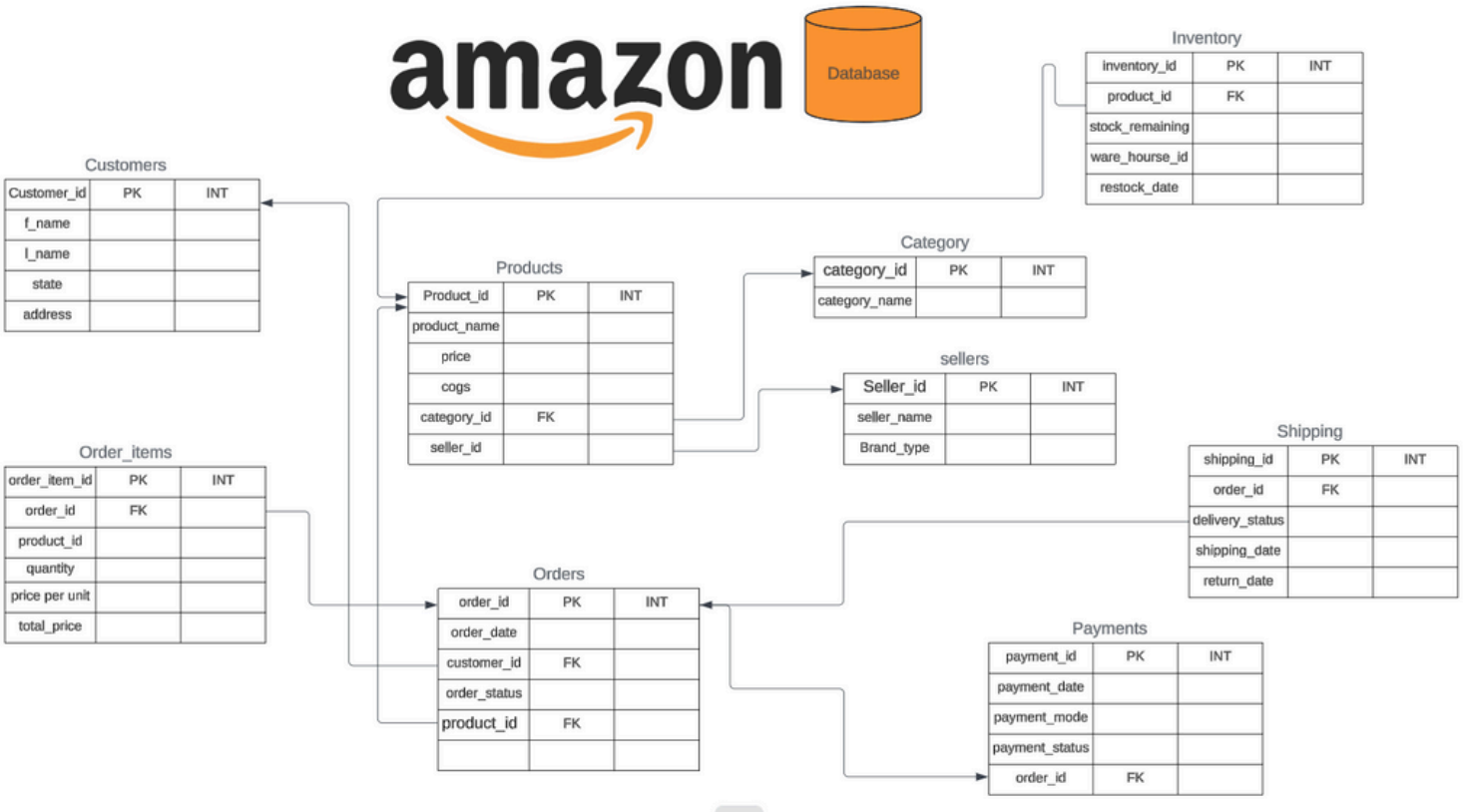
AMAZON SALES DATA ANALYSIS



OVERVIEW

I have worked on analyzing a dataset of over 20,000 sales records from an Amazon-like e-commerce platform. This project involves extensive querying of customer behavior, product performance, and sales trends using PostgreSQL. Through this project, I have tackled various SQL problems, including revenue analysis, customer segmentation, and inventory management.

LEARNING OUTCOMES

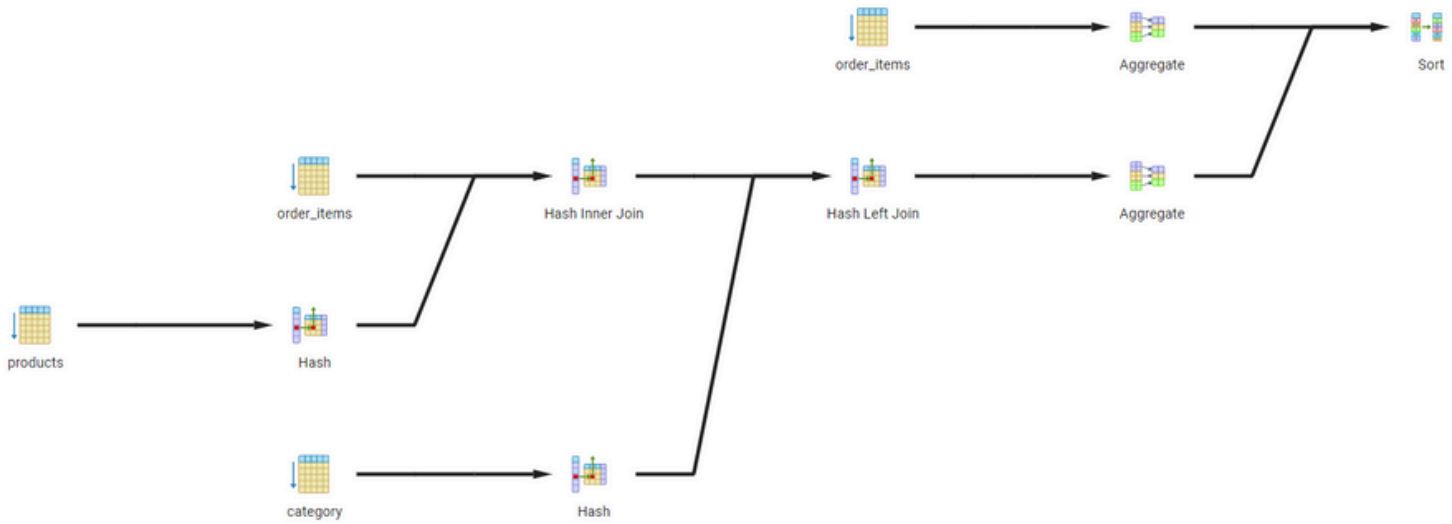
- Design and implement a normalized database schema.
- Clean and preprocess real-world datasets for analysis.
- Use advanced SQL techniques, including window functions, subqueries, and joins.
- Conduct in-depth business analysis using SQL.
- Optimize query performance and handle large datasets efficiently.

```sql
/*
1. Top Selling Products
Query the top 10 products by total sales value.
Challenge: Include product name, total quantity sold, and total sales value.
*/

-- Altering order_item tables to create total order value
ALTER TABLE order_items
ADD COLUMN total_order_value FLOAT;

-- Updating values in new column
UPDATE order_items
SET total_order_value = quantity*price_per_unit;
SELECT * FROM order_items

EXPLAIN ANALYZE
SELECT
    p.product_id,
    p.product_name,
    SUM(oi.quantity) AS sold_quantity,
    SUM(oi.total_order_value) AS total_sales
FROM order_items oi
JOIN products p ON p.product_id = oi.product_id
GROUP BY p.product_id
ORDER BY total_sales DESC
LIMIT 10;
```

Graphical | Analysis | Statistics

#	Node
1.	→ Limit
2.	→ Sort
3.	→ Aggregate
4.	→ Hash Inner Join **Hash Cond**: (oi.product_id = p.product_id)
5.	→ Seq Scan on order_items as oi
6.	→ Hash
7.	→ Seq Scan on products as p

