==========

REAL-WORLD SQL CHALLENGES PROJECT

Author: Md Irshad Alam

==========

This query helps identify churned customers (those who ordered last year but not this year). Useful for retention strategy

==========

REAL-WORLD SQL CHALLENGES

==========

1. Find the second highest transaction amount (Subquery) ??

```
SELECT MAX(amount) AS second_highest_payment
FROM payments
WHERE amount < (SELECT MAX(amount) FROM payments);
```

1b. Find 2nd highest using OFFSET LIMIT

```
SELECT DISTINCT amount AS second_highest_payment
FROM payments
ORDER BY amount DESC
LIMIT 1 OFFSET 1;
```

	second_highest_payment numeric
1	60000.00

1c. Find Nth highest without LIMIT/TOP

This query helps find the 2nd or Nth largest transaction in the payments table. It is useful when ranking payments, salaries, or sales figures without directly using built-in ranking keywords.!

```
WITH RankedValues AS (
   SELECT amount,
      ROW_NUMBER() OVER (ORDER BY amount DESC) AS rnk
   FROM payments
)
SELECT amount
FROM RankedValues
 WHERE rnk = 5;
```

	amount numeric (10,2)	rnk bigint
1	3000.00	5

2. Find customers who purchased the same product more than once ??
   This identifies customers who repeatedly bought the same product. It is useful for analyzing loyal buyers and repeat purchase patterns

```
SELECT
  c.customer_id,     oi.product_id,
    COUNT(*) AS total_orders
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY c.customer_id, oi.product_id
HAVING COUNT(*) > 1;
```

	customer_id integer	product_id integer	total_orders bigint
1	1	101	2

3. Customers who registered but never placed orders ??
   This query finds inactive customers who signed up but never purchased anything. Businesses use this insight to target them with promotions or onboarding campaigns

```
SELECT c.customer_id, c.customer_name
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

__No output: every customer has placed at least one order.

4. Daily Active Users (DAU)

This query calculates the number of unique active users per day. It is a key engagement metric used in apps and e-commerce to measure how often customers return

```
SELECT     login_date,
    COUNT(DISTINCT customer_id) AS daily_active_users
FROM logins
GROUP BY login_date
ORDER BY login_date;
```

	login_date date	daily_active_users bigint
1	2024-04-10	1
2	2024-04-11	2
3	2024-04-12	2

5. Month-over-Month Revenue Growth % ??
   This measures how revenue is growing or declining compared to the previous month. It is crucial for tracking business performance and growth trends

```
WITH MonthlyRevenue AS (
    SELECT DATE_TRUNC('month', order_date) AS month,
        SUM(total_amount) AS revenue
    FROM orders
    GROUP BY DATE_TRUNC('month', order_date)
),
RevenueWithLag AS (
    SELECT month, revenue,
        LAG(revenue) OVER (ORDER BY month) AS prev_month_revenue
    FROM MonthlyRevenue
)
SELECT month, revenue, prev_month_revenue,
      ROUND(
        CASE WHEN prev_month_revenue IS NULL OR prev_month_revenue = 0
            THEN NULL
            ELSE ((revenue - prev_month_revenue) / prev_month_revenue) * 100
        END, 2
      ) AS mom_growth_percent
FROM RevenueWithLag;
```

	month timestamp with time zone	revenue numeric	prev_month_revenue numeric	mom_growth_percent numeric
1	2024-01-01 00:00:00+05:30	85000.00	[null]	[null]
2	2024-02-01 00:00:00+05:30	25000.00	85000.00	-70.59
3	2024-03-01 00:00:00+05:30	60000.00	25000.00	140.00
4	2024-04-01 00:00:00+05:30	8000.00	60000.00	-86.67
5	2024-05-01 00:00:00+05:30	5000.00	8000.00	-37.50

6. Latest order per customer (ROW_NUMBER) ??
This finds the most recent order for every customer. It helps businesses track customer recency, which is important for retention and re-engagement strategies.

```
WITH ranked AS (
    SELECT customer_id, order_id, order_date, total_amount,
        ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date DESC) AS rnk
FROM orders
)
SELECT * FROM ranked WHERE rnk = 1;
```

	customer_id integer	order_id [PK] integer	order_date date	total_amount numeric (10,2)	rnk bigint
1	1	1003	2024-03-05	60000.00	1
2	2	1002	2024-02-10	25000.00	1
3	3	1004	2024-04-12	5000.00	1
4	4	1005	2024-04-15	3000.00	1
5	5	1006	2024-05-30	5000.00	1

7. Products whose sales dropped compared to previous month ??
This query identifies products whose sales decreased month-over-month. It helps detect declining products early so businesses can investigate causes.

