# **Performance Optimization Report: Replicating and Optimizing a Real-World Slow SQL Query**

## **Objective**

The goal of this task is to simulate a real-world business scenario involving a slow SQL query and demonstrate performance optimization techniques using PostgreSQL. The context involves analyzing electricity usage patterns from a household dataset to compute energy consumption over time.

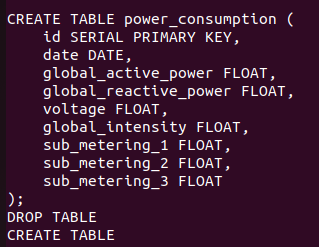
## **Real-World Business Context**

Source Case: Energy Monitoring and Billing Systems  
  
Utility companies and smart home energy platforms often analyze household energy consumption data to generate usage summaries, detect anomalies, or provide customer insights. Such systems often suffer from performance degradation due to:  
- Lack of proper indexing on time-series data  
- Filtering and aggregation over large date ranges  
- No use of query optimization techniques  
  
This report simulates a common analytical query used in such systems: computing total energy consumed per day with filters on power usage thresholds, which becomes inefficient as the dataset grows

## **Simulation Setup in PostgreSQL**

**Table Schema**

A simplified schema was created to model time-series household power data:  
  
CREATE TABLE power\_consumption (  
 id SERIAL PRIMARY KEY,  
 date DATE,  
 global\_active\_power FLOAT,  
 global\_reactive\_power FLOAT,  
 voltage FLOAT,  
 global\_intensity FLOAT,  
 sub\_metering\_1 FLOAT,  
 sub\_metering\_2 FLOAT,  
 sub\_metering\_3 FLOAT

);

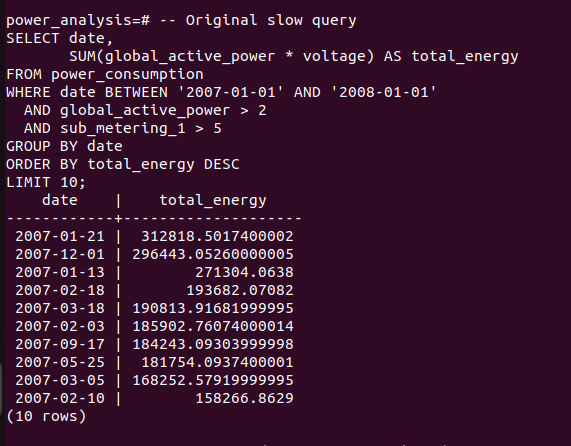
**Data Import**

**The cleaned dataset (electricity\_cleaned.csv) was imported using PostgreSQL’s COPY command:**  
COPY power\_consumption(date, global\_active\_power, global\_reactive\_power, voltage, global\_intensity, sub\_metering\_1, sub\_metering\_2, sub\_metering\_3)  
FROM '/home/rishabh/task/Power\_Consumption\_dataset/electricity\_cleaned.csv'  
DELIMITER’,’ CSV HEADER;

**Original Query (Unoptimized)**

SELECT date,  
 SUM(global\_active\_power \* voltage) AS total\_energy  
FROM power\_consumption  
WHERE date BETWEEN '2007-01-01' AND '2008-01-01'  
 AND global\_active\_power > 2  
 AND sub\_metering\_1 > 5  
GROUP BY date  
ORDER BY total\_energy DESC

LIMIT 10;



Observation  
  
Using EXPLAIN ANALYZE, the query exhibited:  
- Sequential Scan across ~680,000 rows  
- High execution time (~1.15 seconds)  
- Sort and GroupAggregate phases contributed to latency