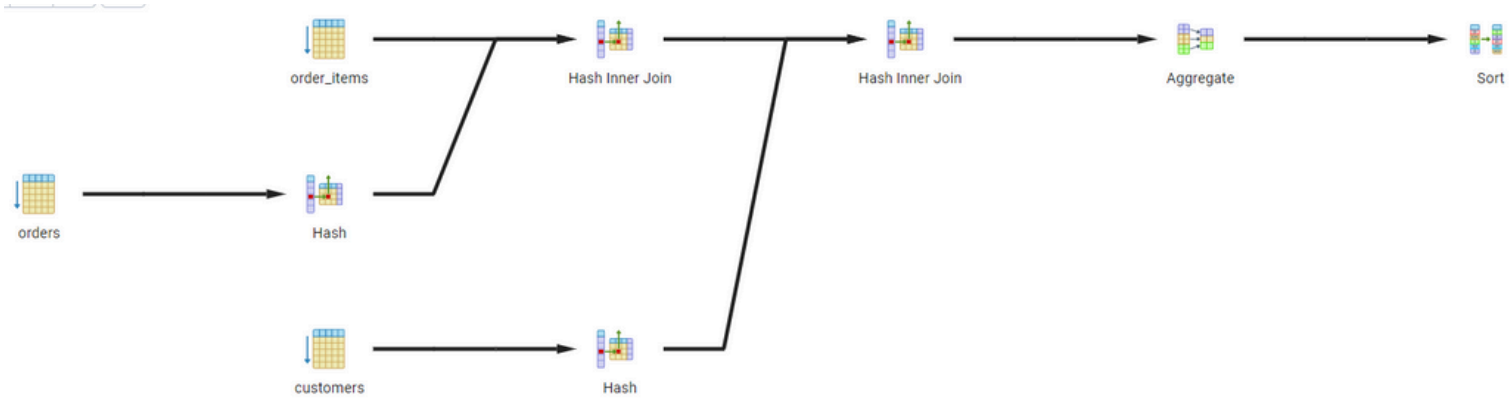
```sql
/*
2. Revenue by Category
Calculate total revenue generated by each product category.
Challenge: Include the percentage contribution of each category to total revenue.
*/

EXPLAIN ANALYZE
SELECT
c.category_id,
c.category_name,
SUM(oi.total_order_value) AS total_category_sales,
ROUND(100*(SUM(oi.total_order_value)/
(SELECT SUM(total_order_value) FROM order_items))) AS revenue_share
FROM order_items oi
JOIN products p
on p.product_id = oi.product_id
LEFT JOIN category c
on c.category_id = p.category_id
GROUP BY c.category_id
ORDER BY total_category_sales DESC;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Sort
2.	→ Aggregate
3.	→ Seq Scan on order_items as order_items
4.	→ Aggregate
5.	→ Hash Left Join **Hash Cond**: (p.category_id = c.category_id)
6.	→ Hash Inner Join **Hash Cond**: (oi.product_id = p.product_id)
7.	→ Seq Scan on order_items as oi
8.	→ Hash
9.	→ Seq Scan on products as p
10.	→ Hash
11.	→ Seq Scan on category as c

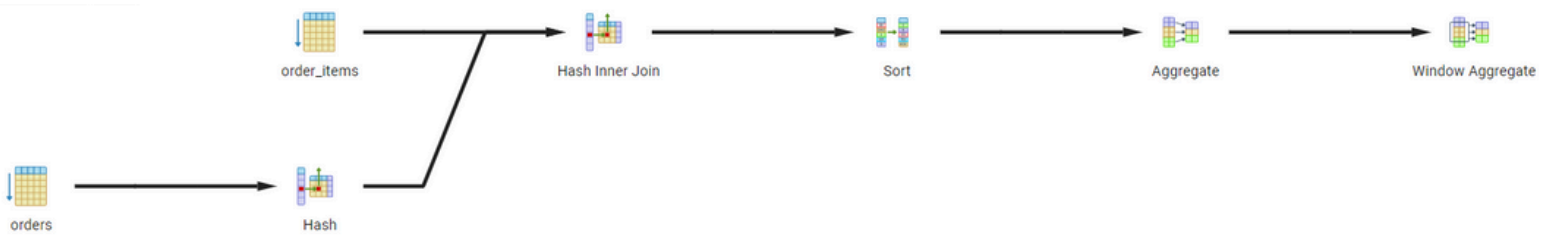
```sql
/*
3. Average Order Value (AOV)
Compute the average order value for each customer.
Challenge: Include only customers with more than 5 orders.
*/

EXPLAIN ANALYZE
SELECT
c.customer_id,
CONCAT(c.first_name,' ',c.last_name) AS full_name,
COUNT(o.order_id),
AVG(oi.total_order_value) AS AOV
FROM order_items oi
JOIN orders o
ON oi.order_id = o.order_id
JOIN customers c
ON c.customer_id = o.customer_id
GROUP BY c.customer_id
HAVING COUNT(o.order_id)>5
ORDER BY AOV DESC;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Sort
2.	→ Aggregate **Filter**: (count(o.order_id) > 5)
3.	→ Hash Inner Join **Hash Cond**: (o.customer_id = c.customer_id)
4.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o.order_id)
5.	→ Seq Scan on order_items as oi
6.	→ Hash
7.	→ Seq Scan on orders as o
8.	→ Hash
9.	→ Seq Scan on customers as c

