NORTHWIND TRADERS ANALYSIS

VINCENT MUKOMBA

TABLE OF CONTENTS

INTRODUCTION

Northwind Traders is a global gourmet food distributor specializing in high-quality products sourced from premium suppliers worldwide. The company manages an extensive catalog of products across multiple categories and serves customers ranging from small businesses to large enterprises.

As a **Data Analyst**, the role is to provide actionable insights that drive strategic decision-making. The company's leadership team relies on data-driven reports to optimize inventory, improve customer relationships, increase sales, and enhance operational efficiency.

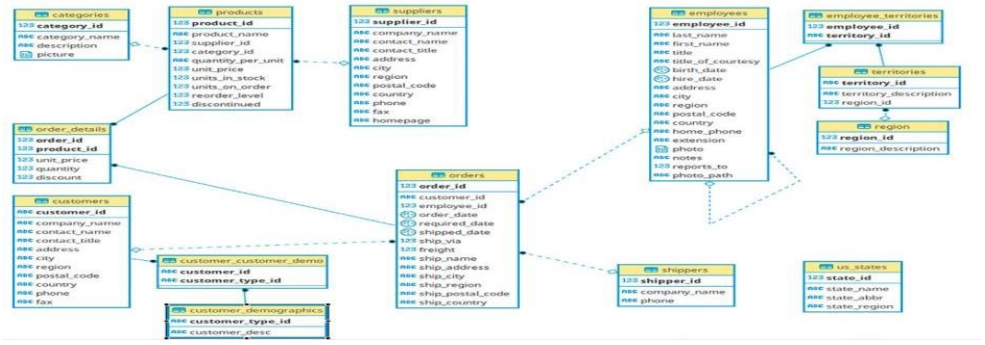
OBJECTIVE OF THE ANALYSIS

The management has asked to explore and answer critical business questions using advanced SQL techniques. This analysis will focus on:

- Evaluating employee sales performance to recognize top performers.
- Understanding product sales trends to optimize inventory and marketing strategies.
- Identifying high-value customers for targeted promotions.
- Monitoring sales growth and forecasting trends to support company expansion.
- Analyzing supplier contributions to determine procurement efficiency.

DATASET

Here is the dataset required for the analysis and the schema is shared below:



PROBLEM STATEMENT 1: SECOND-BEST SELLING PRODUCT BY CATEGORY

Business Scenario: Category managers at Northwind want to promote products that are strong sellers but not the top in their category. They decide to look at the **second-highest grossing product in each category** by total sales revenue. This helps identify products that have high sales potential right behind the category leaders.

Question: *Which product is the second-best selling (by total revenue) in each product category, and how much revenue did it generate?*

Answer: Provide the category name, product name, and total sales for that product. Use a CTE to organize the calculation.

STRATEGIC APPROACH:

1. **Schema:**
    - To solve this, we need to join the following tables:
    - order_details: has product_id, unit_price, quantity, discount (used to calculate revenue)
    - products: has product_id, product_name, category_id
    - categories: has category_id, category_name
2. **CTE: Revenue by Product**
    - Calculate total revenue per product (accounting for discounts):
    - Join order_details and products
    - Group by product_id
    - Revenue formula: SUM(unit_price * quantity * (1 - discount))

3. **CTE: Add Category Info**
    - Join the revenue result with products and categories to get:
    - category_id
    - category_name
    - product_name
    - Total_revenue

4. **CTE: Rank Products Within Categories**
    - Use ROW_NUMBER() or DENSE_RANK() window function partitioned by category_id, ordered by revenue descending
    - This helps to assign ranks (1 = highest revenue, 2 = second-highest, etc.)

5. **Final Step: Filter for Rank = 2**
    - Only select products where rank = 2 (i.e., second-highest revenue in that category)

QUERY:

```
WITH product_revenue AS (
    SELECT
        c."categoryName",
        p."productName",
        c."categoryID",
        p."productID",
        SUM(od."unitPrice" * od."quantity" * (1 - od."discount")) AS total_revenue
--Total Revenue for Each Product
FROM
        northwind_traders."order_details" AS od
        JOIN northwind_traders."products" AS p ON od."productID" = p."productID"
        JOIN northwind_traders."categories" AS c ON p."categoryID" = c."categoryID"
    GROUP BY
        c."categoryName", c."categoryID", p."productID",
p."productName" ),
ranked_products AS (
    SELECT
        "categoryName",
"productName",
total_revenue,
RANK() OVER (PARTITION BY "categoryID" ORDER BY total_revenue DESC) AS
revenue_rank    --Rank Products Within Each Category by Revenue
FROM
        product_revenue
)
--Select Only the Second-Best
SELECT
    "categoryName",
"productName",
total_revenue,
       revenue_rank
FROM
ranked_products
WHERE
    revenue_rank = 2  --Filter to revenue_rank = 2
ORDER BY
     "categoryName";   -- second-highest grossing products per category
```