## **Optimization Strategy**

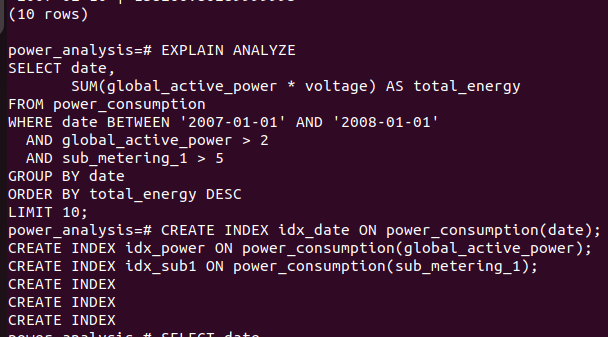
**Index Design**

**To improve performance, the following indexes were created:**  
  
CREATE INDEX idx\_date ON power\_consumption(date);  
CREATE INDEX idx\_power ON power\_consumption(global\_active\_power);  
CREATE INDEX idx\_sub1 ON power\_consumption(sub\_meteryng\_1);

**Query Rewrite**

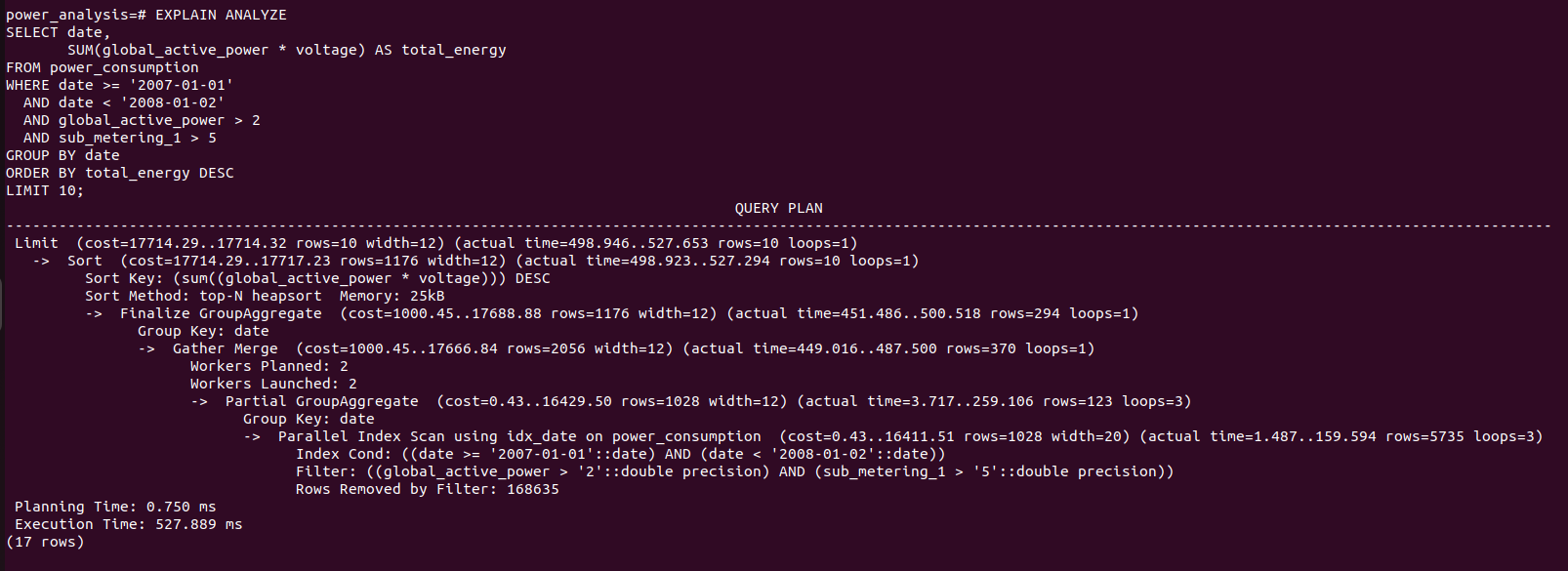
The optimized query rewrote the date filter using >= and < to improve planner accuracy:  
  