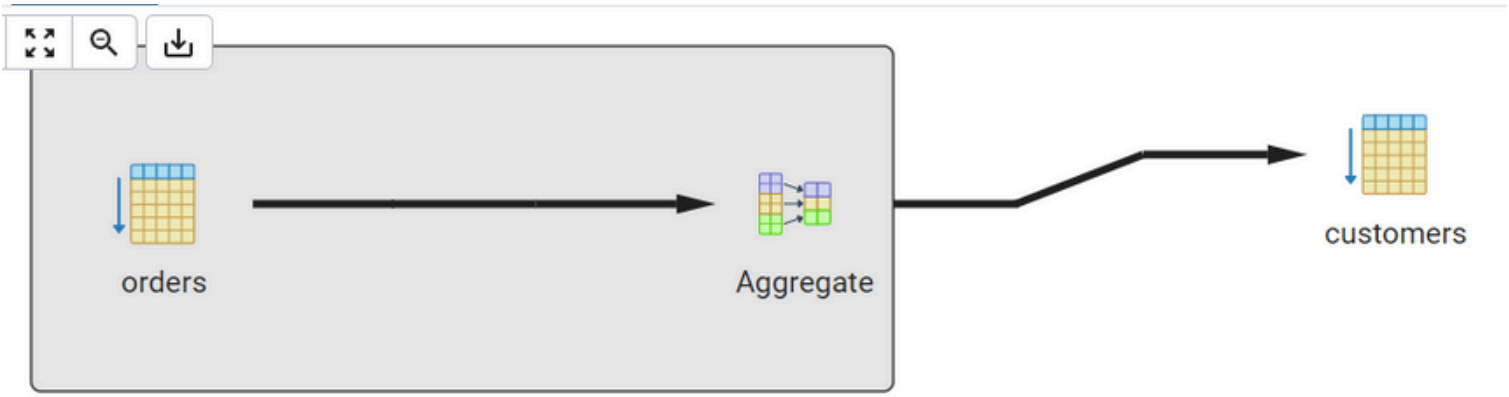
```sql
/*
4. Monthly Sales Trend
Query monthly total sales over the past year.
Challenge: Display the sales trend, grouping by month,
return current_month sale, last month sale!
*/

EXPLAIN ANALYZE
SELECT year,month,monthly_sales AS current_month_sales,
LAG(monthly_sales,1) OVER(ORDER BY year,month) AS last_month_sales
FROM
(SELECT
EXTRACT(YEAR FROM o.order_date) AS year,
EXTRACT(MONTH FROM o.order_date) AS month,
ROUND(SUM(oi.total_order_value::NUMERIC),2) AS monthly_sales
FROM orders o
JOIN order_items oi ON oi.order_id = o.order_id
WHERE o.order_date >= CURRENT_DATE - INTERVAL '1 YEAR'
GROUP BY 1,2
ORDER BY 1,2) AS t1;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Window Aggregate
2.	→ Aggregate
3.	→ Sort
4.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o.order_id)
5.	→ Seq Scan on order_items as oi
6.	→ Hash
7.	→ Seq Scan on orders as o **Filter**: (order_date >= (CURRENT_DATE - '1 year'::interval))

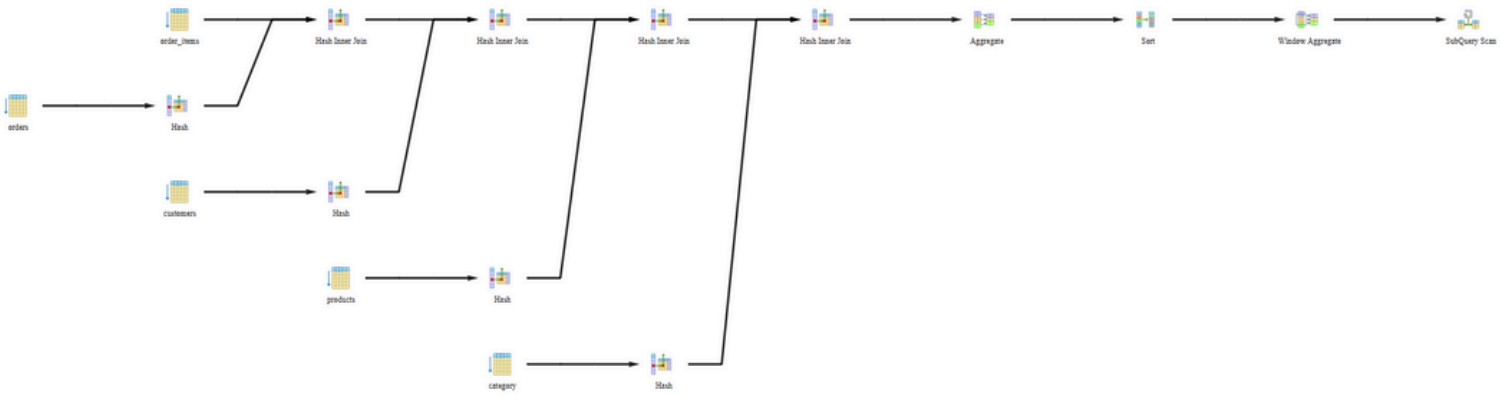
```sql
/*
5. Customers with No Purchases
Find customers who have registered but never placed an order.
Challenge: List customer details and the time since their registration.
*/

EXPLAIN ANALYZE
SELECT *
FROM customers
WHERE customer_id NOT IN
(SELECT DISTINCT customer_id FROM orders);
```



Graphical | Analysis | Statistics

#	Node
1.	→ Seq Scan on customers as customers **Filter**: (NOT (ANY (customer_id = (hashed SubPlan 1).col1)))
2.	→ Aggregate
3.	→ Seq Scan on orders as orders

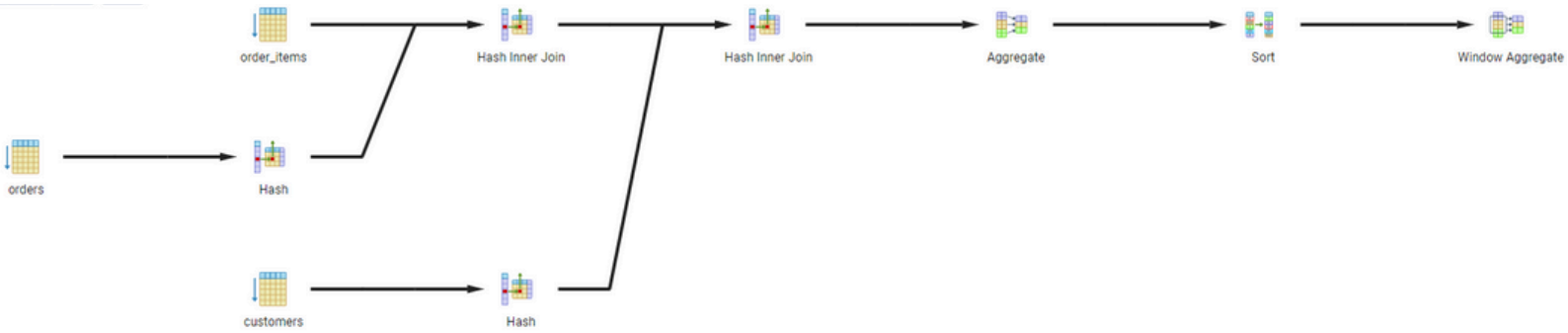
```sql
/*
6. Least-Selling Categories by State
Identify the least-selling product category for each state.
Challenge: Include the total sales for that category within each state.
*/

EXPLAIN ANALYZE
SELECT state,category,sales
FROM
(SELECT
c.state AS state,
pc.category_name AS category,
ROUND(SUM(oi.total_order_value::NUMERIC),2) as sales,
RANK() OVER(PARTITION BY state ORDER BY SUM(oi.total_order_value)) AS rnk
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.customer_id
JOIN order_items oi ON oi.order_id = o.order_id
JOIN products p ON p.product_id = oi.product_id
JOIN category pc ON pc.category_id = p.product_id
GROUP BY 1,2) AS t1
WHERE rnk=1;
```



Graphical | Analysis | Statistics

	Filter: (t1.rnk = 1)
2.	→ Window Aggregate
3.	→ Sort
4.	→ Aggregate
5.	→ Hash Inner Join Hash Cond: (p.product_id = pc.category_id)
6.	→ Hash Inner Join Hash Cond: (oi.product_id = p.product_id)
7.	→ Hash Inner Join Hash Cond: (o.customer_id = c.customer_id)
8.	→ Hash Inner Join Hash Cond: (oi.order_id = o.order_id)
9.	→ Seq Scan on order_items as oi
10.	→ Hash
11.	→ Seq Scan on orders as o
12.	→ Hash
13.	→ Seq Scan on customers as c
14.	→ Hash
15.	→ Seq Scan on products as p
16.	→ Hash
17.	→ Seq Scan on category as pc

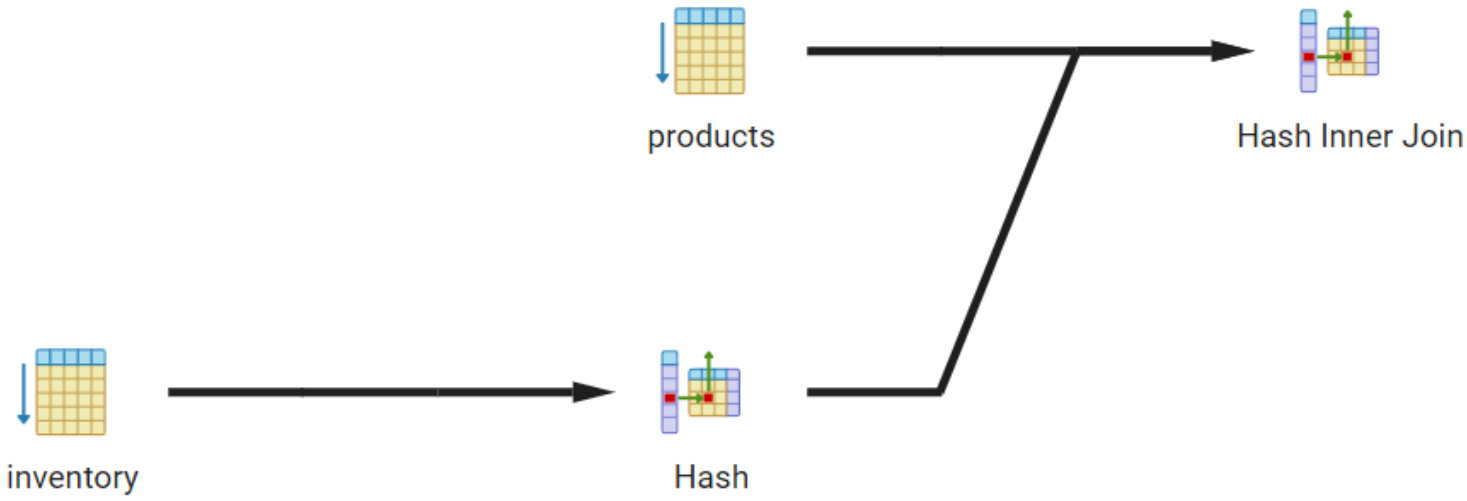
```sql
/*
7. Customer Lifetime Value (CLTV)
Calculate the total value of orders placed by each customer
over their lifetime.
Challenge: Rank customers based on their CLTV.
*/