OUTPUT:

	categoryName character varying	productName character varying	total_revenue numeric	revenue_rank bigint
1	Beverages	Ipoh Coffee	23526.700	2
2	Condiments	Sirop d'érable	14352.600	2
3	Confections	Sir Rodney's Marmalade	22563.360	2
4	Dairy Products	Camembert Pierrot	46825.480	2
5	Grains/Cereals	Wimmers gute Semmelknödel	21957.9675	2
6	Meat/Poultry	Alice Mutton	32698.380	2
7	Produce	Rössle Sauerkraut	25696.640	2
8	Seafood	Ikura	20867.340	2

Total rows: 8 Query complete 00:00:00.085

RESULT ANALYSIS:

1. Camembert Pierrot (Dairy) and Alice Mutton (Meat/Poultry) stand out with high second-place revenues, indicating these products are strong contenders for top positions.
2. Even the second-place products generate substantial revenue, suggesting the sales volume is not heavily skewed toward a single product in each category — this is a good sign of product diversity and demand.
3. Revenue gaps between first and second place (not shown here, but implied) can inform promotional prioritization.

RECOMMENDATION:

1. **Promote Products**: Target second-best sellers with marketing campaigns to boost them toward the top.
2. **Ensure Stock Availability**: Maintain strong inventory for these high-performing items to meet demand.
3. **Use in Bundles**: Pair them with best-sellers for cross-sell opportunities and increased sales.

PROBLEM STATEMENT 2: TOP 3 CUSTOMERS BY TOTAL SALES

Business Scenario: The sales team is planning a loyalty program and wants to reward the top 3 customers by purchase volume. If there is a tie for third place, all tied customers should be included. Knowing the biggest spenders will help tailor special offers to them.

Question: *Who are the top three customers in terms of total sales revenue?*
Show customer's name, total spending, and their sales rank. Include any customers tied for third place by using a window function to rank the totals.

STRATEGIC APPROACH:

---

1. **Schema:**

   To solve this, we need to join the following tables:

   - orders: contains order_id, customer_id
   - order_details: contains order_id, unit_price, quantity, discount
   - customers: contains customer_id, company_name (customer name)

2. **CTE: Total Revenue per Order**

   - Join order_details and orders
   - total_revenue = unit_price * quantity * (1 - discount)
   - Group by order_id

3. **CTE: Total Revenue per Customer**

   - Join the revenue per order back with the orders table (to get customer_id)
   - Sum the revenue **per customer**
   - Join with customers to get company_name

4. **CTE: Rank Customers by Total Spending**

- Use a window function like RANK() or DENSE_RANK() ordered by total spending **descending**
- RANK() if you want to leave gaps in case of ties
- DENSE_RANK() if you want contiguous ranks even if there are ties

5. **Final Step: Filter for Rank = 2**

- Filter for customers where rank <= 3
- This ensures:
- You get the top 3 spenders
- And you **include any ties** at rank 3

QUERY:

```
WITH customer_revenue AS (
    SELECT
        c."companyName" AS customer_name, -- taking company name as a customer name
bcz customer name is not given in the schema
        o."customerID",
SUM(od."unitPrice" * od."quantity" * (1 - od."discount")) AS total_spent --The
portion of the price that is paid (1 - od."discount")     FROM
        northwind_traders."order_details" AS od
        JOIN northwind_traders."orders" AS o ON od."orderID" = o."orderID"
        JOIN northwind_traders."customers" AS c ON o."customerID" = c."customerID"
    GROUP BY
        o."customerID", c."companyName"
),
--Rank customers using
RANK() ranked_customers AS (
SELECT
        customer_name,
total_spent,
        DENSE_RANK() OV ER (ORDER BY total_spent DESC) AS sales_rank
    FROM




        customer_revenue
)
--top 3 ranks including ties at 3rd SELECT
    customer_name,
total_spent,
sales_rank FROM
    ranked_customers WHERE
    sales_rank <= 3
ORDER BY
sales_rank;
```

OUTPUT RESULT SNAPSHOT:

	customer_name character varying	total_spent numeric	sales_rank bigint
1	QUICK-Stop	110277.3050	1
2	Ernst Handel	104874.9785	2
3	Save-a-lot Markets	104361.9500	3

Total rows: 3 | Query complete 00:00:00.084

RESULT ANALYSIS:

- All three customers have spent over **$100K**, making them high-value clients.
- The spending gap between rank 2 and 3 is small (~$513), indicating a closely competitive top tier.

