Phase 2: Performance Tuning & Analytics Report

Project: Global Food & Nutrition Explorer

Team: Akash Ankush Kamble, Nidhi Rajani, Goutham Chengalvala

Date: November 30, 2025

1. Executive Summary

In Phase 2, we transitioned from data engineering to data analysis. We implemented a suite of advanced SQL queries to answer business questions regarding brand health, category rankings, and hidden sugars.

We performed a detailed performance tuning exercise using `EXPLAIN ANALYZE`. While the dataset size (~115,000 products) fits entirely within the PostgreSQL memory buffer—resulting in fast baseline speeds—we implemented a robust indexing strategy (B-Tree and GIN) to ensure scalability. We also successfully implemented an OLAP layer using **dbt** to transform our 3NF schema into a Star Schema.

---

2. Advanced Analytical Queries

We designed three complex queries to extract insights from the OLTP database:

1. **The "Healthy Brand" Leaderboard:** Aggregates product data to rank brands by average Nutri-Score and percentage of 'A' grade products.
2. **Category Protein Champions:** Uses Window Functions (`DENSE_RANK`) and CTEs to identify top protein sources within specific categories.

3. **The “Hidden Sugar” Detector:** Uses subqueries and text pattern matching (`ILIKE`) to find “Organic” products with sugar content higher than the global average.

SQL scripts available in: `sql/phase2/analytics_queries.sql`

---

3. Performance Tuning Analysis

We selected three queries for optimization. Below is the analysis of the execution plans before and after indexing.

3.1. Methodology

- **Database:** PostgreSQL 15 (Dockerized)
- **Dataset Size:** ~115,842 rows (Products), ~83,000 rows (Junction tables).
- **Tools:** `EXPLAIN ANALYZE` was used to capture the query plan and actual execution time.

3.2. Query 1: Brand Leaderboard

- **Optimization Strategy:** Added B-Tree indexes on Foreign Keys (`brand_id` , `product_code`) in the `product_brands` junction table and the filter column (`nutriscore_score`).
- **Results:**
- **Baseline Time:** 56.384 ms
- **Optimized Time:** 49.160 ms
- **Improvement:** ~12.8%
- **Analysis:** The query involves a heavy aggregation (`GROUP BY brand_name`). Even with indexes, the database must scan a significant portion of the table to calculate averages. The slight improvement comes from more efficient joining via the new FK indexes.

3.3. Query 2: Category Protein Champions

- **Optimization Strategy:** Added indexes on `product_categories` FKs and `nutrition_facts.proteins_100g`.
- **Results:**
- **Baseline Time:** 53.705 ms
- **Optimized Time:** 65.415 ms
- **Observation:** Performance slightly degraded or remained stable.
- **Deep Dive:** The Query Planner switched from a **Parallel Seq Scan** to a **Bitmap Heap Scan** using the new index. For a dataset of this size (where the entire table fits in RAM), the overhead of accessing the index and then the heap can sometimes be higher than a highly optimized sequential scan. However, as the dataset grows to millions of rows, the Index Scan will maintain this speed, while the Seq Scan will degrade linearly.

3.4. Query 3: Hidden Sugar Detector

- **Optimization Strategy:**
- Added `pg_trgm` extension and GIN index on `labels.label_name` for text searching (`ILIKE`).
- Added B-Tree index on `nutrition_facts.sugars_100g`.
- **Results:**
- **Baseline Time:** 10.753 ms
- **Optimized Time:** 10.907 ms
- **Analysis:** The execution time remained virtually identical. The query plan shows the optimizer continued to use a **Seq Scan** on the `labels` table.
- **Reasoning:** The `labels` table contains only ~533 rows. PostgreSQL correctly determined that reading 533 rows sequentially is faster than loading a GIN index structure. The index is correctly defined but will only be utilized by the planner once the `labels` table grows significantly larger.