```
WITH MonthlySales AS (
    SELECT DATE_TRUNC('month', o.order_date) AS month,
        oi.product_id,
        SUM(oi.quantity) AS total_sales
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY DATE_TRUNC('month', o.order_date), oi.product_id
),
Sales AS (
    SELECT product_id, month, total_sales,
        LAG(total_sales) OVER (PARTITION BY product_id ORDER BY month) AS prev_month_sales
    FROM MonthlySales
)
SELECT * FROM Sales
WHERE total_sales < prev_month_sales;

--There is no product whose sales have dropped compared to the previous month.
```

8. Classify customers into spending categories ??

This groups customers into High, Medium, and Low spenders. It is useful for segmentation, targeted marketing, and personalized offers

```
WITH CustomerSpend AS (
    SELECT customer_id, SUM(total_amount) AS total_spend
    FROM orders
    GROUP BY customer_id
)
SELECT customer_id, total_spend,
      CASE WHEN total_spend >= 10000 THEN 'High Spender'
           WHEN total_spend >= 5000 THEN 'Medium Spender'
           ELSE 'Low Spender'
      END AS spend_category FROM
CustomerSpend ORDER BY
total_spend DESC;
```

	customer_id integer	total_spend numeric	spend_category text
1	1	145000.00	High Spender
2	2	25000.00	High Spender
3	3	5000.00	Medium Spender
4	5	5000.00	Medium Spender
5	4	3000.00	Low Spender

•

8. Most popular product category in each region ??
   This finds the best-selling product category in every region. Businesses use this insight for regional marketing strategies and inventory planning.

```
WITH CategorySales AS {
    SELECT c.region, p.category, SUM(oi.quantity) AS total_units_sold
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    JOIN products p ON oi.product_id = p.product_id
    JOIN customers c ON o.customer_id = c.customer_id
    GROUP BY c.region, p.category
),
RankedCategories AS (
```

```
    SELECT region, category, total_units_sold,
        ROW_NUMBER() OVER (PARTITION BY region ORDER BY total_units_sold DESC) AS rn
FROM CategorySales
)
SELECT region, category AS most_popular_category, total_units_sold
FROM RankedCategories
WHERE rn = 1;
```

	region character varying (30)	most_popular_category character varying (30)	total_units_sold bigint
1	East	Fashion	1
2	North	Electronics	2
3	South	Electronics	1
4	West	Electronics	1

9. Customers who ordered last year but not this year ??
   This detects churned customers who stopped ordering. It is critical for retention campaigns and win-back strategies

```
WITH LastYear AS (
    SELECT DISTINCT customer_id
    FROM orders
    WHERE EXTRACT(YEAR FROM order_date) = 2024
),
ThisYear AS (
    SELECT DISTINCT customer_id
    FROM orders
    WHERE EXTRACT(YEAR FROM order_date) = 2025
)
SELECT customer_id FROM LastYear
WHERE customer_id NOT IN (SELECT customer_id FROM ThisYear);
```

	customer_id integer
1	1
2	2
3	3
4	4
5	5

10. Cumulative sales month by month ??
This query calculates sales progressively over time. It helps businesses visualize revenue growth and overall sales performance.

```
WITH MonthlySales AS (
    SELECT DATE_TRUNC('month', order_date) AS month,
        SUM(total_amount) AS monthly_sales
    FROM orders
    GROUP BY DATE_TRUNC('month', order_date)
)
SELECT month, monthly_sales,
    SUM(monthly_sales) OVER (ORDER BY month) AS cumulative_sales
FROM MonthlySales
ORDER BY month;
```

	month timestamp with time zone	monthly_sales numeric	cumulative_sales numeric
1	2024-01-01 00:00:00+05:30	85000.00	85000.00
2	2024-02-01 00:00:00+05:30	25000.00	110000.00
3	2024-03-01 00:00:00+05:30	60000.00	170000.00
4	2024-04-01 00:00:00+05:30	8000.00	178000.00
5	2024-05-01 00:00:00+05:30	5000.00	183000.00

11. Detect duplicate transactions ??
This identifies duplicate payments (same customer, product, and date) and removes them. It ensures data accuracy and prevents financial reporting errors

```
WITH RankedTransactions AS (
    SELECT p.payment_id, o.customer_id, oi.product_id, DATE(o.order_date) AS order_date,
```

```
        ROW_NUMBER() OVER (PARTITION BY o.customer_id, oi.product_id, DATE(o.order_date)
ORDER BY p.payment_id) AS rn
    FROM payments p
    JOIN orders o ON p.order_id = o.order_id
    JOIN order_items oi ON o.order_id = oi.order_id
)
SELECT * FROM RankedTransactions WHERE rn > 1;
```

```
--There are no duplicate transactions in this dataset.
```

```
--Remove Or Delete Duplicate !
```

```
WITH RankedTransactions AS (
    SELECT
        p.payment_id,
        ROW_NUMBER() OVER (
            PARTITION BY o.customer_id, oi.product_id, DATE(o.order_date)
            ORDER BY p.payment_id
        ) AS rn
    FROM payments p
    JOIN orders o ON p.order_id = o.order_id
    JOIN order_items oi ON o.order_id = oi.order_id
)
DELETE FROM payments
WHERE payment_id IN (
    SELECT payment_id
    FROM RankedTransactions
    WHERE rn > 1
); --0 Duplicate!
```

==========

END OF PROJECT