RECOMMENDATION:

1. **Launch Loyalty Rewards**: Offer exclusive perks (discounts, early access) to retain these top customers.
2. **Personalized Engagement**: Assign account reps or create custom offers to strengthen relationships.
3. **Upsell Opportunities**: Target them with premium products or bulk-buy incentives to boost future sales.

PROBLEM STATEMENT 3: TOP SUPPLIERS BY PRODUCT VARIETY

Business Scenario: The procurement department wants to evaluate supplier partnerships. They are interested in which suppliers offer the widest variety of products. Using a ranking that does not skip numbers (so ties share the same rank), they can list the top suppliers by product count to see who has a broad catalog.

Question: *Which suppliers provide the most products to Northwind, and how do they rank in terms of product count?*

List each top supplier's name, the number of different products they supply, and their rank. Use DENSE_RANK so that suppliers with equal product counts share the same rank.

STRATEGIC APPROACH:

---

1. **Schema:**
    - products: contains product_id, supplier_id, etc.
    - suppliers: contains supplier_id, company_name (or similar)
2. **CTE: Count of Products per Supplier**
    - Group products by supplier_id
    - Count the number of products (e.g., COUNT(*))
    - Join with suppliers table to get the supplier's name
3. **CTE: Rank Suppliers by Product Count**
    - Use DENSE_RANK() window function
    - Partitioning is not needed (rank globally)
    - Order by product count **descending**

QUERY:

```
WITH supplier_product_count AS (
   SELECT
      s."companyName" AS supplier_name,
COUNT(p."productID") AS product_count
   FROM
      northwind_traders."products" AS p
      JOIN northwind_traders."suppliers" AS s ON p."supplierID" = s."supplierID"
   GROUP BY
      s."companyName".      --Count of Products per Supplier
),
--Rank Suppliers by Product Count
ranked_suppliers AS (
SELECT
supplier_name,
product_count,
      DENSE_RANK() OVER (ORDER BY product_count DESC) AS supplier_rank
   FROM
      supplier_product_count
)
SELECT
   supplier_name,
product_count,
   supplier_rank
FROM
   ranked_suppliers
ORDER BY
    supplier_rank;
```

OUTPUT :

14	Norske Meierier	3	3
15	Formaggi Fortini s.r.l.	3	3
16	Heli Süßwaren GmbH & Co. KG	3	3
17	Pasta Buttini s.r.l.	2	4
18	Cooperativa de Quesos 'Las Cabras'	2	4
19	PB Knäckebröd AB	2	4
20	Gai pâturage	2	4
21	Aux joyeux ecclésiastiques	2	4
22	New England Seafood Cannery	2	4
23	Ma Maison	2	4
24	Forêts d'érables	2	4
25	Zaanse Snoepfabriek	2	4
26	Lyngbysild	2	4
27	Escargots Nouveaux	1	5
28	Nord-Ost-Fisch Handelsgesellschaft mbH	1	5
29	Refrescos Americanas LTDA	1	5

Total rows: 29 Query complete 00:00:00.062

	supplier_name character varying	product_count bigint	supplier_rank bigint
1	Pavlova	5	1
2	Plutzer Lebensmittelgroßmärkte AG	5	1
3	Specialty Biscuits	4	2
4	New Orleans Cajun Delights	4	2
5	Karkki Oy	3	3
6	Leka Trading	3	3
7	G'day	3	3
8	Mayumi's	3	3
9	Bigfoot Breweries	3	3
10	Exotic Liquids	3	3
11	Tokyo Traders	3	3
12	Grandma Kelly's Homestead	3	3
13	Svensk Sjöföda AB	3	3

RESULT ANALYSIS:

- Two suppliers tie for the top spot with **5 distinct products** each.

- A total of **29 suppliers** are involved, showcasing a diversified supplier base.
- Many suppliers provide **3 or more products**, reflecting stable procurement sources.

RECOMMENDATION:

1. **Strengthen Partnerships** with top-ranked suppliers to ensure consistent supply and negotiate better terms.
2. **Consolidate Orders** with suppliers offering a broader catalog to reduce complexity and shipping costs
3. **Evaluate Low Variety Suppliers** for potential consolidation or secondary supplier status.

PROBLEM STATEMENT 4: MOST RECENT ORDER PER CUSTOMER

Business Scenario: The customer relations team wants to improve engagement by contacting customers who haven't ordered recently. They need to know the date of each customer's most recent order. This

information will be used to schedule follow-up calls or emails, focusing on those whose last orders were a while ago.

Question: *For each customer, what is the date of their latest order?* Provide the customer name and the date of their most recent order.
Use a window function to pick the most recent order for each customer.

