@codeyourpath

7 SQL TRICKS IN DATA ANALYSIS

Swipe Left

codeyourpath

1. USING SUBQUERIES

Subqueries are powerful tools in SQL that allow you to perform complex operations by nesting one query within another. For example, you can use a subquery to find all the customers who made their first purchase in a specific month, and then use that result set to analyze their subsequent purchases.

```sql
SELECT customer_id, SUM(amount) as total_purchase_amt
FROM orders
WHERE customer_id IN (SELECT customer_id
                      FROM orders
                      WHERE DATE_TRUNC('month', order_date) = '2022-01-01')
GROUP BY customer_id;
```

Swipe Left

codeyourpath

2. AGGREGATION FUNCTIONS

SQL has powerful aggregation functions such as COUNT, SUM, AVG, and MAX that allow you to quickly calculate summary statistics for large datasets. You can use these functions to calculate metrics such as revenue, average order value, or customer lifetime value.

```sql
SELECT COUNT(*) as total_orders,
SUM(amount) as total_revenue,
AVG(amount) as avg_order_value
FROM orders;
```

Swipe Left

codeyourpath

3. JOINS

Joins are used to combine data from multiple tables into a single result set. For example, you can use a join to combine customer data with order data, allowing you to analyze customer behavior based on their purchase history.

```sql
SELECT customers.customer_id, customers.first_name, customers.last_name,
SUM(orders.amount) as total_purchase_amt
FROM customers
JOIN orders
ON customers.customer_id = orders.customer_id
GROUP BY customers.customer_id;
```

Swipe Left

codeyourpath

4. WINDOW FUNCTIONS

Window functions allow you to perform calculations over a specific range of rows in a result set. For example, you can use a window function to calculate the running total of sales over a specific period, such as the last 30 days.

```sql
SELECT order_date, SUM(amount)
OVER (ORDER BY order_date ROWS BETWEEN 29 PRECEDING AND CURRENT ROW) as last_30d_sales
FROM orders;
```

Swipe Left

codeyourpath

5. CONDITIONAL STATEMENTS

SQL has powerful conditional statements such as CASE and IF-THEN-ELSE that allow you to perform complex logic in your queries. For example, you can use a CASE statement to classify customers into different segments based on their purchase behavior.

```sql
SELECT customer_id,
CASE WHEN SUM(amount) < 1000 THEN 'Low Value Customer'
        WHEN SUM(amount) >= 1000 AND SUM(amount) < 5000 THEN 'Medium Value Customer'
        ELSE 'High Value Customer' END as customer_segment
FROM orders
GROUP BY customer_id;
```

Swipe Left

codeyourpath

6. INDEXING

Indexing is a technique used to optimize the performance of queries by creating a data structure that allows for faster data retrieval. By indexing frequently queried columns, you can significantly improve the performance of your SQL queries.

```sql
CREATE INDEX idx_orders_customer_id ON orders (customer_id);
```

Swipe Left

codeyourpath

7.CTEs

CTEs (Common Table Expressions) allow you to define a temporary result set that can be referenced in subsequent queries. For example, you can use a CTE to create a summary table of customer purchases, and then use that table in subsequent queries to analyze customer behavior.

```sql
WITH customer_purchases AS (
  SELECT customer_id, SUM(amount) as total_purchase_amt
  FROM orders
  GROUP BY customer_id)
SELECT customer_id, total_purchase_amt,
       CASE WHEN total_purchase_amt < 1000 THEN 'Low Value Customer'
            WHEN total_purchase_amt >= 1000 AND total_purchase_amt < 5000 THEN 'Medium Value Customer'
            ELSE 'High Value Customer' END as customer_segment
FROM customer_purchases;
```

Swipe Left

@codeyourpath

If you found it helpful, leave a comment and share your thoughts.

Follow me for more updates. Thanks!

Follow