EXPLAIN ANALYZE
SELECT c.customer_id AS customer_id,
CONCAT(c.first_name,' ',c.last_name) AS name,
ROUND(SUM(oi.total_order_value::numeric),2) as order_value,
DENSE_RANK() OVER(ORDER BY SUM(oi.total_order_value) DESC) as rnk
FROM customers c
JOIN orders o on o.customer_id = c.customer_id
JOIN order_items oi ON oi.order_id = o.order_id
GROUP BY 1;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Window Aggregate
2.	→ Sort
3.	→ Aggregate
4.	→ Hash Inner Join **Hash Cond**: (o.customer_id = c.customer_id)
5.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o.order_id)
6.	→ Seq Scan on order_items as oi
7.	→ Hash
8.	→ Seq Scan on orders as o
9.	→ Hash
10.	→ Seq Scan on customers as c

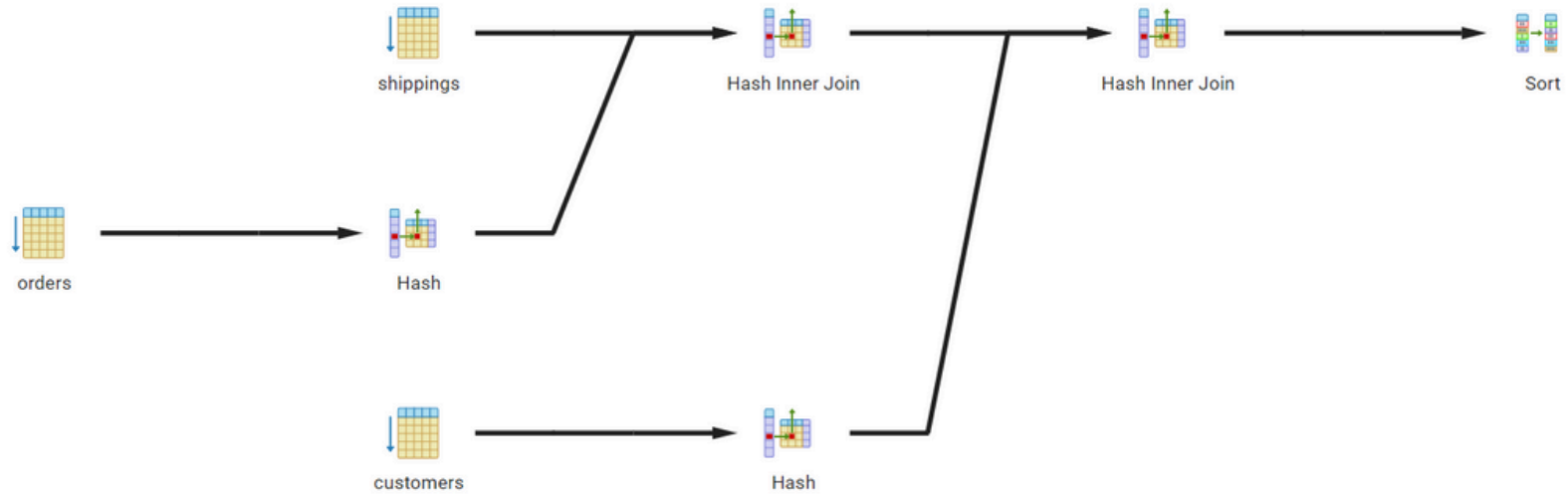
```sql
/*
8. Inventory Stock Alerts
Query products with stock levels below a certain threshold
(e.g., less than 10 units).
Challenge: Include last restock date and warehouse information.
*/

EXPLAIN ANALYZE
SELECT p.product_name,p.product_id,
i.inventory_id,i.stock,i.warehouse_id,
i.last_stock_date
FROM inventory i
JOIN products p ON p.product_id = i.product_id
WHERE stock<10;
```



Graphical | Analysis | Statistics

	#	Node
	1.	→ Hash Inner Join **Hash Cond**: (p.product_id = i.product_id)
	2.	→ Seq Scan on products as p
	3.	→ Hash
	4.	→ Seq Scan on inventory as i **Filter**: (stock < 10)

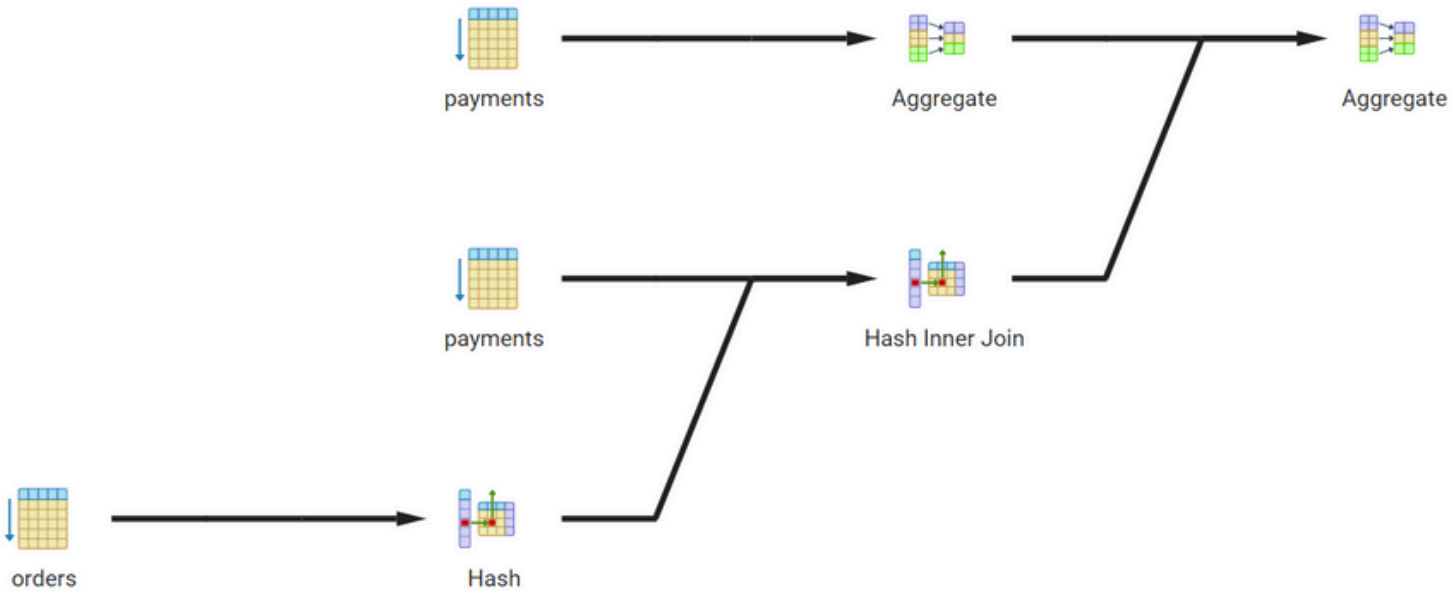
```sql
/*
9. Shipping Delays
Identify orders where the shipping date is later than 3 days
after the order date.
Challenge: Include customer, order details, and delivery provider.
*/

EXPLAIN ANALYZE
SELECT c.customer_id,
CONCAT(c.first_name,' ',c.last_name) as customer_name,c.state,
o.order_id,o.order_date,
s.shipping_date,s.shipping_providers,
s.shipping_date-o.order_date AS delivery_days
FROM orders o
JOIN shippings s ON o.order_id = s.order_id
JOIN customers c ON o.customer_id = c.customer_id
WHERE s.shipping_date-o.order_date>3
ORDER BY o.order_date;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Sort
2.	→ Hash Inner Join **Hash Cond**: (o.customer_id = c.customer_id)
3.	→ Hash Inner Join **Join Filter**: ((s.shipping_date - o.order_date) > 3) **Hash Cond**: (s.order_id = o.order_id)
4.	→ Seq Scan on shippings as s
5.	→ Hash
6.	→ Seq Scan on orders as o
7.	→ Hash
8.	→ Seq Scan on customers as c

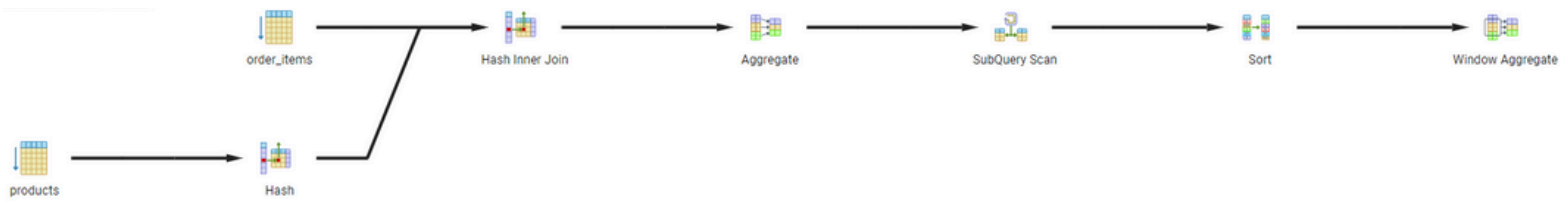
```sql
/*
10. Payment Success Rate
Calculate the percentage of successful payments across all orders.
Challenge: Include breakdowns by payment status
(e.g., failed, pending).
*/