STRATEGIC APPROACH:

---

1. **Schema**

   To solve this, we need to join the following tables:

   - orders: contains order_id, customer_id, order_date
   - customers: contains customer_id, company_name

2. **Join Customers with Orders**

   - Join customers to orders using customer_id
   - This gives access to both company_name and their order_dates

3. **Use a Window Function to Rank Orders per Customer**

   - Use ROW_NUMBER() or RANK() **partitioned by customer_id**, ordered by order_date DESC
   - This assigns a 1st rank to the most recent order for each customer

4. **Filter for the Most Recent Order Only**

   - Wrap the above in a CTE or subquery
   - Filter for row_number = 1 to get only the **latest order** for each customer

QUERY:

```sql
--Join Customers with Orders
WITH ranked_orders AS (
    SELECT
        c."companyName" AS customer_name,
        o."orderDate",
--Window Function to Rank Orders per Customer
        RANK() OVER ( PARTITION BY o."customerID" ORDER BY o."orderDate"
DESC ) AS order_rank     FROM
        northwind_traders."orders" AS o
        JOIN northwind_traders."customers" AS c
        ON o."customerID" = c."customerID"
)
SELECT
    customer_name,
    "orderDate" FROM
    ranked_orders WHERE
    order_rank = 1 --to get only the latest order for each customer ORDER
BY
    "orderDate" ASC;  -- oldest recent orders first
```

OUTPUT:

	customer_name character varying	orderDate timestamp without time zone
1	Centro comercial Moctezuma	1996-07-18 00:00:00
2	Lazy K Kountry Store	1997-05-22 00:00:00
3	Hungry Coyote Import Store	1997-09-08 00:00:00
4	Mère Paillarde	1997-10-30 00:00:00
5	Familia Arquibaldo	1997-10-31 00:00:00
6	Vins et alcools Chevalier	1997-11-12 00:00:00
7	GROSELLA-Restaurante	1997-12-18 00:00:00
8	Folies gourmandes	1997-12-22 00:00:00
9	Laughing Bacchus Wine Cellars	1998-01-01 00:00:00
10	Trail's Head Gourmet Provisioners	1998-01-08 00:00:00
11	Blondesddsl père et fils	1998-01-12 00:00:00
12	Tradição Hipermercados	1998-01-19 00:00:00
13	Consolidated Holdings	1998-01-23 00:00:00
14	Victuailles en stock	1998-01-23 00:00:00
15	Antonio Moreno Taquería	1998-01-28 00:00:00
16	Seven Seas Imports	1998-02-04 00:00:00
17	Let's Stop N Shop	1998-02-12 00:00:00
18	Du monde entier	1998-02-16 00:00:00
19	Berglunds snabbköp	1998-03-04 00:00:00
20	Ana Trujillo Emparedados y helados	1998-03-04 00:00:00
21	Galería del gastrónomo	1998-03-05 00:00:00
22	Island Trading	1998-03-06 00:00:00
23	Wellington Importadora	1998-03-09 00:00:00
24	Morgenstern Gesundkost	1998-03-12 00:00:00
25	Magazzini Alimentari Riuniti	1998-03-16 00:00:00
26	Furia Bacalhau e Frutos do Mar	1998-03-19 00:00:00
27	Toms Spezialitäten	1998-03-23 00:00:00
28	La corne d'abondance	1998-03-24 00:00:00
29	France restauration	1998-03-24 00:00:00
30	Bólido Comidas preparadas	1998-03-24 00:00:00
31	La corne d'abondance	1998-03-24 00:00:00
32	Split Rail Beer & Ale	1998-03-25 00:00:00
33	Océano Atlántico Ltda.	1998-03-30 00:00:00
34	Que Delícia	1998-03-31 00:00:00
35	The Big Cheese	1998-04-01 00:00:00
36	Vaffeljernet	1998-04-02 00:00:00
37	The Cracker Box	1998-04-06 00:00:00
38	Wilman Kala	1998-04-07 00:00:00
39	Maison Dewey	1998-04-07 00:00:00
40	Princesa Isabel Vinhos	1998-04-08 00:00:00
41	Frankenversand	1998-04-09 00:00:00
42	Romero y tomillo	1998-04-09 00:00:00
43	Alfreds Futterkiste	1998-04-09 00:00:00
44	Around the Horn	1998-04-10 00:00:00
45	Santé Gourmet	1998-04-10 00:00:00
46	Rancho grande	1998-04-13 00:00:00
47	Lonesome Pine Restaurant	1998-04-13 00:00:00
48	Ottilies Käseladen	1998-04-14 00:00:00
49	QUICK-Stop	1998-04-14 00:00:00
50	B's Beverages	1998-04-14 00:00:00
51	Wartian Herkku	1998-04-15 00:00:00
52	Königlich Essen	1998-04-16 00:00:00
53	Old World Delicatessen	1998-04-20 00:00:00
54	LINO-Delicateses	1998-04-21 00:00:00
55	Godos Cocina Típica	1998-04-21 00:00:00
56	Suprêmes délices	1998-04-21 00:00:00
57	Chop-suey Chinese	1998-04-22 00:00:00
58	Comércio Mineiro	1998-04-22 00:00:00
59	Spécialités du monde	1998-04-22 00:00:00
60	Die Wandernde Kuh	1998-04-23 00:00:00
61	Wolski Zajazd	1998-04-23 00:00:00
62	Bottom-Dollar Markets	1998-04-24 00:00:00
63	Gourmet Lanchonetes	1998-04-24 00:00:00
64	Folk och fä HB	1998-04-27 00:00:00
65	Hanari Carnes	1998-04-27 00:00:00
66	La maison d'Asie	1998-04-27 00:00:00
68	Cactus Comidas para llevar	1998-04-28 00:00:00
69	Eastern Connection	1998-04-28 00:00:00
70	HILARION-Abastos	1998-04-28 00:00:00
71	Ricardo Adocicados	1998-04-29 00:00:00
72	Blauer See Delikatessen	1998-04-29 00:00:00
73	North/South	1998-04-29 00:00:00
74	Great Lakes Food Market	1998-04-30 00:00:00
75	Reggiani Caseifici	1998-04-30 00:00:00
76	Franchi S.p.A.	1998-04-30 00:00:00
77	Hungry Owl All-Night Grocers	1998-04-30 00:00:00
78	White Clover Markets	1998-05-01 00:00:00
79	Save-a-lot Markets	1998-05-01 00:00:00
80	Tortuga Restaurante	1998-05-04 00:00:00
81	Queen Cozinha	1998-05-04 00:00:00
82	Drachenblut Delikatessen	1998-05-04 00:00:00
83	Lehmanns Marktstand	1998-05-05 00:00:00
84	Ernst Handel	1998-05-05 00:00:00
85	LILA-Supermercado	1998-05-05 00:00:00
86	Pericles Comidas clásicas	1998-05-05 00:00:00
87	Richter Supermarkt	1998-05-06 00:00:00
88	Rattlesnake Canyon Grocery	1998-05-06 00:00:00
89	Bon app'	1998-05-06 00:00:00
90	Simons bistro	1998-05-06 00:00:00

