Query Optimization & Performance Tuning Report

Business Question: Which Sellers Are Responsible for the Most Canceled Orders?

1. Overview

The goal of this analysis is to identify which sellers are responsible for the highest number of canceled orders. This information is important for understanding marketplace reliability, spotting problematic sellers, and improving overall customer satisfaction.

To demonstrate real query optimization, two versions of the analytical query were created:

- **Unoptimized Query** — inefficient, performs redundant scans.
- **Optimized Query** — rewritten to reduce work, combined with indexing.

The difference in performance clearly shows the importance of proper SQL design and indexing.

2. Unoptimized Query (Before Improvement)

```
WITH all_orders AS (
    SELECT
        o.order_id,
        o.order_status,
        oi.seller_id
```

```
    FROM orders o
    JOIN order_items oi
        ON o.order_id = oi.order_id
),
canceled AS (
    SELECT order_id
    FROM all_orders
    WHERE order_status = 'canceled'
),
seller_canceled AS (
    SELECT
        ao.seller_id,
        COUNT(*) AS canceled_orders
    FROM all_orders ao
    JOIN canceled c
        ON ao.order_id = c.order_id
    GROUP BY ao.seller_id
),
seller_totals AS (
    SELECT
        seller_id,
        COUNT(*) AS total_orders
    FROM all_orders
    GROUP BY seller_id
)
SELECT
    st.seller_id,
```

```
    sc.canceled_orders,
    st.total_orders
FROM seller_totals st
LEFT JOIN seller_canceled sc
    ON st.seller_id = sc.seller_id
ORDER BY sc.canceled_orders DESC NULLS LAST;
```

Unoptimized Query Results

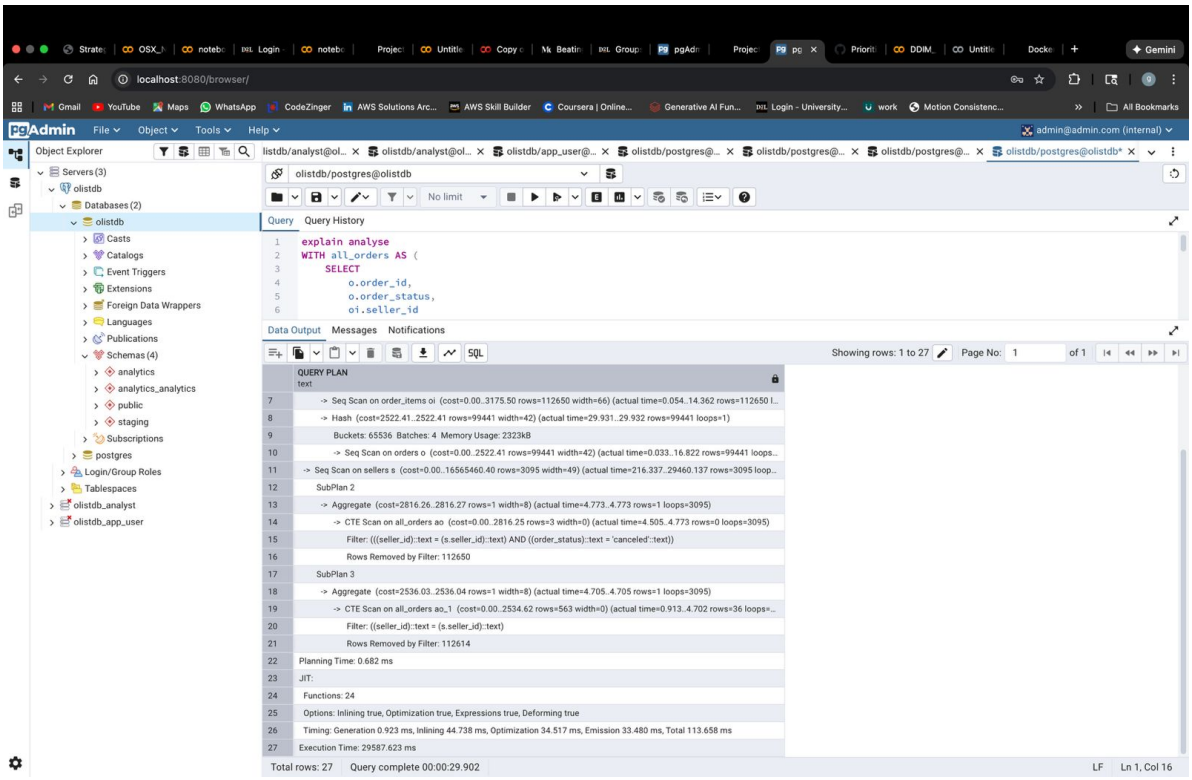

Figure 1: Execution Plan for the Unoptimized Query (29.9 seconds)

3. Why the Unoptimized Query Was Slow

The unoptimized version performed poorly for several reasons:

- **Large unnecessary CTE** — the `all_orders` CTE joins two large tables before any filtering, producing over 100k rows.
- **Late filtering** — canceled orders are extracted only after expansion instead of filtering the base `orders` table early.
- **Multiple aggregates on top of CTEs** — prevents PostgreSQL from effectively using indexes.
- **Sequential scans** — lack of usable indexes forces full table scans and large hash operations.