EXPLAIN ANALYZE
SELECT
    p.payment_status,
    COUNT(*) as total_cnt,
    ROUND(COUNT(*)::numeric/
    (SELECT COUNT(*) FROM payments)::numeric * 100,2) AS pct
FROM orders as o
JOIN
payments as p
ON o.order_id = p.order_id
GROUP BY 1;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Aggregate
2.	→ Aggregate
3.	→ Seq Scan on payments as payments
4.	→ Hash Inner Join **Hash Cond**: (p.order_id = o.order_id)
5.	→ Seq Scan on payments as p
6.	→ Hash
7.	→ Seq Scan on orders as o

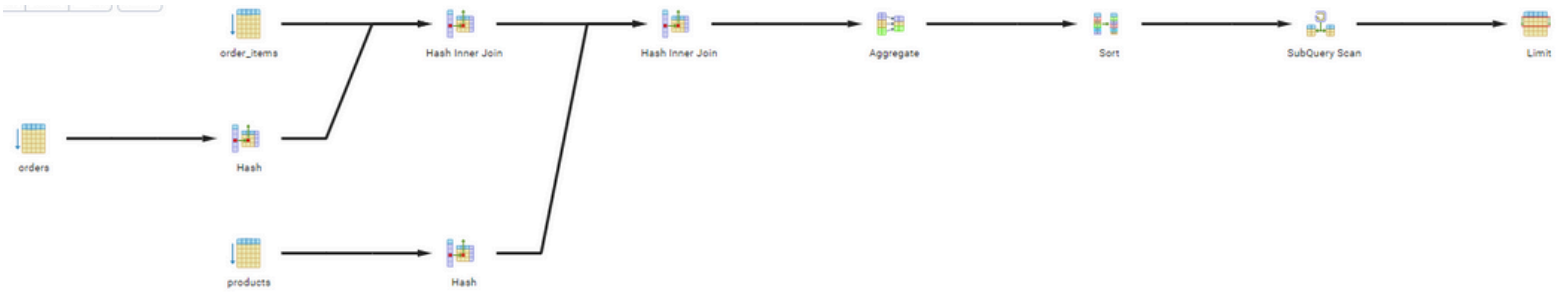
```sql
/*
11. Product Profit Margin
Calculate the profit margin for each product
(difference between price and cost of goods sold).
Challenge: Rank products by their profit margin, showing highest to lowest
*/

EXPLAIN ANALYZE
SELECT
    product_id,
    product_name,
    ROUND(profit_margin::NUMERIC,4) AS profit_margin,
    DENSE_RANK() OVER( ORDER BY profit_margin DESC) as product_ranking
FROM
(SELECT
    p.product_id,
    p.product_name,
    -- SUM(total_order_value - (p.cogs * oi.quantity)) as profit,
    SUM(total_order_value - (p.cogs * oi.quantity))/
        sum(total_order_value) * 100 as profit_margin
FROM order_items as oi
JOIN
products as p
ON oi.product_id = p.product_id
GROUP BY 1, 2
) as t1
```



Graphical | Analysis | Statistics

#	Node
1.	→ Window Aggregate
2.	→ Sort
3.	→ Subquery Scan
4.	→ Aggregate
5.	→ Hash Inner Join **Hash Cond**: (oi.product_id = p.product_id)
6.	→ Seq Scan on order_items as oi
7.	→ Hash
8.	→ Seq Scan on products as p

