Title :- Northwind Traders: SQL-Based Business Insights Report
Sub Title :- A Comprehensive Data Analysis of Sales, Customers, and Suppliers using PostgreSQL
By :- Pratik Garud
Date :- 30/10/2025

```
-- Question 1: Second-Best Selling Product by Category

WITH my_cte AS (
        SELECT c.category_name,p.product_name,SUM(od.unit_price * od.quantity * (1 -
COALESCE(od.discount,0))) AS total_revenue FROM categories c
JOIN products p
ON c.category_id = p.category_id
JOIN order_details od
ON p.product_id = od.product_id
GROUP BY c.category_name,p.product_name
),

ranked AS (
 SELECT
  *,
  ROW_NUMBER() OVER (PARTITION BY category_name ORDER BY total_revenue
DESC) AS rn
 FROM my_cte
)

SELECT
 category_name,
 product_name,
 total_revenue
FROM ranked
WHERE rn = 2
ORDER BY category_name;
```

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

Explanation :-
This query identifies the **second-highest revenue-generating product** within each product category.
First, total revenue for each product is calculated using the formula: *Unit Price* × *Quantity* × *(1 - Discount)*.
Then, products are ranked within their respective categories by revenue using a window function (DENSE_RANK).
Finally, the query filters to show only those products ranked second, revealing items that perform strongly but are not the top seller in their category.

```
-- Question 2: Top 3 Customers by Total Sales

WITH my_cte AS (
SELECT o.customer_id,
        cs.contact_name as customer_name,
SUM(od.unit_price * od.quantity * (1 - COALESCE(od.discount,0))) AS total_spent
FROM order_details od
JOIN orders o
ON od.order_id = o.order_id
JOIN customers cs
ON cs.customer_id = o.customer_id
GROUP BY o.customer_id,cs.contact_name
),

ranked AS (
SELECT * ,
DENSE_RANK() over (order by total_spent DESC) as rn
from my_cte
)

SELECT customer_name,total_spent,rn from ranked
where rn<=3
```

	customer_name character varying	total_spent numeric	rn bigint
1	Horst Kloss	110277.3050	1
2	Roland Mendel	104874.9785	2
3	Jose Pavarotti	104361.9500	3

Explanation :- This analysis determines the **top three customers based on total purchase value**.
Customer spending is aggregated from all their orders to compute total sales revenue.
A ranking function (RANK) is applied to order customers by spending in descending order.
If multiple customers tie for third place, they are all included in the results.
The output helps the company identify high-value customers for loyalty programs or personalized offers.

--Question 3: Top category by Product Variety

```
WITH category_product_count AS (
  SELECT
    c.category_id,
    c.category_name,
    COUNT(p.product_id) AS product_count
  FROM categories c
  JOIN products p ON p.category_id = c.category_id
  GROUP BY c.category_id, c.category_name
)
SELECT
  category_name,
  product_count,
  DENSE_RANK() OVER (ORDER BY product_count DESC) AS category_rank
FROM category_product_count
ORDER BY product_count DESC;
```

	category_name character varying	product_count bigint	category_rank bigint
1	Confections	13	1
2	Condiments	12	2
3	Seafood	12	2
4	Beverages	12	2
5	Dairy Products	10	3
6	Grains & Cereals	7	4
7	Meat & Poultry	6	5
8	Produce	5	6

Explanation :- Here, suppliers are evaluated based on the **number of unique products** they provide.
Each supplier's product count is calculated and ranked using the DENSE_RANK function.
This ensures suppliers with the same product count share the same rank without gaps in numbering.
The resulting list highlights which suppliers offer the broadest product catalog, helping procurement teams assess partnership value.