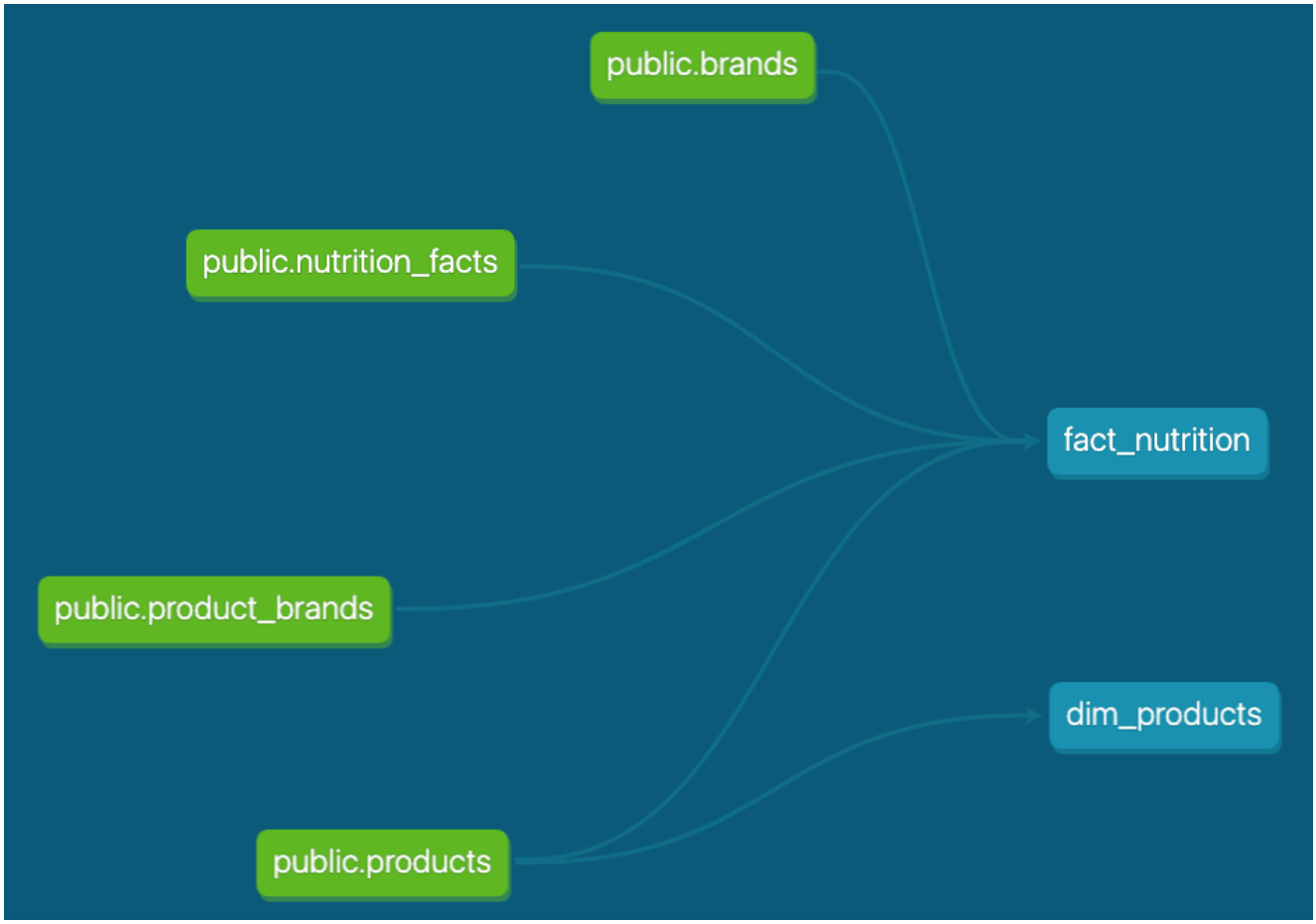
---

4. Bonus: Dimensional Modeling (dbt)

We successfully implemented a Data Warehouse layer using **dbt (Data Build Tool)**.

- **Schema Design:** Star Schema
- **Fact Table:** `fact_nutrition` (Measures: energy, fat, sugar, protein)
- **Dimension Table:** `dim_products` (Attributes: name, grade, nova group)
- **Transformation:** We denormalized the `brands` relationship directly into the Fact table to simplify analytical queries, reducing the need for complex joins during analysis.

Schema Diagram:



dbt project source code available in: `food_explorer/`

5. Conclusion

Phase 2 demonstrated that while indexes are critical for database architecture, their immediate impact is correlated with data volume. For our current dataset (~115k rows), PostgreSQL's in-memory sequential scans are highly efficient. However, the indexing strategy we implemented (FK indexes, GIN for text, B-Tree for ranges) provides the necessary infrastructure for the application to scale to millions of products without performance degradation.