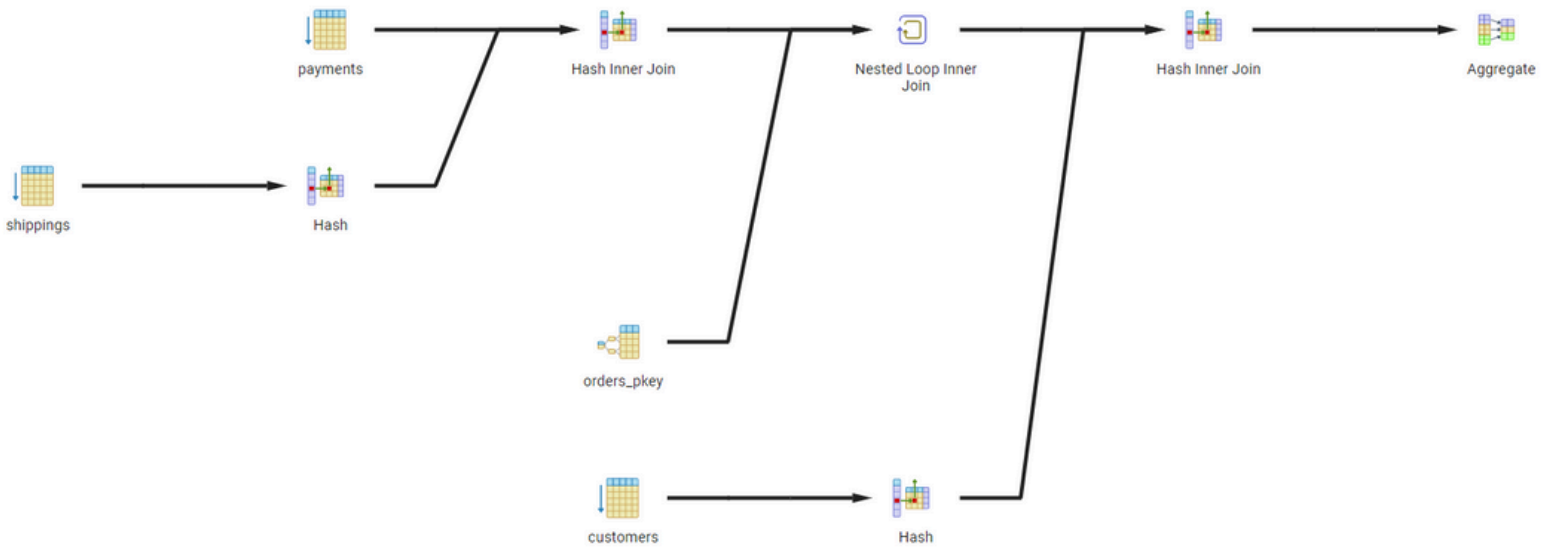
```sql
/*
12. Most Returned Products
Query the top 10 products by the number of returns.
Challenge: Display the return rate as a percentage of total units sold for each product
*/


SELECT *,
(no_items_returned*100/unit_ordered) as return_pct
FROM
(SELECT
p.product_id,p.product_name,
SUM(oi.quantity) AS unit_ordered,
COUNT(CASE WHEN o.order_status='Returned' THEN 1 END) AS times_returned,
SUM(CASE WHEN o.order_status='Returned' THEN oi.quantity END) AS no_items_returned
FROM orders o
JOIN order_items oi ON oi.order_id = o.order_id
JOIN products p ON p.product_id = oi.product_id
GROUP BY 1,2
ORDER BY 4 DESC) as t1
LIMIT 10;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Limit
2.	→ Subquery Scan
3.	→ Sort
4.	→ Aggregate
5.	→ Hash Inner Join **Hash Cond**: (oi.product_id = p.product_id)
6.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o.order_id)
7.	→ Seq Scan on order_items as oi
8.	→ Hash
9.	→ Seq Scan on orders as o
10.	→ Hash
11.	→ Seq Scan on products as p

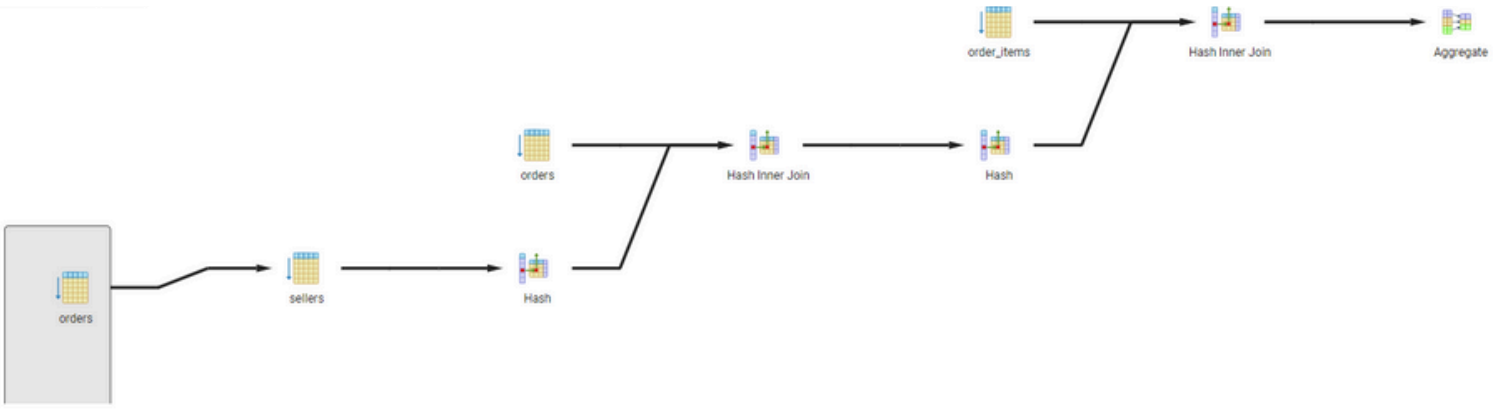
```sql
/*
13. Orders Pending Shipment
Find orders that have been paid but are still pending shipment.
Challenge: Include order details, payment date, and customer information.
*/

EXPLAIN ANALYZE
SELECT DISTINCT o.order_id,o.order_date,
c.customer_id,
CONCAT(c.first_name,' ',c.last_name) AS full_name,
p.payment_date,s.delivery_status
FROM shippings s
JOIN orders o ON o.order_id = s.order_id
JOIN payments p ON p.order_id = s.order_id
JOIN customers c ON c.customer_id = o.customer_id
WHERE s.delivery_status = 'Shipped'
AND p.payment_status = 'Payment Successed';
```



Graphical | Analysis | Statistics

#	Node
1.	→ Aggregate
2.	→ Hash Inner Join **Hash Cond**: (o.customer_id = c.customer_id)
3.	→ Nested Loop Inner Join **Join Filter**: (o.order_id = s.order_id)
4.	→ Hash Inner Join **Hash Cond**: (p.order_id = s.order_id)
5.	→ Seq Scan on payments as p **Filter**: ((payment_status)::text = 'Payment Successed'::text)
6.	→ Hash
7.	→ Seq Scan on shippings as s **Filter**: ((delivery_status)::text = 'Shipped'::text)
8.	→ Index Scan using orders_pkey on orders as o **Index Cond**: (order_id = p.order_id)
9.	→ Hash
10.	→ Seq Scan on customers as c

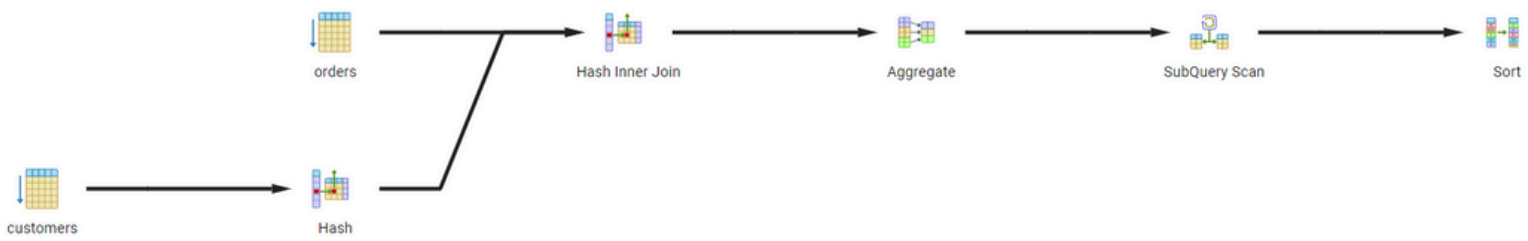
```sql
/*
14. Inactive Sellers
Identify sellers who haven't made any sales in the last 6 months.
Challenge: Show the last sale date and total sales from those sellers.
*/

EXPLAIN ANALYZE
WITH cte1 AS -- as these sellers has not done any sale in last 6 month
(SELECT * FROM sellers
WHERE seller_id NOT IN (SELECT seller_id FROM orders
WHERE order_date >= CURRENT_DATE - INTERVAL '6 month')
)

SELECT
o.seller_id,
MAX(o.order_date) as last_sale_date,
ROUND(MAX(oi.total_order_value)::numeric,2) as last_sale_amount
FROM orders as o
JOIN cte1 ON cte1.seller_id = o.seller_id
JOIN order_items as oi ON o.order_id = oi.order_id
GROUP BY 1;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Aggregate
2.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o.order_id)
3.	→ Seq Scan on order_items as oi
4.	→ Hash
5.	→ Hash Inner Join **Hash Cond**: (o.seller_id = sellers.seller_id)
6.	→ Seq Scan on orders as o
7.	→ Hash
8.	→ Seq Scan on sellers as sellers **Filter**: (NOT (ANY (seller_id = (hashed SubPlan 1).col1)))
9.	→ Seq Scan on orders as orders **Filter**: (order_date >= (CURRENT_DATE - '6 mons'::interval))

