RetailSQL: Relational Data Platform

Kevin Mota da Costa

January 30, 2026

Abstract

RetailSQL is a relational data platform designed to model and enforce core retail business processes, including sales transactions, product catalog, store operations, and inventory tracking. This document presents the architectural decisions, data modeling strategy, and verification evidence of the platform, treating RetailSQL as a production-ready data product rather than an analytical or exploratory project.

1 Product Scope and Objectives

RetailSQL is designed as a foundational transactional data layer for retail operations. Its primary objectives are:

- Enforce business rules at the database level,
- Provide a clean and normalized relational model,
- Guarantee referential and domain integrity,
- Serve as a reliable base for downstream analytics and reporting systems.

The platform intentionally avoids analytical abstractions (views, aggregates, materialized models) and focuses strictly on data correctness, consistency, and structural clarity.

2 Data Modeling Approach

The data model was developed using a layered methodology:

1. **Business Rules Specification** — formal definition of domain rules,
2. **Conceptual Model** — entities and relationships,
3. **Logical Model** — attributes, keys, and constraints,
4. **Physical Model** — PostgreSQL implementation.

This separation ensures that business semantics remain independent from implementation details.

3 Entity-Relationship Model

The conceptual structure of RetailSQL is expressed through an ER diagram, implemented using Mermaid syntax and maintained as code.

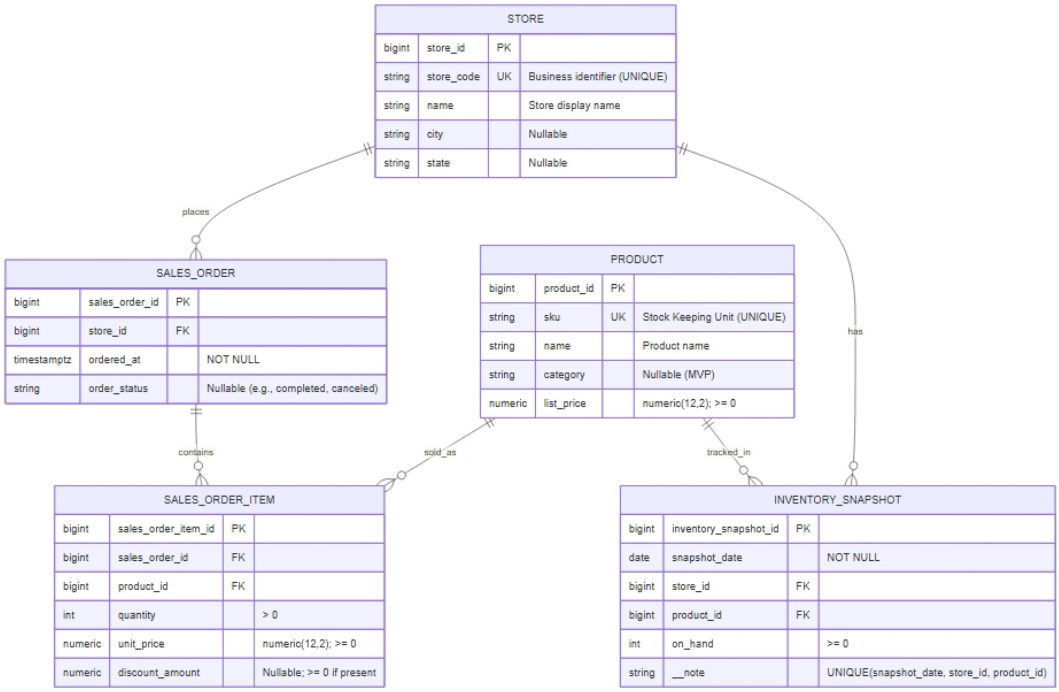

Figure 1: RetailSQL Entity-Relationship Diagram

The core entities are:

- **STORE** — physical retail locations,
- **PRODUCT** — sellable items,
- **SALES_ORDER** — transactional sales events,
- **SALES_ORDER_ITEM** — line-level sales details,
- **INVENTORY_SNAPSHOT** — point-in-time inventory states.

All many-to-many relationships are resolved through associative entities, and no redundant or derived attributes are stored.

4 Physical Implementation

The physical data model is implemented in PostgreSQL 16 and structured into clearly separated SQL layers:

- `schema.sql` — table definitions and primary keys,
- `constraints.sql` — business rules and referential integrity,
- `indexes.sql` — performance-oriented physical indexes,

- `seed.sql` — minimal deterministic seed data,
- `queries.sql` — verification and inspection queries.

The database runs inside a Docker container to ensure reproducibility and environment isolation.

5 Business Rule Enforcement

Business rules are enforced directly at the database level through constraints.

Examples include:

- Quantities must be strictly positive,
- Monetary values must be non-negative,
- A product cannot appear more than once in the same sales order,
- Inventory snapshots are unique per (date, store, product).

These rules are expressed via `CHECK`, `UNIQUE`, and `FOREIGN KEY` constraints, ensuring that invalid states cannot be persisted.

6 Verification Evidence

This section presents selected outputs from the verification queries executed after database initialization. Complete outputs are available in `docs/sample_output.txt`.

6.1 Schema Objects

```
table_schema | table_name
-------------+--------------------
retailsql    | inventory_snapshot
retailsql    | product
retailsql    | sales_order
retailsql    | sales_order_item
retailsql    | store
```

6.2 Referential Integrity

```
sales_order_item.product_id     -> product.product_id
sales_order_item.sales_order_id -> sales_order.sales_order_id
inventory_snapshot.store_id     -> store.store_id
inventory_snapshot.product_id   -> product.product_id
```

6.3 Domain Constraints

```
CHECK (quantity > 0)
CHECK (unit_price >= 0)
CHECK (discount_amount IS NULL OR discount_amount >= 0)
CHECK (on_hand >= 0)
```

6.4 Sample Data

```
store_code | name
-----------+--------------------
S001       | RetailSQL Downtown
S002       | RetailSQL Mall
```

```
sku      | name
---------+------------------
SKU-1001 | Coffee Beans 1kg
SKU-2001 | Wireless Mouse
```

6.5 Relational Join Validation

```
sales_order_id | store_code | sku      | quantity
---------------+------------+----------+----------
1              | S001       | SKU-1001 | 2
1              | S001       | SKU-3001 | 1
2              | S002       | SKU-2001 | 1
```

This confirms that the relational model correctly supports multi-entity joins without ambiguity or data duplication.

7 Operational Environment

RetailSQL is deployed using Docker Compose, providing:

- Isolated PostgreSQL runtime,
- Persistent project-local data storage,
- Reproducible initialization process,
- Explicit health checks.

All SQL scripts are executed manually or sequentially to preserve full control over schema evolution.

8 Conclusion

RetailSQL delivers a clean, enforceable, and production-aligned relational data platform for retail operations. By prioritizing correctness, explicit business rules, and structural clarity, the system provides a robust foundation for transactional integrity and future analytical extensions.

The product is intentionally minimal, deterministic, and transparent, making it suitable as a core data layer in real-world retail architectures.