**SQL Questions**

1. Assume you're given a table containing information about Wayfair user transactions for different products. Write a query to calculate the next to the current month data for the total spend of each product, grouping the results by product ID

SELECT product\_id, SUM(amount) as total\_spend

FROM transactions

WHERE

YEAR(transaction\_date) = 2025

AND MONTH(transaction\_date) = 6

GROUP BY product\_id;

2. You have a table Transactions with columns TransactionID, AccountID, TransactionDate and Amount. Write a query to find out all transactions that occured in the last 30 days

SELECT \* FROM Transactions   
WHERE TransactionDate   
 BETWEEN CURRENT\_DATE AND CURRENT\_DATE - 30;

3. You have a Sales table with the following columns: Salesdate(date), ProductID (integer) and UnitsSold (integer). Each row in this table is unique. Write a SQL query to identify the products that have achieved 3 consequetive months of increasing unit sales based on the data:

WITH MonthySales AS (

SELECT

ProductID, YEAR(SalesDate) AS SalesYear, MONTH(SalesDate) AS SalesMonth,

UnitsSold, LAG(UnitsSold, 1, 0) OVER (PARTITION BY ProductID ORDER BY SalesDate)   
 AS PreviousMonthUnitsSold

FROM

Sales ),

ThreeMonthTrend AS (

SELECT

ProductID, SalesYear, SalesMonth, UnitsSold, PreviousMonthUnitsSold,

LEAD(UnitsSold, 1, 0) OVER (PARTITION BY ProductID ORDER BY SalesDate) AS NextMonthUnitsSold,

LEAD(UnitsSold, 2, 0) OVER (PARTITION BY ProductID ORDER BY SalesDate) AS TwoMonthsAheadUnitsSold

FROM

MonthySales

)

SELECT DISTINCT ProductID

FROM ThreeMonthTrend

WHERE

UnitsSold > PreviousMonthUnitsSold

AND NextMonthUnitsSold > UnitsSold

AND TwoMonthsAheadUnitsSold > NextMonthUnitsSold;

4. Identify any payments made at the same merchant with the same credit card for the same amount within 10 minutes of each other and reporting the count of such repeated payments.

WITH DuplicatePayments AS (

SELECT

t1.TransactionID AS TransactionID1,

t2.TransactionID AS TransactionID2

FROM

Transactions t1

INNER JOIN Transactions t2

ON t1.AccountID = t2.AccountID

AND t1.Amount = t2.Amount

AND t1.TransactionID < t2.TransactionID

AND ABS(DATEDIFF(minute, t1.TransactionDate, t2.TransactionDate)) <= 10

)

SELECT

COUNT(\*) AS RepeatedPaymentCount

FROM

DuplicatePayments;

5. Marketing Team needed to add simple statistics to their upcoming SuperbowlAd: the median number of searches made per year. You were given a summary table that tells you the number of searches made last year, write a query to report the median searches made per user:

WITH RankedSearches AS (

-- Assign row numbers and count total users

SELECT

SearchCount,

ROW\_NUMBER() OVER (ORDER BY SearchCount) AS RowAsc,

ROW\_NUMBER() OVER (ORDER BY SearchCount DESC) AS RowDesc,

COUNT(\*) OVER () AS TotalUsers

FROM

UserSearches

)

SELECT

AVG(1.0 \* SearchCount) AS MedianSearches

FROM

RankedSearches

WHERE

RowAsc IN (FLOOR((TotalUsers + 1)/2), CEIL((TotalUsers + 1)/2));

6. Write a query to print the cumulative balance of the merchant account at the end of each day , with the total balance reset back to zero at the end of the month. Output the transaction date and the cumulative balance:

WITH daily\_transactions AS (

-- Calculate the daily balance for each transaction

SELECT

DATE\_TRUNC('day', transaction\_date) AS transaction\_date,

SUM(CASE WHEN transaction\_type = 'deposit' THEN amount ELSE -amount END) AS daily\_balance

FROM

merchant\_transactions

GROUP BY

transaction\_date

),

monthly\_cumulative\_balances AS (

-- Calculate the cumulative balance within each month

SELECT

transaction\_date,

DATE\_TRUNC('month', transaction\_date) AS transaction\_month,

SUM(daily\_balance) OVER (PARTITION BY DATE\_TRUNC('month', transaction\_date) ORDER BY transaction\_date) AS cumulative\_balance\_within\_month

FROM

daily\_transactions

ORDER BY

transaction\_date

)

-- Output the desired results

SELECT

transaction\_date,

cumulative\_balance\_within\_month AS cumulative\_balance

FROM

monthly\_cumulative\_balances

ORDER BY

transaction\_date;

7. Explain the concept of correlated subqueries in SQL. Can you provide an example of how you can use them?

A correlated subquery is a subquery that depends on the outer query. It means that the WHERE clause of the correlated subquery uses the data of the outer query.The main difference between a correlated subquery and a non-correlated subquery is that you cannot execute a correlated subquery alone like a non-correlated subquery. In addition, a correlated subquery executes once for each selected row from the outer query.

A correlated subquery is also known as a repeating subquery or synchronized subquery.

Below are some examples of usage of correlated subquery:

***a. SQL correlated subquery in the SELECT clause example***

The following query selects the top five customers by sales:

SELECT companyname, city,

(SELECT

SUM(unitprice \* quantity)

FROM

orders

INNER JOIN

orderdetails ON orderdetails.orderid = orders.orderid

WHERE

orders.customerid = customers.customerid) AS total

FROM

customers

ORDER BY total DESC

LIMIT 5;

The correlated subquery calculates total sales for each selected customer from the customers table. The selected customerid from the outer query is passed to the correlated subquery for getting the corresponding sales data.

***b. SQL correlated subquery in WHERE clause example***

You can also use the correlated subquery in a WHERE clause. For example, the following example uses a correlated subquery in the WHERE clause to find customers that have total sales more than 100K:

SELECT companyname, city

FROM customers

WHERE

100000 < (

SELECT

SUM(unitprice \* quantity)

FROM

orders

INNER JOIN

orderdetails ON orderdetails.orderid = orders.orderid

WHERE

orders.customerid = customers.customerid);

For each customer, the correlated subquery calculates the total sales. The WHERE clause checks if the total sales, which is returned by the correlated subquery, is greater than 100K.

***c. SQL correlated subquery in HAVING clause example***

You can use a correlated subquery in the HAVING clause of an outer query. See the following example:

SELECT

t1.categoryID, categoryName

FROM

products t1

INNER JOIN

categories c ON c.categoryID = t1.categoryID

GROUP BY categoryID

HAVING MAX(unitprice) > ALL (

SELECT 2 \* AVG(unitprice)

FROM

products t2

WHERE

t1.categoryID = t2.categoryID)

SQL correlated subquery with HAVING example

In the above query: The subquery calculates the average unit price in each category and multiplies it with 2.

The outer query selects the category of the product whose unit price is greater than the double average unit price returned by the correlated subquery.