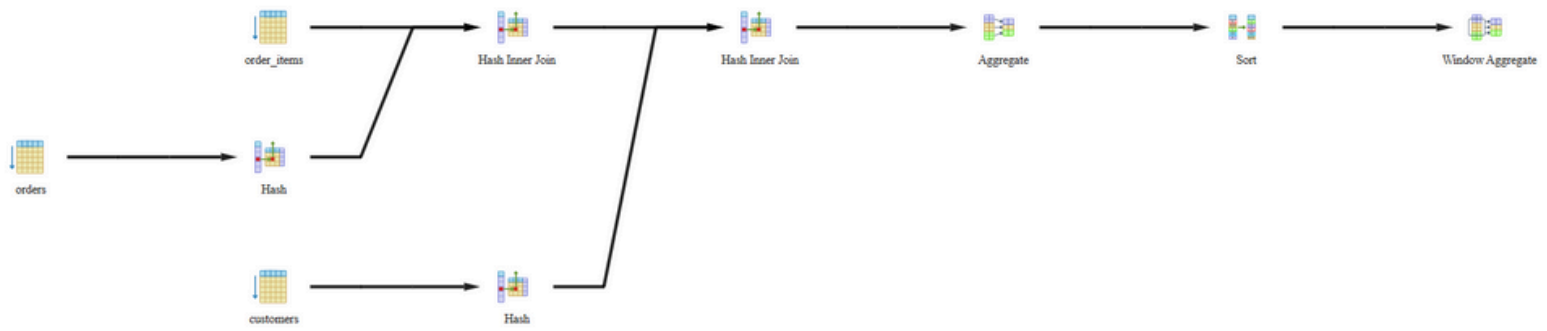
```sql
/*
15. IDENTITY customers into returning or new
if the customer has done more than 5 return categorize them as returning otherwise new
Challenge: List customers id, name, total orders, total returns
*/

EXPLAIN ANALYZE
SELECT customer_id,full_name,orders,order_returned,
CASE
    WHEN order_returned> 5 THEN 'Returning'
    ELSE 'New'
END AS identity
FROM
(SELECT
c.customer_id,CONCAT(c.first_name,' ',c.last_name) AS full_name,
COUNT(o.order_id) as orders,
SUM(CASE WHEN o.order_status = 'Returned' THEN 1 ELSE 0 END) as order_returned
FROM customers c
JOIN orders o ON o.customer_id = c.customer_id
GROUP BY 1) AS t1
ORDER BY 4 DESC;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Sort
2.	→ Subquery Scan
3.	→ Aggregate
4.	→ Hash Inner Join **Hash Cond**: (o.customer_id = c.customer_id)
5.	→ Seq Scan on orders as o
6.	→ Hash
7.	→ Seq Scan on customers as c

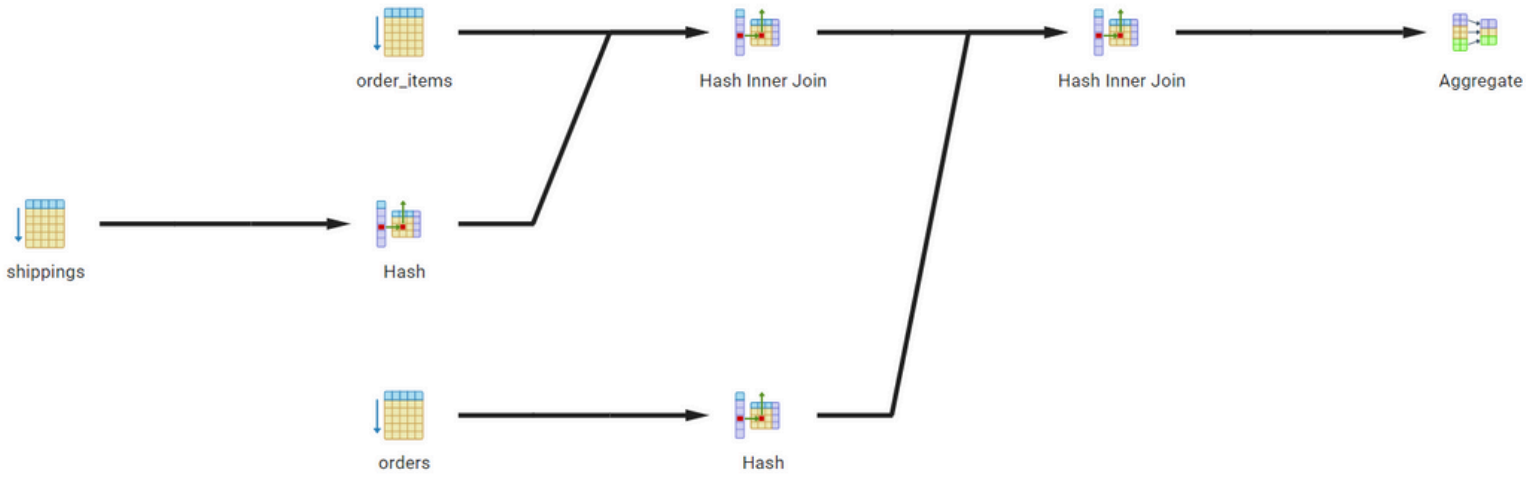
```sql
/*
16. Top 5 Customers by Orders in Each State
Identify the top 5 customers with the highest number of orders for each state.
Challenge: Include the number of orders and total sales for each customer.
*/

EXPLAIN ANALYZE
SELECT * FROM
(SELECT c.state,c.customer_id,
CONCAT(c.first_name,' ',c.last_name) AS full_name,
COUNT(o.order_id) as total_orders,
ROUND(SUM(oi.total_order_value)::NUMERIC,2) AS total_sales,
DENSE_RANK() OVER(PARTITION BY c.state ORDER BY COUNT(o.order_id) DESC) AS rnk
FROM orders o
JOIN order_items oi ON o.order_id = oi.order_id
JOIN customers c ON c.customer_id = o.customer_id
GROUP BY c.state,c.customer_id) AS t1
WHERE rnk<6;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Window Aggregate
2.	→ Sort
3.	→ Aggregate
4.	→ Hash Inner Join **Hash Cond**: (o.customer_id = c.customer_id)
5.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o.order_id)
6.	→ Seq Scan on order_items as oi
7.	→ Hash
8.	→ Seq Scan on orders as o
9.	→ Hash
10.	→ Seq Scan on customers as c

```sql
/*
17. Revenue by Shipping Provider
Calculate the total revenue handled by each shipping provider.
Challenge: Include the total number of orders handled
and the average delivery time for each provider.
*/

EXPLAIN ANALYZE
SELECT s.shipping_providers,
COUNT(o.order_id) AS order_handle,
ROUND(SUM(oi.total_order_value)::NUMERIC,2) AS total_revenue,
COALESCE(AVG(s.shipping_date-o.order_date), 0) as average_days
FROM shippings s
JOIN orders o ON o.order_id = s.order_id
JOIN order_items oi ON oi.order_id = s.order_id
GROUP BY s.shipping_providers;
```



Graphical | Analysis | Statistics

#	Node
1.	→ Aggregate
2.	→ Hash Inner Join **Hash Cond**: (s.order_id = o.order_id)
3.	→ Hash Inner Join **Hash Cond**: (oi.order_id = s.order_id)
4.	→ Seq Scan on order_items as oi
5.	→ Hash
6.	→ Seq Scan on shippings as s
7.	→ Hash
8.	→ Seq Scan on orders as o

ADVANCE QURIES

- Nested subqueries, window functions, CTEs, and performance optimization

1. Top Performing Sellers

Find the top 5 sellers based on total sales value.
Challenge: Include both successful and failed orders and display their percentage of successful orders.