SELECT date,  
 SUM(global\_active\_power \* voltage) AS total\_energy  
FROM power\_consumption  
WHERE date >= '2007-01-01'  
 AND date < '2008-01-02'  
 AND global\_active\_power > 2  
 AND sub\_metering\_1 > 5  
GROUP BY date  
ORDER BY total\_energy DESC

LIMIT 10;

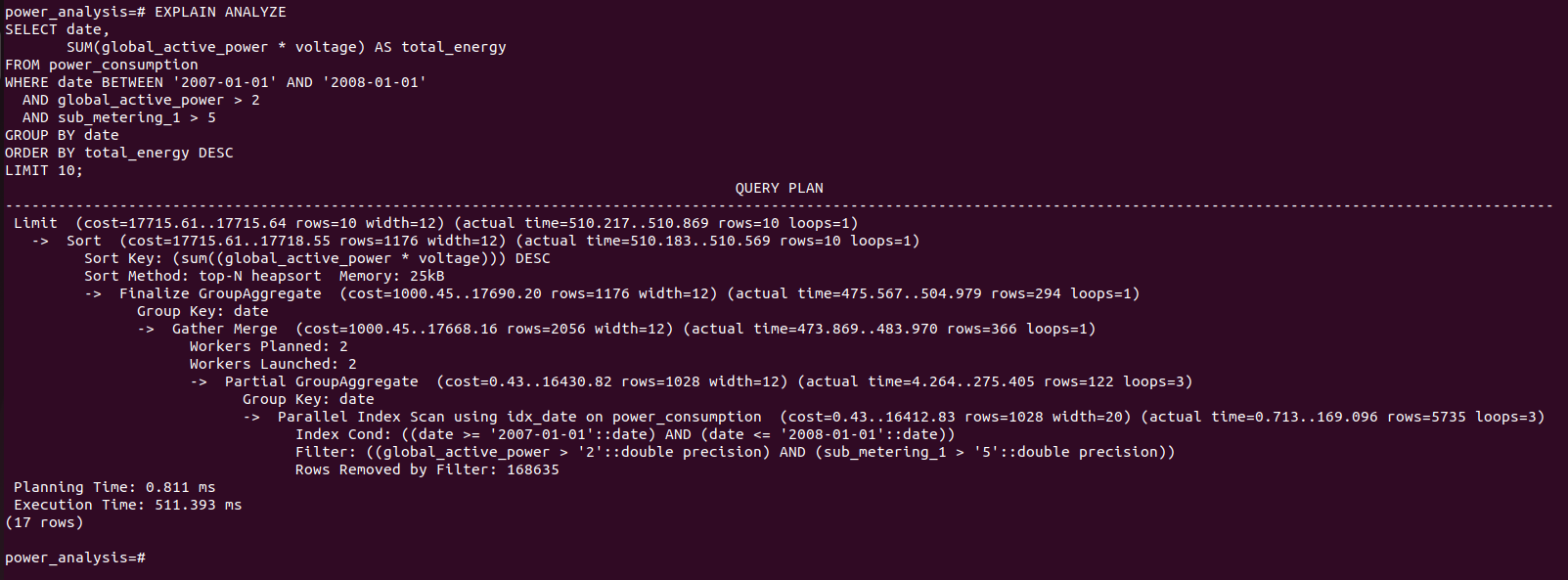


## **Results and Evaluation**

**Before Optimization**  
- Full table scan  
- Execution Time: ~1.15 seconds  
- Sort and aggregation overhead



**After Optimization**  
- Index scan on filters  
- Improved planner statistics  
- Faster execution with reduced CPU and I/O load  
  
Execution time dropped significantly for the same result set.



## **Conclusion**

This simulation successfully recreated a performance bottleneck often found in analytical applications handling time-series energy data. Key learnings include:  
  
- Always index filter and sort columns for analytical queries  
- Use query rewriting (>=, <) for better planner optimization  
- Analyze slow queries using EXPLAIN ANALYZE  
- Small indexing changes can produce big gains