--Question 4: Most Recent Order per Customer

SELECT distinct c.contact_name,max(o.order_date) AS most_recent_order_date from orders o
JOIN customers c
ON c.customer_id = o.customer_id
group by c.contact_name
ORDER BY most_recent_order_date

	contact_name character varying	most_recent_order_date date
1	Francisco Chang	2013-07-18
2	John Steel	2014-05-22
3	Yoshi Latimer	2014-09-08
4	Jean Fresnière	2014-10-30
5	Aria Cruz	2014-10-31
6	Paul Henriot	2014-11-12
7	Manuel Pereira	2014-12-18

	contact_name character varying	most_recent_order_date date
80	Miguel Angel Paolino	2015-05-04
81	Sven Ottlieb	2015-05-04
82	Carlos González	2015-05-05
83	Guillermo Fernández	2015-05-05
84	Renate Messner	2015-05-05
85	Roland Mendel	2015-05-05
86	Jytte Petersen	2015-05-06
87	Laurence Lebihan	2015-05-06
88	Michael Holz	2015-05-06
89	Paula Wilson	2015-05-06

Explanation :- This query finds the **latest order date** for each customer.
Using a window function (ROW_NUMBER) partitioned by customer, orders are sorted in descending order of date.
Only the most recent order (rank 1) for each customer is selected.
This information is useful for the customer relations team to identify inactive customers and plan re-engagement campaigns.

--Question 5: Cumulative Sales by Month

```
WITH monthly_sales AS (
  SELECT
    date_trunc('month', o.order_date) AS month_start,
    SUM(od.unit_price * od.quantity * (1 - COALESCE(od.discount,0))) AS month_sales
  FROM orders o
  JOIN order_details od ON o.order_id = od.order_id
  WHERE o.order_date >= '2014-01-01'::date
    AND o.order_date <  '2015-01-01'::date
  GROUP BY date_trunc('month', o.order_date)
)
SELECT
  to_char(month_start, 'YYYY-MM') AS month,
  month_sales,
  SUM(month_sales) OVER (ORDER BY month_start
                         ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS
running_total_ytd
FROM monthly_sales
ORDER BY month_start;
```

	month text	month_sales numeric	running_total_ytd numeric
1	2014-01	61258.070	61258.070
2	2014-02	38483.635	99741.705
3	2014-03	38547.220	138288.925
4	2014-04	53032.9525	191321.8775
5	2014-05	53781.2900	245103.1675
6	2014-06	36362.8025	281465.9700
7	2014-07	51020.8575	332486.8275
8	2014-08	47287.6700	379774.4975
9	2014-09	55629.2425	435403.7400
10	2014-10	66749.2260	502152.9660
11	2014-11	43533.8090	545686.7750
12	2014-12	71398.4285	617085.2035

Explanation :-

This report provides a **month-by-month sales summary for the year 2014** and includes a cumulative (year-to-date) total.
Sales for each month are calculated by grouping order data by month, then applying a

running total using the SUM window function.
The cumulative value shows how sales grew over time, enabling the finance department to monitor performance against targets and track growth trends throughout the year.

--Question 6: Days Between Customer Orders

```
WITH ordered AS (
  SELECT
    o.order_id,
    o.customer_id,
    c.contact_name AS customer_name,
    o.order_date::date AS order_date,
    LAG(o.order_date::date) OVER (PARTITION BY o.customer_id ORDER BY o.order_date,
o.order_id) AS prev_order_date
  FROM orders o
  JOIN customers c ON o.customer_id = c.customer_id
)
SELECT
  customer_name,
  order_date,
  (order_date - prev_order_date) AS days_since_prev_order
FROM ordered
WHERE prev_order_date IS NOT NULL
ORDER BY customer_name, order_date;
```

	customer_name character varying	order_date date	days_since_prev_order integer
1	Alejandra Camino	2013-08-15	1
2	Alejandra Camino	2013-09-16	32
3	Alejandra Camino	2015-03-02	532
4	Alejandra Camino	2015-04-09	38
5	Alexander Feuer	2014-06-20	315
6	Alexander Feuer	2014-10-09	111
7	Alexander Feuer	2014-12-16	68
8	Alexander Feuer	2015-03-12	86
9	Ana Trujillo	2014-08-08	324
10	Ana Trujillo	2014-11-28	112

	customer_name character varying	order_date date	days_since_prev_order integer
732	Yvonne Moncada	2014-05-08	119
733	Yvonne Moncada	2015-02-20	288
734	Yvonne Moncada	2015-03-18	26
735	Yvonne Moncada	2015-03-30	12
736	Zbyszek Piestrzeniewicz	2014-07-25	232
737	Zbyszek Piestrzeniewicz	2014-12-23	151
738	Zbyszek Piestrzeniewicz	2015-02-04	43
739	Zbyszek Piestrzeniewicz	2015-02-25	21
740	Zbyszek Piestrzeniewicz	2015-04-03	37
741	Zbyszek Piestrzeniewicz	2015-04-23	20

Explanation :-

This analysis calculates the **number of days between consecutive orders** for each customer.

