Columbia University

Driving Scalable Growth: Database Solutions for ABC Foodmart Final Report

Adeel Arif, Depali Kulkarni, Suchakrey Nitisanon, Kristen Campbell, Sana Khan,
Malaikah Khan

APANPS5310

Professor M. Coakley

December 3, 2025

Business Overview

ABC Foodmart is a regional grocery retailer operating two stores in Queens, New York, with plans to expand into Brooklyn by opening three additional locations. As the organization grows, its leadership has recognized the need for a more reliable, scalable, and centralized approach to managing operational data. The company currently maintains essential information such as product details, customer records, transaction histories, and store-level operations primarily through spreadsheets. While this method has supported the business at a smaller scale, it is no longer sufficient for a multi-location environment. ABC Foodmart, therefore, engaged our consulting team to design a relational database system that can serve as the foundation for efficient data management, performance analysis, and informed decision-making as the company expands.

Business Challenges

The shift from a two-store operation to a multi-borough grocery chain introduces several operational and analytical challenges that ABC Foodmart's existing spreadsheet-based processes cannot adequately support. As data volume increases across additional stores, spreadsheets become prone to errors, inconsistencies, and version control issues. Duplicate entries, conflicting product or aisle labels, and manual edits increase the risk of inaccuracies that can affect inventory tracking, customer records, and transaction reporting.

Furthermore, spreadsheets offer limited analytical capability. They cannot easily support multi-table joins, time-series analysis, customer-level tracking, or store-level comparisons at scale. Executives lack timely visibility into performance metrics such as daily revenue, product demand, discount effectiveness, and customer purchasing patterns. Analysts face difficulty producing reliable insights due to inconsistent data structures and the absence of a centralized system.

With upcoming store openings, ABC Foodmart requires a data environment that ensures accuracy, improves operational coordination across locations, and supports both day-to-day operations and long-term strategic analysis. Without such a system, the company risks inefficiencies, reporting delays, and decision-making limitations that could hinder expansion, which can ultimately weaken overall operational performance.

Business Proposal

To address these challenges, our team is designing and implementing a centralized relational database system using PostgreSQL. The proposed solution organizes ABC Foodmart's data into six core tables representing stores, customers, aisles, products, transactions, and transaction-level detail.

This schema follows a normalized structure that reduces redundancy, enforces data integrity through foreign keys, and provides a clear framework for storing and managing information consistently across all locations.

The database will replace the current spreadsheet-driven workflow with a structured system that supports scalable growth. Product and aisle information will be standardized, customer records will be stored consistently, and all transactions will be captured with item-level accuracy. This structure enables analysts to conduct advanced SQL-based analyses on sales trends, product performance, customer behavior, and store-level outcomes. Executives and non-technical users will have access to visual dashboards through Metabase, allowing them to review key performance indicators without writing SQL queries.

The relational database not only addresses ABC Foodmart's immediate operational needs but also establishes the technical foundation required for future expansion. It improves data accuracy, enhances reporting capabilities, reduces operational risk, and ensures the organization is prepared for continued growth across multiple boroughs.

Schedule - Deliverables

Project Component	Checkpoint (PC)	Responsible Persons	Delivery Date
Database Schema Design & Data Storage Requirements	PC3	Kristen, Depali	11/11
Data Transformation & Data Loading	PC4	Adeel, Malaikah	11/18
Real-Time Dashboards (Sales Trends, KPIs)	PC5	Philip, Sana	11/25
Final Presentation	Final Deliverable	Entire Team	12/3
Final Report	Final Deliverable	Entire Team	12/3

Data Description

The dataset contains core components needed to create a relational database design across 11 columns and 1980 rows. It includes fields such as customer_id, store_name, quantity, transaction_date, which naturally match to normalized tables like Stores, Customers, and Transactions. As the data is structured across multiple stores, it directly supports ABC Foodmart's expansion challenge. It also enables basic customer and business metrics, such as revenue, units sold, and purchasing behaviour. The dataset was extracted from [Kaggle](https://www.kaggle.com) (Puri, 2025). Figure 1 displays a preview of the rows and columns included.

grocery_chain_data.csv (163.33 kB)

Detail Compact Column 10 of 11 columns

customer_id	store_name	transactio...	aisle	product_n...	quantity	unit_price
2824	GreenGrocer Plaza	2023-08-26	Produce	Pasta	2	7.46
5506	ValuePlus Market	2024-02-13	Dairy	Cheese	1	1.85
4657	ValuePlus Market	2023-11-23	Bakery	Onions	4	7.38