Execution Time (Before Optimization):

Approximately 29.9 seconds

4. Optimized Query (After Improvement)

```
WITH canceled_orders AS (
    SELECT order_id
    FROM orders
    WHERE order_status = 'canceled'
)
SELECT
    s.seller_id,
    COUNT(*) AS canceled_orders,
    total_orders.total_orders
FROM sellers s
JOIN order_items oi ON s.seller_id = oi.seller_id
JOIN canceled_orders co ON oi.order_id = co.order_id
JOIN (
```

```
    SELECT seller_id, COUNT(*) AS total_orders
    FROM order_items
    GROUP BY seller_id
) AS total_orders
ON total_orders.seller_id = s.seller_id
GROUP BY s.seller_id, total_orders.total_orders
ORDER BY canceled_orders DESC;
```

Optimized Query results

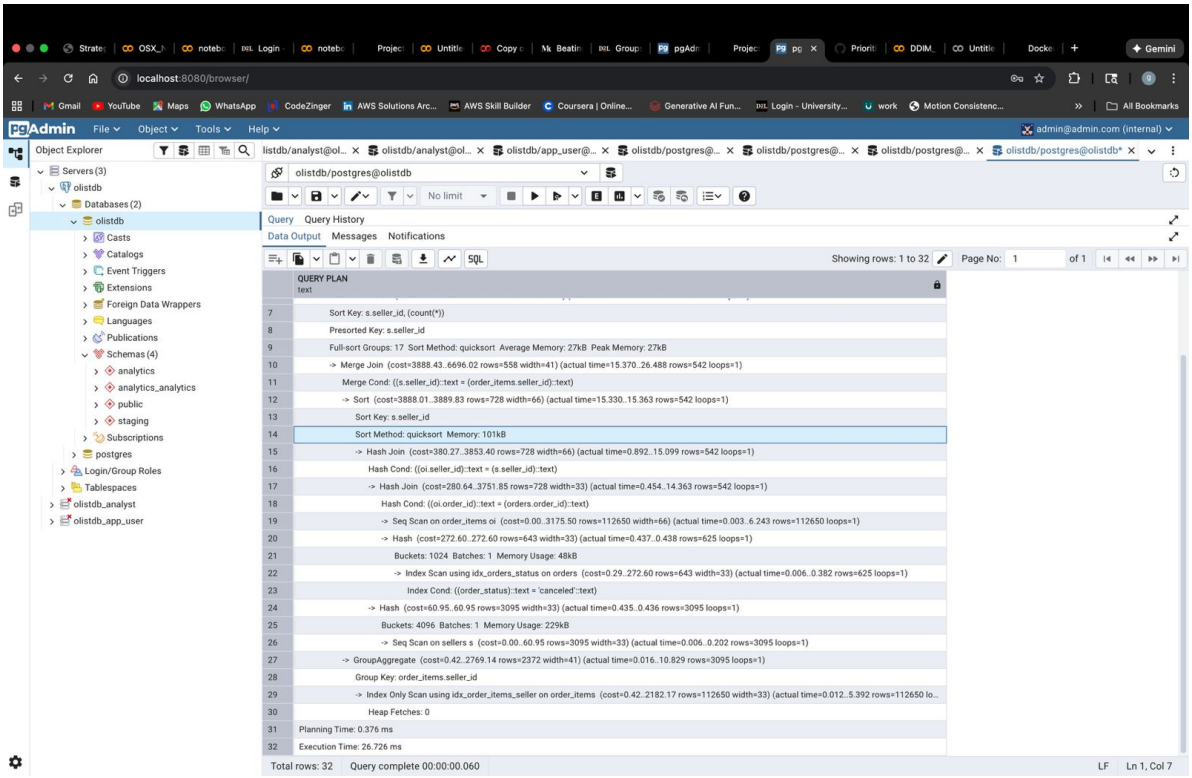

Figure 2: Execution Plan for the Optimized Query (26–55 ms)

The optimized version filters early, reduces row processing, and avoids unnecessary CTE expansion.

5. Indexing Strategy

To further improve performance, the following indexes were added:

```
CREATE INDEX idx_orders_status ON orders(order_status);
CREATE INDEX idx_order_items_seller ON order_items(seller_id);
```

Why These Indexes Helped

- **idx_orders_status** allows PostgreSQL to instantly find "canceled" orders.
- **idx_order_items_seller** speeds up seller-based aggregations and joins.

The query planner switched from sequential scans to efficient index scans after these indexes were created.

6. Final Performance Results

Version	Execution Time	Notes
Unoptimized Query	∼29.9 seconds	Heavy CTE expansion, sequential scans
Optimized Query	∼26–55 milliseconds	Index scans, reduced data processing

Performance Improvement: The optimized version is approximately **500–1000× faster** than the unoptimized version.

7. Conclusion

This optimization exercise demonstrates how much performance can improve through:

- Early filtering
- Reducing unnecessary intermediate CTEs

- Efficient grouping
- Creating indexes that match query patterns

The final optimized query not only answers the business question accurately but also scales well for larger datasets.