The LAG window function retrieves the date of the previous order within each customer's order history.
The difference between the current and previous order dates indicates how frequently customers reorder.
This insight helps marketing teams understand purchasing behavior and plan follow-up or reminder campaigns accordingly.

--Question 7: Next Order Date and Reorder Interval

```
WITH ordered_forward AS (
  SELECT
    o.order_id,
    o.customer_id,
    c.contact_name AS customer_name,
    o.order_date::date AS order_date,
    LEAD(o.order_date::date) OVER (PARTITION BY o.customer_id ORDER BY
o.order_date, o.order_id) AS next_order_date
  FROM orders o
  JOIN customers c ON o.customer_id = c.customer_id
)
SELECT
  customer_name,
  order_date AS current_order_date,
  next_order_date,
  (next_order_date - order_date) AS days_until_next_order
FROM ordered_forward
WHERE next_order_date IS NOT NULL
ORDER BY customer_name, order_date;
```

	customer_name character varying	current_order_date date	next_order_date date	days_until_next_order integer
1	Alejandra Camino	2013-08-14	2013-08-15	1
2	Alejandra Camino	2013-08-15	2013-09-16	32
3	Alejandra Camino	2013-09-16	2015-03-02	532
4	Alejandra Camino	2015-03-02	2015-04-09	38
5	Alexander Feuer	2013-08-09	2014-06-20	315
6	Alexander Feuer	2014-06-20	2014-10-09	111
7	Alexander Feuer	2014-10-09	2014-12-16	68
8	Alexander Feuer	2014-12-16	2015-03-12	86
9	Ana Trujillo	2013-09-18	2014-08-08	324
10	Ana Trujillo	2014-08-08	2014-11-28	112

	customer_name character varying	current_order_date date	next_order_date date	days_until_next_order integer
732	Yvonne Moncada	2014-01-09	2014-05-08	119
733	Yvonne Moncada	2014-05-08	2015-02-20	288
734	Yvonne Moncada	2015-02-20	2015-03-18	26
735	Yvonne Moncada	2015-03-18	2015-03-30	12
736	Zbyszek Piestrzeniewicz	2013-12-05	2014-07-25	232
737	Zbyszek Piestrzeniewicz	2014-07-25	2014-12-23	151
738	Zbyszek Piestrzeniewicz	2014-12-23	2015-02-04	43
739	Zbyszek Piestrzeniewicz	2015-02-04	2015-02-25	21
740	Zbyszek Piestrzeniewicz	2015-02-25	2015-04-03	37
741	Zbyszek Piestrzeniewicz	2015-04-03	2015-04-23	20

Explanation :- This query identifies the **date of the next order** after each purchase and the **time gap (in days)** until that next order.
It uses the LEAD window function to access the following order date for each customer.
By subtracting the current order date from the next, the company can determine

average reorder intervals.
This forward-looking metric helps forecast customer demand and measure loyalty or repeat purchase tendencies.

--Question 8: Highest-Value Order and Its Salesperson

```
WITH order_totals AS (
  SELECT
    o.order_id,
    o.employee_id,
    SUM(od.unit_price * od.quantity * (1 - COALESCE(od.discount,0))) AS order_total
  FROM orders o
  JOIN order_details od ON o.order_id = od.order_id
  GROUP BY o.order_id, o.employee_id
),
max_order AS (
  SELECT order_id, employee_id, order_total
  FROM order_totals
  ORDER BY order_total DESC
  LIMIT 1
)
SELECT
  mo.order_id,
  mo.order_total,
  e.employee_name
FROM max_order mo
LEFT JOIN employees e ON mo.employee_id = e.employee_id;
```

	order_id numeric	order_total numeric	employee_name character varying
1	10865	16387.500	Andrew Fuller

Explanation :-
This query determines the **single largest order by total value** and identifies the **employee who managed it**.
Each order's total value is computed by summing line-item revenues.
The order with the highest total amount is then selected, and the corresponding employee's name is retrieved.

This insight highlights exceptional sales performance and can be used for internal recognition or case studies on successful transactions.