Figure 1: *Sample Dataset*

Database Schema

The database for ABC Foodmart was designed to achieve the goal of minimizing redundancy, improving data integrity, and supporting efficient analytical queries. The schema follows 3NF, and the design process began by analyzing the dataset to identify core business entities, which include customers, stores, products, aisles, transactions, and transaction line items. Data was separated into different tables so each type of data stays organized and easy to work with as the database grows. The central table in the design is the transactions table, which stores high-level transaction information such as the customer ID, the store where the transaction was made, and the transaction date. To handle the one-to-many relationship between transactions and the products included in each transaction, a separate transaction_items table was created to represent each line item in a receipt, and includes fields such as product, quantity, pricing, and discounts. This design makes it easier to understand what actually happens in a store and lets us answer useful questions, like which products sell the most, how each store is performing, and what customers tend to buy.

Other entities, such as customers, stores, aisles, and products, were created because the dataset needed a way to organize core information that appears across many transactions. For example, customers and stores both show up a lot in the raw data, so giving each its own table prevents duplication and makes it easier to track information about them later. Aisles and products were separated for the same reason because products only need to be stored once with their details and aisle instead of being rewritten for every transaction. Products are connected to aisles using foreign keys, which helps the database keep track of which category each item belongs to. This design also leaves room to collect additional data in the future, such as staffing, vendors, and other operational details, and it allows new tables to be added without disrupting the existing structure.

An entity relationship diagram was created to visually represent the relationships between all tables in the database (Figure 2). There is a one-to-many relationship between customers and transactions because one customer can make many purchases over time, but each transaction is associated with only one customer. Stores and transactions have a one-to-many relationship because each store can process many transactions while each transaction occurs at a one store location. The relationship between aisles and products is modeled as one-to-many since one aisle represents a specific category for products, and a product is typically assigned to one aisle to ensure clear categorization. Additionally, the ERD shows a one-to-many relationship between transactions and transaction_items because you can purchase many items in a single transaction. Finally, a many-to-one relationship exists between transaction_items and products because many transaction items in various purchases can refer to the same product, but each line item refers to only one product.