2. Top 10 product with highest decreasing revenue ratio compared to year(2022) and year(2023)

Challenge: Return product_id, product_name, category_name, 2022 revenue and 2023 revenue decrease ratio at end Round the result

Note: Decrease ratio = cr-ls/ls* 100 (cs = current_year ls=last_year)

3. Store Procedure

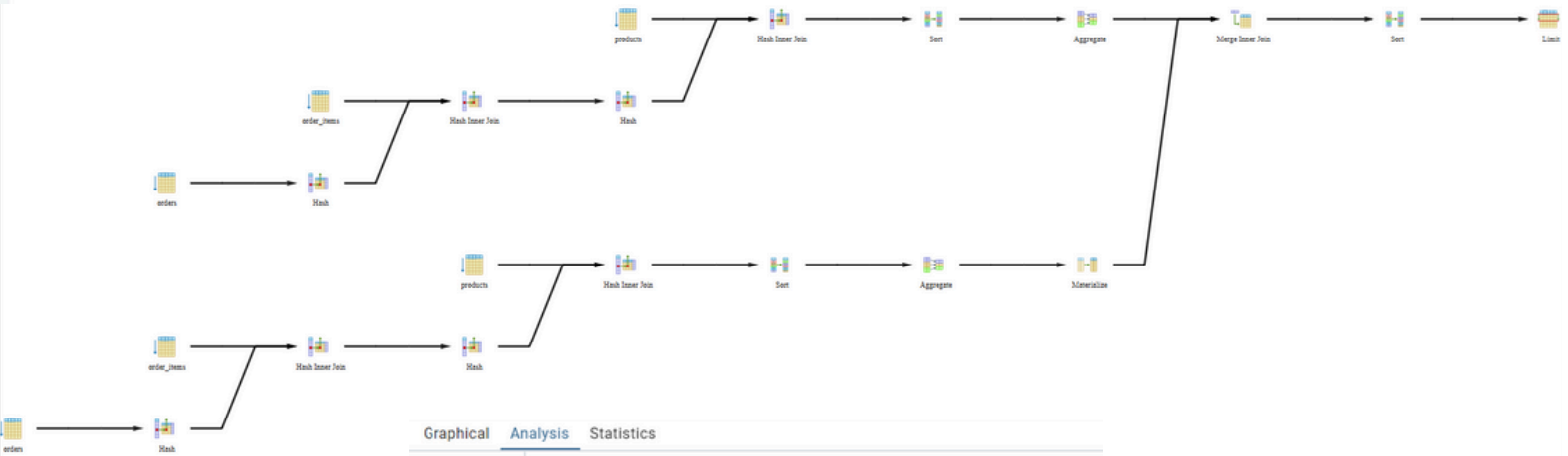
Create a function as soon as the product is sold the the same quantity should reduced from inventory table
after adding any sales records it should update the stock in the inventory table based on the product and qty purchased

```sql
EXPLAIN ANALYZE
WITH seller_sales AS(
SELECT s.seller_id,s.seller_name,ROUND(SUM(total_order_value::NUMERIC),2) AS sales
FROM orders o
JOIN sellers s ON s.seller_id = o.seller_id
JOIN order_items oi ON oi.order_id = o.order_id
GROUP BY s.seller_id
ORDER BY 3 DESC
LIMIT 5),

order_report AS(
SELECT o.seller_id,ts.seller_name,o.order_status,COUNT(*) as total_orders
FROM orders o
JOIN seller_sales ts ON ts.seller_id = o.seller_id
WHERE o.order_status NOT IN ('Inprogress','Returned')
GROUP BY 1,2,3
ORDER BY 1)

SELECT
    seller_id,
    seller_name,
    SUM(CASE WHEN order_status = 'Completed' THEN total_orders ELSE 0 END) as Completed_orders,
    SUM(CASE WHEN order_status = 'Cancelled' THEN total_orders ELSE 0 END) as Cancelled_orders,
    SUM(total_orders) as total_orders,
    ROUND(SUM(CASE WHEN order_status = 'Completed' THEN total_orders ELSE 0 END)::numeric/
    SUM(total_orders)::numeric * 100,2) as successful_orders_percentage

FROM order_report
GROUP BY 1, 2
```

Graphical | Analysis | Statistics

#	Node
1.	→ Aggregate
2.	→ Sort
3.	→ Aggregate
4.	→ Hash Inner Join **Hash Cond**: (o.seller_id = ts.seller_id)
5.	→ Seq Scan on orders as o **Filter**: ((order_status)::text <> ALL ('{Inprogress,Returned}'::text[]))
6.	→ Hash
7.	→ Subquery Scan
8.	→ Limit
9.	→ Sort
10.	→ Aggregate
11.	→ Hash Inner Join **Hash Cond**: (o_1.seller_id = s.seller_id)
12.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o_1.order_id)
13.	→ Seq Scan on order_items as oi
14.	→ Hash
15.	→ Seq Scan on orders as o_1
16.	→ Hash
17.	→ Seq Scan on sellers as s

```sql
EXPLAIN ANALYZE
WITH last_year_sale AS(
SELECT p.product_id,p.product_name,SUM(oi.total_order_value) AS revenue
FROM orders as o
JOIN order_items as oi ON oi.order_id = o.order_id
JOIN products as p ON p.product_id = oi.product_id
WHERE EXTRACT(YEAR FROM o.order_date) = 2022
GROUP BY 1, 2),

current_year_sale AS(
SELECT p.product_id,p.product_name,SUM(oi.total_order_value) as revenue
FROM orders as o
JOIN order_items as oi ON oi.order_id = o.order_id
JOIN products as p ON p.product_id = oi.product_id
WHERE EXTRACT(YEAR FROM o.order_date) = 2023
GROUP BY 1, 2)

SELECT
    cs.product_id,
    ROUND(ls.revenue::NUMERIC,2) as last_year_revenue,
    ROUND(cs.revenue::NUMERIC,2) as current_year_revenue,
    ROUND((ls.revenue - cs.revenue)::NUMERIC,2) as rev_diff,
    ROUND((cs.revenue - ls.revenue)::numeric/ls.revenue::numeric * 100, 2) as reveneue_dec_ratio
FROM last_year_sale as ls
JOIN current_year_sale as cs
ON ls.product_id = cs.product_id
WHERE ls.revenue > cs.revenue
ORDER BY 5 DESC
LIMIT 10
```



Graphical | Analysis | Statistics

#	Node
1.	→ Limit
2.	→ Sort
3.	→ Merge Inner Join **Join Filter**: ((sum(oi.total_order_value)) > (sum(oi_1.total_order_value)))
4.	→ Aggregate
5.	→ Sort
6.	→ Hash Inner Join **Hash Cond**: (p.product_id = oi.product_id)
7.	→ Seq Scan on products as p
8.	→ Hash
9.	→ Hash Inner Join **Hash Cond**: (oi.order_id = o.order_id)
13.	→ Materialize
14.	→ Aggregate
15.	→ Sort
16.	→ Hash Inner Join **Hash Cond**: (p_1.product_id = oi_1.product_id)
17.	→ Seq Scan on products as p_1
18.	→ Hash
19.	→ Hash Inner Join **Hash Cond**: (oi_1.order_id = o_1.order_id)
20.	→ Seq Scan on order_items as oi_1
21.	→ Hash

```sql
CREATE OR REPLACE PROCEDURE add_sales(
p_order_id INT,
p_customer_id INT,
p_seller_id INT,
p_order_item_id INT,
p_product_id INT,
p_quantity INT
)
LANGUAGE plpgsql
AS $$

DECLARE -- DECLARING VARIABLES
v_count INT;
v_price FLOAT;
v_product VARCHAR(50);
BEGIN

-- Fetching product name and price based p id entered
    SELECT price, product_name INTO v_price, v_product
    FROM products WHERE product_id = p_product_id;

-- checking stock and product availability in inventory
    SELECT COUNT(*) INTO v_count
    FROM inventory WHERE
        product_id = p_product_id AND stock >= p_quantity;

    IF v_count > 0 THEN -- add into orders and order_items table
        INSERT INTO orders(order_id, order_date, customer_id, seller_id) -- update inventory
        VALUES (p_order_id, CURRENT_DATE, p_customer_id, p_seller_id);

        -- adding into order list
        INSERT INTO order_items(order_item_id, order_id, product_id, quantity, price_per_unit, total_order_value)
        VALUES
        (p_order_item_id, p_order_id, p_product_id, p_quantity, v_price, v_price*p_quantity);

        --updating inventory
        UPDATE inventory
        SET stock = stock - p_quantity
        WHERE product_id = p_product_id;

        RAISE NOTICE 'Thank you product: % sale has been added also inventory stock updates',v_product;

    ELSE
        RAISE NOTICE 'Thank you for for your info the product: % is not available', v_product;

    END IF;

END;
$$
```

THANK YOU

GitHub Link