Total rows: 90 | Query complete 00:00:00.062

RESULT ANALYSIS:

- The data lists **each customer's most recent order date** using a window function.
- The **latest order date** in the dataset is **1998-04-27**.
- Many customers haven’t placed an order since **early 1998**, indicating a need for re-engagement.

RECOMMENDATION:

1. **Prioritize Outreach** to customers who haven't ordered since early 1998 to revive engagement.
2. **Segment Customers** by recency tiers (e.g., 3+ months inactive) for targeted follow-up campaigns.
3. **Automate Reminders** to proactively nudge customers before long gaps in ordering occur.

PROBLEM STATEMENT 5: CUMULATIVE SALES BY MONTH

Business Scenario: The finance department is tracking sales trends over time. They want a month-by-month sales report for 1997 that includes the running total revenue up to the end of each month. This cumulative total helps visualize growth and determine if sales targets are being met as the year progresses.

Question: *How can we calculate the cumulative year-to-date sales total at the end of each month in 1997?*

List each month of 1997, the sales for that month, and the running total of sales through that month.

STRATEGIC APPROACH:

1. **Schema:**

   To solve this, we need to join the following tables:

   - orders: contains order_date, order_id
   - order_details: contains unit_price, quantity, discount, order_id

2. **Join Orders with Order Details**

   - Link orders and order_details using order_id
   - This will calculate revenue per order line

3. **Filter to Only Orders from 1997**

   - Use a filter like WHERE EXTRACT(YEAR FROM order_date) = 1997

4. **Group by Month**

   - Extract month from order_date
   - Group by month and sum revenue using: unit_price * quantity * (1 - discount)

- This will give **monthly sales**

5. **Use a Window Function for the Running Total**

- Used SUM(monthly_sales) OVER (ORDER BY month) to get the cumulative sum • This provides the year-to-date total as each month progresses

QUERY:

```
WITH monthly_sales AS (
    SELECT
        DATE_TRUNC('month', o."orderDate") AS month,
--monthly sales
        SUM(od."unitPrice" * od."quantity" * (1 - od."discount")) AS
monthly_revenue     FROM
        northwind_traders."orders" AS o
        JOIN northwind_traders."order_details" AS od
        ON o."orderID" = od."orderID"
    WHERE
        EXTRACT(YEAR FROM o."orderDate") = 1997
    GROUP BY
        DATE_TRUNC('month', o."orderDate")
)
SELECT
    TO_CHAR(month, 'YYYY-MM') AS month,
    ROUND(monthly_revenue, 2) AS monthly_sales,
--to get the cumulative sum
    ROUND(SUM(monthly_revenue) OVER (ORDER BY month), 2) AS cumulative_sales
FROM

monthly_sales
ORDER BY
month;
```

