SQL of the day: Finding Purchases

PRESENTED BY NIVAN RAMADHAN SUGIANTORO

The Problem

DEFINING REPEAT PURCHASES

We need to identify users who made a purchase with a **gap of 1 to 7 days** after their last purchase, excluding same-day transactions.

The Approach: LAG() Function

LAG()

The LAG() function pulls the previous purchase date, enabling straightforward comparisons between the current and prior purchases for each user.

PARTITIONING

Partitioning by user_id allows independent tracking of each user's purchase history, ensuring that the comparison is accurate and relevant for each individual.

ORDERING

Ordering by created_at ensures that the purchase dates are chronologically arranged, allowing for correct calculation of the gap between purchases.

The SQL Solution Explained

```sql
WITH transactions AS (
    SELECT
        user_id,
        created_at,
        LAG(created_at) OVER (
            PARTITION BY user_id
            ORDER BY created_at
        ) AS prev_date
    FROM amazon_transactions
)
SELECT DISTINCT user_id
FROM transactions
WHERE prev_date IS NOT NULL
  AND created_at - prev_date BETWEEN 1 AND 7;
```

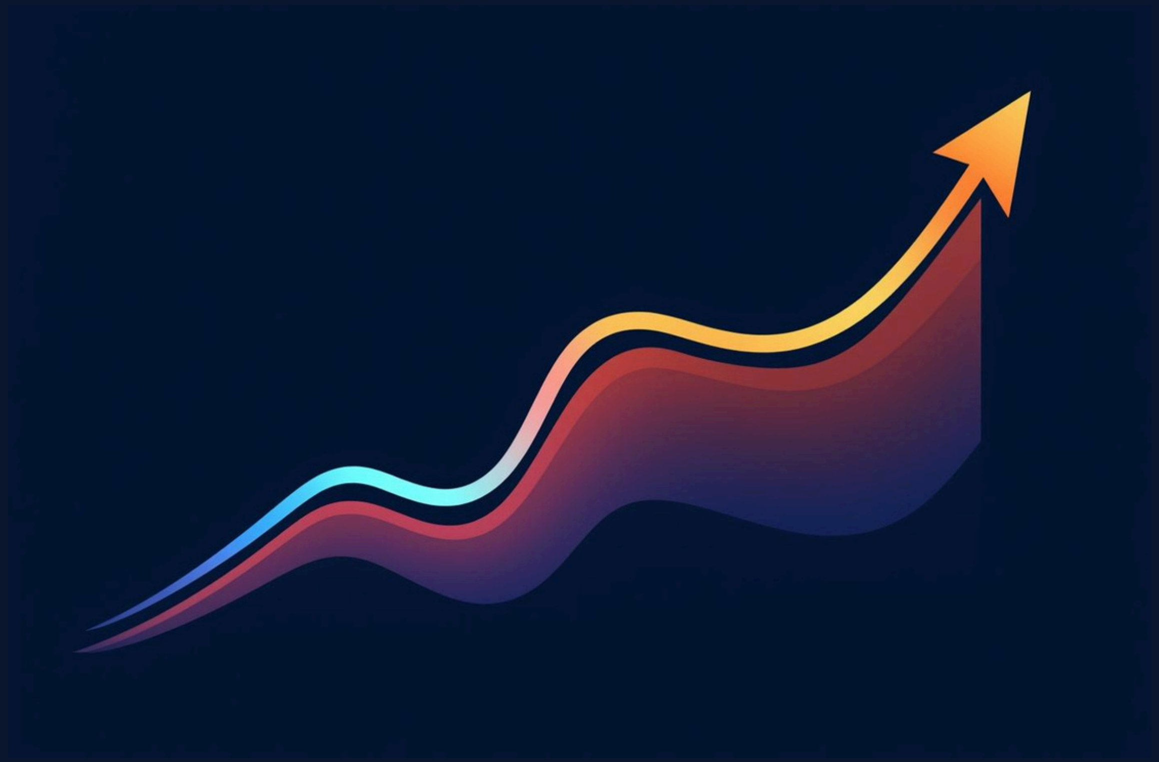
HIGHLIGHTING DISTINCT USERS

The query identifies users who made repeat purchases within the specified gap, ensuring accurate results without duplicates.

UNDERSTANDING THE CODE

This SQL snippet demonstrates the use of LAG() and DISTINCT to analyze user purchasing behavior effectively and efficiently.

Understanding Repeat Purchaser Patterns for Business Insights



LOYALTY

Identifying customers who return frequently helps strategies.

FREQUENCY

Understanding how often customers return is crucial.

ENGAGEMENT

A drop in metrics signals a need for campaigns.

Key Takeaways

LAG()

The **LAG()** function simplifies time comparisons between rows, making it ideal for analyzing sequential data trends in SQL queries.

DISTINCT

Using **DISTINCT** is essential to avoid counting the same user multiple times, ensuring accurate analysis of unique repeat purchases.

WINDOW FUNCTIONS

Window functions combined with date arithmetic are powerful tools in SQL, enhancing time-series analysis and enabling deeper insights into customer behavior.