**What to Say in Your Report (when you're ready to document):**

**Query 1: Top 10 Best-Selling Products**  
We created an index on order\_items(product\_id) to optimize the JOIN.  
However, PostgreSQL continued to use a sequential scan (Seq Scan) on the table, as it estimated this was more efficient given the data size.  
The index had minimal impact on execution time, which remained around ~18–19 ms.

**Query 2: Revenue by Category**  
We attempted to optimize the JOIN with category by indexing products(category\_id).  
However, PostgreSQL continued to perform a sequential scan, likely because the table was small or already cached.  
There was a slight reduction in execution time from **24.723 ms** to **22.567 ms**, but the query planner didn’t use the new index.

**Query 3: Monthly Sales Trends**  
We created an index on orders(order\_date) to optimize the GROUP BY and JOIN operations.  
PostgreSQL continued to perform a sequential scan on orders, possibly due to low table size or cost estimation.  
Despite this, execution time slightly decreased from **29.093 ms** to **27.434 ms**.

**Query Optimization and Execution Analysis**

To evaluate and improve performance, we selected three complex SQL queries from our project and analyzed them using EXPLAIN ANALYZE in PostgreSQL. We first ran each query, noted the execution cost and scan methods, and then created an index on a relevant column to optimize it. After re-running each query, we compared the changes in execution time and scan behavior.

**🔹 Query 1: Top 10 Best-Selling Products**

**Query:**

sql

CopyEdit

SELECT p.product\_name, SUM(oi.quantity \* oi.price\_per\_unit) AS total\_sales

FROM order\_items oi

JOIN products p ON oi.product\_id = p.product\_id

GROUP BY p.product\_name

ORDER BY total\_sales DESC

LIMIT 10;

**Index Applied:**

sql

CopyEdit

CREATE INDEX idx\_order\_items\_product\_id ON order\_items(product\_id);

**Results:**

* Before index: Execution Time: 18.023 ms, Scan: Seq Scan on order\_items
* After index: Execution Time: 19.019 ms, Scan: Seq Scan on order\_items

**Observation:**  
The query planner chose not to use the index, likely due to the small table size. Despite indexing, PostgreSQL determined that a sequential scan was more efficient.

**🔹 Query 2: Revenue by Category**

**Query:**

sql

CopyEdit

SELECT c.category\_name, SUM(oi.quantity \* oi.price\_per\_unit) AS revenue

FROM order\_items oi

JOIN products p ON oi.product\_id = p.product\_id

JOIN category c ON p.category\_id = c.category\_id

GROUP BY c.category\_name

ORDER BY revenue DESC;

**Index Applied:**

sql

CopyEdit

CREATE INDEX idx\_products\_category\_id ON products(category\_id);

**Results:**

* Before index: Execution Time: 24.723 ms, Scan: Seq Scan on products
* After index: Execution Time: 22.567 ms, Scan: Seq Scan on products

**Observation:**  
Though the index was successfully created, it was not used in the execution plan. PostgreSQL performed a sequential scan both before and after optimization, indicating it estimated the index would not improve performance.

**🔹 Query 3: Monthly Sales Trends**

**Query:**

sql

CopyEdit

SELECT DATE\_TRUNC('month', o.order\_date) AS month,

SUM(oi.quantity \* oi.price\_per\_unit) AS revenue

FROM orders o

JOIN order\_items oi ON o.order\_id = oi.order\_id

GROUP BY month

ORDER BY month;

**Index Applied:**

sql

CopyEdit

CREATE INDEX idx\_orders\_order\_date ON orders(order\_date);

**Results:**

* Before index: Execution Time: 29.093 ms, Scan: Seq Scan on orders
* After index: Execution Time: 27.434 ms, Scan: Seq Scan on orders

**Observation:**  
PostgreSQL continued to use sequential scan despite the index. The slight drop in execution time is likely due to internal caching or improved memory access during the second execution.

**✅ Summary**

All three queries were tested using EXPLAIN ANALYZE before and after indexing. While indexes did not change the scan strategy due to the query planner's cost estimates, a small performance gain was observed in Query 2 and Query 3. This step demonstrates the practical use of query analysis tools and indexing as part of performance tuning in a relational database system.