OUTPUT:

	month text	monthly_sales numeric	cumulative_sales numeric
1	1997-01	61258.07	61258.07
2	1997-02	38483.64	99741.71
3	1997-03	38547.22	138288.93
4	1997-04	53032.95	191321.88
5	1997-05	53781.29	245103.17
6	1997-06	36362.80	281465.97
7	1997-07	51020.86	332486.83
8	1997-08	47287.67	379774.50
9	1997-09	55629.24	435403.74
10	1997-10	66749.23	502152.97
11	1997-11	43533.81	545686.78
12	1997-12	71398.43	617085.20

Total rows: 12 | Query complete 00:00:00.081

RESULT ANALYSIS:

- **Total sales in 1997** reached **617,085.20** units of currency.
- The **highest sales month** was **December (71,398.43)**, suggesting strong end-of-year performance—possibly due to holiday demand.
- **Lowest sales** occurred in **June (36,362.80)**, which could indicate a seasonal dip.
- Sales grew steadily through the year, with **notable growth from September to December**, where cumulative sales rose by over **180K** in just four months.
- The growth trend is **non-linear**, showing volatility—especially between February, March, and April, followed by a recovery mid-year.

RECOMMENDATION:

1. **Capitalize on Year-End Momentum**: Increase marketing spend and promotions in Q4 to amplify already strong sales trends.
2. **Investigate June Dip**: Explore internal and external factors behind the June sales drop (e.g., fewer promotions, supply chain issues).
3. **Set Quarterly Benchmarks**: Use cumulative targets per quarter to better align sales efforts and track against goals.

PROBLEM STATEMENT 6: DAYS BETWEEN CUSTOMER ORDERS

Business Scenario: Marketing analysts are studying **customer reorder patterns**. For each order a customer makes (after their first), they want to know how many days have passed since that customer's previous order. This helps identify purchasing frequency—whether customers order weekly, monthly, etc.—to tailor marketing communications.

Question: *For each customer order (except their first), how many days elapsed since the customer's prior order?*

Show the customer name, the order date, and the number of days since that customer's previous order. Use the LAG window function to access the date of the prior order.

STRATEGIC APPROACH:

1. **Schema:**

   To solve this, we need to join the following tables:

   - orders: contains order_id, order_date, customer_id
   - customers: contains customer_id, company_name

2. **Join Orders with Customers**

   - Join orders with customers on customer_id
   - This will give access to both order_date and company_name

3. **Use the LAG() Function**

   - Use LAG(order_date) to get the **previous order's date** for the same customer
   - Use PARTITION BY customer_id so the lag only considers **that customer's history**
   - Use ORDER BY order_date to ensure the order sequence is correct

4. **Calculate Days Between Orders**

- Subtract: order_date - previous_order_date
- This gives the **number of days since the last order**

QUERY:

```
WITH customer_orders AS (
    SELECT
        c."companyName" AS customer_name,
        o."orderDate",
--get the previous order's date for the same customer
LAG(o."orderDate") OVER (PARTITION BY o."customerID" ORDER BY
o."orderDate") AS previous_order_date
FROM
        northwind_traders."orders" AS o
        JOIN northwind_traders."customers" AS c
        ON o."customerID" = c."customerID"
)
SELECT
    customer_name,
    "orderDate",
--This gives the number of days since the last order
    ("orderDate" - previous_order_date) AS days_since_last_order
FROM
    customer_orders WHERE
    previous_order_date IS NOT NULL ORDER
BY
    customer_name, "orderDate";
```

OUTPUT:

	customer_name character varying	orderDate timestamp without time zone	days_since_last_order interval
1	Alfreds Futterkiste	1997-10-03 00:00:00	39 days
2	Alfreds Futterkiste	1997-10-13 00:00:00	10 days
3	Alfreds Futterkiste	1998-01-15 00:00:00	94 days
4	Alfreds Futterkiste	1998-03-16 00:00:00	60 days
5	Alfreds Futterkiste	1998-04-09 00:00:00	24 days
6	Ana Trujillo Emparedados y helados	1997-08-08 00:00:00	324 days
7	Ana Trujillo Emparedados y helados	1997-11-28 00:00:00	112 days
8	Ana Trujillo Emparedados y helados	1998-03-04 00:00:00	96 days
9	Antonio Moreno Taquería	1997-04-15 00:00:00	139 days
10	Antonio Moreno Taquería	1997-05-13 00:00:00	28 days
11	Antonio Moreno Taquería	1997-06-19 00:00:00	37 days
12	Antonio Moreno Taquería	1997-09-22 00:00:00	95 days
13	Antonio Moreno Taquería	1997-09-25 00:00:00	3 days
14	Antonio Moreno Taquería	1998-01-28 00:00:00	125 days
15	Around the Horn	1996-12-16 00:00:00	31 days
16	Around the Horn	1997-02-21 00:00:00	67 days

