(total\_amount) FROM Past\_Purchases)) \* 100 AS sales\_percentage

FROM Products p

JOIN ProductSales ps ON p.product\_id = ps.product\_id;

-- 30. Identify users who made purchases of more than $500 in a single transaction:

SELECT user\_id, purchase\_id, total\_amount

FROM Past\_Purchases

WHERE total\_amount > 500;

-- 31. Calculate the average time between consecutive purchases for each user:

WITH UserPurchaseTimes AS (

SELECT user\_id,

purchase\_date - LAG(purchase\_date, 1) OVER (PARTITION BY user\_id ORDER BY purchase\_date) AS time\_diff

FROM Past\_Purchases

)

SELECT user\_id, AVG(time\_diff) AS avg\_time\_between\_purchases

FROM UserPurchaseTimes

GROUP BY user\_id;

-- Above query uses a window function to calculate the time difference between consecutive purchases for each user. Then, it calculates the average time between purchases for each user.

-- 32. Identify products that have been interacted with but not purchased:

SELECT i.product\_id, p.product\_name

FROM Interactions i

LEFT JOIN Past\_Purchases pp ON i.product\_id = pp.product\_id

JOIN Products p ON i.product\_id = p.product\_id

WHERE pp.product\_id IS NULL;

-- 33. Find users who made purchases in the first and last quarter of the year:

SELECT user\_id

FROM Past\_Purchases

WHERE EXTRACT(QUARTER FROM purchase\_date) = 1

OR EXTRACT(QUARTER FROM purchase\_date) = 4;

-- 34. Calculate the average quantity of products purchased by users who interacted with products priced above the average price:

WITH InteractedProducts AS (

SELECT DISTINCT user\_id, product\_id

FROM Interactions

),

AvgProductPrice AS (

SELECT AVG(price) AS avg\_price

FROM Products

)

SELECT ip.user\_id, AVG(pp.quantity) AS avg\_quantity

FROM InteractedProducts ip

JOIN Products p ON ip.product\_id = p.product\_id

JOIN Past\_Purchases pp ON ip.user\_id = pp.user\_id AND ip.product\_id = pp.product\_id

CROSS JOIN AvgProductPrice avgp

WHERE p.price > avgp.avg\_price

GROUP BY ip.user\_id;

-- 35. Find users who have interacted with products across multiple categories:

SELECT user\_id

FROM (

SELECT user\_id, COUNT(DISTINCT category) AS num\_categories

FROM Interactions i

JOIN Products p ON i.product\_id = p.product\_id

GROUP BY user\_id

) AS user\_categories

WHERE num\_categories > 1;