Total rows: 741 Query complete 00:00:00.091

RESULT ANALYSIS:

- **Wide variability in reordering behavior**:
  - **Alfreds Futterkiste** shows both short gaps (10 days) and longer ones (94 days).
  - **Ana Trujillo Emperadados y helados** had a **324-day gap**, indicating potential churn risk or seasonality.
- Some customers like **Antonio Moreno Taquería** had highly irregular patterns—ranging from **3 days to 139 days**.

  Around the Horn had orders spaced **31 and 67 days apart**, suggesting a **monthly cadence**.

RECOMMENDATION:

1. **Segment by Frequency**: Group customers by reorder intervals (e.g., <30 days = frequent, 30–90 = moderate, >90 = infrequent) to tailor campaigns.
2. **Re-engage Inactive Customers**: Target those with >90-day gaps using win-back promotions or personalized outreach.
3. **Predictive Reminders**: Set automated reminders or discounts **just before the average reorder window** to nudge repeat purchases.

PROBLEM STATEMENT 7: NEXT ORDER DATE AND REORDER INTERVAL

Business Scenario: Continuing the analysis of customer ordering habits, the team now looks forward. After each order, they want to know **when the next order from the same customer occurred** and the gap in days between the two orders. This forward-looking gap (until the next order) can indicate how quickly customers come back to buy again.

Question: *For each customer order (except their last), what was the date of the next order by that same customer, and how many days later did it occur?*
Show the customer name, the current order date, the next order date, and the interval in days. Use the LEAD function to get the next order's date.

STRATEGIC APPROACH:

1. **Schema:**

   To solve this, we need to join the following tables:

   - orders: contains order_id, order_date, customer_id
   - customers: contains customer_id, company_name

2. **Join Orders with Customers**

   - Join orders with customers on customer_id
   - This will give both the customer name and their order date

3. **Use the LEAD() Window Function**

   - Use LEAD(order_date) to get the **next order date**
   - Use PARTITION BY customer_id, only look at orders within the **same customer**
   - Use ORDER BY order_date to ensure order sequence is chronological

4. **Calculate the Reorder Interval**

- Subtract: next_order_date - current_order_date
- This gives the **number of days** until the next order

5. **Use a Window Function for the Running Total**

- Used SUM(monthly_sales) OVER (ORDER BY month) to get the cumulative sum
- This provides the year-to-date total as each month progresses.

QUERY:

```sql
WITH customer_orders AS (
    SELECT
        c."companyName" AS customer_name,
        o."orderDate" AS current_order_date,
--to get the next order date
        LEAD(o."orderDate") OVER (
            PARTITION BY o."customerID"
            ORDER BY o."orderDate"
        ) AS next_order_date
FROM
        northwind_traders."orders" o
        JOIN northwind_traders."customers" c
            ON o."customerID" = c."customerID"
) SELECT
    customer_name,
current_order_date,
next_order_date,
    -- Calculate the gap in days  number of days until the next order
next_order_date - current_order_date AS days_until_next_order FROM
    customer_orders WHERE
    next_order_date IS NOT NULL
ORDER BY     customer_name,
current_order_date;
```

OUTPUT:

	customer_name character varying	current_order_date timestamp without time zone	next_order_date timestamp without time zone	days_until_next_order interval
1	Alfreds Futterkiste	1997-08-25 00:00:00	1997-10-03 00:00:00	39 days
2	Alfreds Futterkiste	1997-10-03 00:00:00	1997-10-13 00:00:00	10 days
3	Alfreds Futterkiste	1997-10-13 00:00:00	1998-01-15 00:00:00	94 days
4	Alfreds Futterkiste	1998-01-15 00:00:00	1998-03-16 00:00:00	60 days
5	Alfreds Futterkiste	1998-03-16 00:00:00	1998-04-09 00:00:00	24 days
6	Ana Trujillo Emparedados y helados	1996-09-18 00:00:00	1997-08-08 00:00:00	324 days
7	Ana Trujillo Emparedados y helados	1997-08-08 00:00:00	1997-11-28 00:00:00	112 days
8	Ana Trujillo Emparedados y helados	1997-11-28 00:00:00	1998-03-04 00:00:00	96 days
9	Antonio Moreno Taquería	1996-11-27 00:00:00	1997-04-15 00:00:00	139 days
10	Antonio Moreno Taquería	1997-04-15 00:00:00	1997-05-13 00:00:00	28 days
11	Antonio Moreno Taquería	1997-05-13 00:00:00	1997-06-19 00:00:00	37 days
12	Antonio Moreno Taquería	1997-06-19 00:00:00	1997-09-22 00:00:00	95 days
13	Antonio Moreno Taquería	1997-09-22 00:00:00	1997-09-25 00:00:00	3 days

Total rows: 741 | Query complete 00:00:00.138

RESULT ANALYSIS:

- **Alfreds Futterkiste** maintains a mostly regular reordering cycle (10–94 days) with **an average of ~45 days** between orders.
- **Ana Trujillo** has **long intervals (up to 324 days)**, suggesting either seasonal buying patterns or reduced engagement.
- **Antonio Moreno Taquería** appears very active over short bursts, with orders just **3 days apart**, but also has gaps over **130 days**.

RECOMMENDATION:

1. **Use LEAD-based reorder gaps for forecasting**: Predict when a customer is likely to reorder again and proactively send reminders.
2. **Implement smart nudges**: If the reorder gap exceeds their historical average, trigger a promo email or check-in campaign.
3. **Create customer lifecycle stages**: Classify customers by expected return window (e.g., “likely to reorder in 30 days”) for CRM automation.

PROBLEM STATEMENT 8: HIGHEST-VALUE ORDER AND ITS SALESPERSON

Business Scenario: Senior management wants to highlight the single largest order in terms of revenue, and recognize the employee who handled it. Knowing which order brought in the most money (and who was responsible for it) can be useful for awards or case studies on successful sales.

Question: *Which order had the highest total value, and which employee handled that order?*
Provide the order ID, the total order amount, and the full name of the employee who handled it. Use an aggregate subquery or CTE to identify the maximum order total.

STRATEGIC APPROACH:

1. **Schema:**

   To solve this, we need to join the following tables:

   - orders: contains order_id, employee_id
   - order_details: orderID, unitPrice, quantity, discount
   - Employees: employee_id, firstName, lastName

2. **Calculate Total Value per Order**

   - Use the order_details table to compute:
   - total_order_value = unit_price × quantity × (1 - discount)
   - Group this by orderID to get **total revenue per order**.

3. **Join with Employee Info**

   - **Use the orders table to:**
   - **Link each orderID to the employeeID.**
   - Join with the employees table to get the **employee's full name**.

4. **Find the Maximum Order Value**

    - Use either:
    - A **CTE** or A **subquery**
    - **Find the maximum of all total_order_values.**

5. **Filter for the Highest Order**

    - Compare the total_order_value from each order with the **maximum** found above.
    - Return the **order ID**, **total value**, and the **employee name** for that highest-value order.

QUERY:

```
WITH order_totals AS (   --calculating the total revenue per order.     SELECT
        o."orderID",
        e."firstName" || ' ' || e."lastName" AS employee_name,
SUM(od."unitPrice" * od."quantity" * (1 - od."discount")) AS total_order_value
FROM
        northwind_traders."orders" o
        JOIN northwind_traders."order_details" od ON o."orderID" = od."orderID"
        JOIN northwind_traders."employees" e ON o."employeeID" = e."employeeID"
    GROUP BY
        o."orderID", e."firstName", e."lastName"  -- to get totals per order.
),
--This part finds the maximum order value across all orders from the previous
step. max_order AS (
    SELECT
        MAX(total_order_value) AS max_value
    FROM









        order_totals
)
--join the order_totals CTE with the max_order CTE. SELECT
    ot."orderID",
ot.total_order_value,
ot.employee_name FROM
    order_totals ot
    JOIN max_order mo ON ot.total_order_value = mo.max_value;
```

OUTPUT :

	orderID integer	total_order_value numeric	employee_name text
1	10865	16387.500	Andrew Fuller

Total rows: 1 | Query complete 00:00:00.102

RESULT ANALYSIS:

- **Order ID**: 10865
- **Total Order Value**: **16,387.50**
- **Employee**: **Andrew Fuller**

- Andrew Fuller handled the **highest-value order** in the dataset, making him a **top-performing salesperson** for high-revenue deals. This order likely represents a significant client or bulk purchase and can be studied as a **model case for successful sales**.

RECOMMENDATION:

1. **Recognize & Reward**: Highlight Andrew Fuller for this high-value achievement to boost motivation across the team.
2. **Analyze this deal further**: Understand the customer, product mix, and discount structure for replicable strategies.
3. **Train using top performance cases**: Use this order as a case study in sales training programs.