The following SQL structure was implemented in PostgreSQL to support the database design:

```sql
CREATE TABLE customers (
        customer_id INT PRIMARY KEY,
        loyalty_points INT
    );

    CREATE TABLE stores (
        store_id SERIAL PRIMARY KEY,
        store_name VARCHAR(100) UNIQUE NOT NULL
    );

    CREATE TABLE aisles (
        aisle_id SERIAL PRIMARY KEY,
        aisle_name VARCHAR(100) UNIQUE NOT NULL
    );

    CREATE TABLE products (
        product_id SERIAL PRIMARY KEY,
        product_name VARCHAR(100) NOT NULL,
        aisle_id INT NOT NULL,
        FOREIGN KEY (aisle_id) REFERENCES aisles(aisle_id)
    );
```

```sql
CREATE TABLE transactions (
    transaction_id SERIAL PRIMARY KEY,
    customer_id INT NOT NULL,
    store_id INT NOT NULL,
    transaction_date DATE NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
    FOREIGN KEY (store_id) REFERENCES stores(store_id)
);

CREATE TABLE transaction_items (
    item_id SERIAL PRIMARY KEY,
    transaction_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    unit_price NUMERIC(10, 2) NOT NULL,
    total_amount NUMERIC(10, 2) NOT NULL,
    discount_amount NUMERIC(10, 2),
    final_amount NUMERIC(10, 2) NOT NULL,
    FOREIGN KEY (transaction_id) REFERENCES transactions(transaction_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

ETL Pipeline

The ETL process was implemented using Python while leveraging the Pandas and SQLAlchemy libraries. The raw dataset provided through Kaggle was first loaded into a data frame, allowing a thorough examination of the structure and quality of the data. During this review, it became clear that the product-to-aisle assignments were incorrect; for example, products, such as pasta, were assigned to the produce aisle. Identifying this issue early in the ETL workflow was important because incorrect foreign-key relationships would have caused problems after loading the data into the database. A manual product-to-aisle mapping dictionary was created to ensure each product referenced the correct aisle category. The original dataset included ten different store names but the project scenario only involved five stores. In order to match the scope of the project, the data was manipulated so that each row was randomly assigned one of the five valid stores. Once the dataset was cleaned, the next step was to reshape the raw data into multiple normalized dataframes that matched the database schema. A key design decision was to retain the original customer IDs from the dataset instead of generating new ones to ensure that a customer is accurately referenced in the transactions table, to prevent foreign key errors. After the data was cleaned and separated into its respective tables, SQLAlchemy was used to connect to the PostgreSQL database and load the transformed DataFrames. The loading process followed the order required by foreign key constraints: stores were inserted first, followed by aisles, products, customer transactions, and finally transaction_items. Loading data in this order ensured that all referenced keys existed before loading the rows that reference them.

All SQL scripts, ETL code, schema files, and dashboard query logic used in this project are available in the team's GitHub repository (https://github.com/adeelarif9/grocery-store-sql).

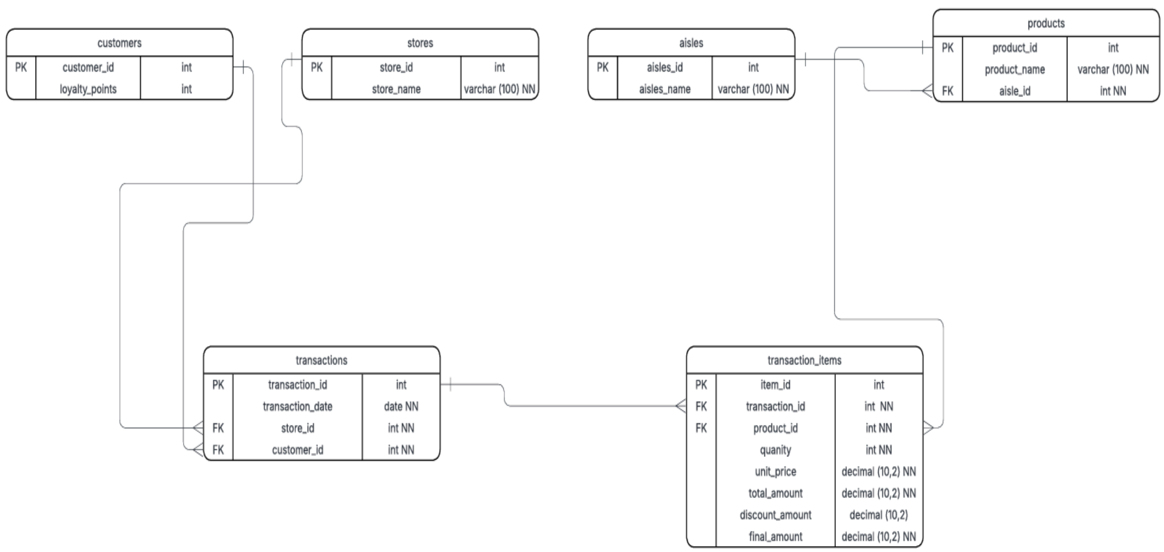

Figure 2: *ER Diagram*

Analytics Applications

This system is designed to support two key groups of users: analysts and executives. Analysts are responsible for performing detailed analytics, writing SQL queries, and solving business problems. Executives rely on quick access to accurate metrics so they can make informed decisions. To meet the needs of both groups, we selected a technology stack that balances performance, usability, and accessibility. PostgreSQL is used as the central database because it can manage large amounts of data, handle complex joins, and support advanced analytical queries. It is reliable and scalable as the business grows.

For daily work with SQL, including writing queries, exploring tables, and creating views, we use pgAdmin4. It provides a simple and intuitive interface that makes it easier for analysts to work efficiently. To deliver insights to non-technical users, we use Metabase for dashboards and reports. Metabase allows us to create clear, interactive dashboards without coding and can automatically send scheduled reports to executives who need regular updates. Together, these tools allow both technical and non-technical users to work from the same database environment. This improves collaboration, ensures consistency in reporting, and supports stronger decision-making across the organization.

Analytical Procedures for Analysts

We created 16 SQL scripts. Each of the scripts can answer one business question. These queries help the analysts understand sales, customers, and store performance. Below are 10 examples of analytical procedures.

<table>
<tr><th>Business Question</th><th>Query Script</th><th>Results</th></tr>
<tr><td>Which stores generate the highest total revenue?</td><td>“TotalRevenueByStore.sql”

SQL
SELECT
    s.store_id,
    s.store_name,
    SUM(ti.final_amount) AS
total_revenue
FROM transactions t
JOIN stores
    ON s.store_id = t.store_id
JOIN transaction_items ti
    ON ti.transaction_id =
t.transaction_id
GROUP BY
    s.store_id,
    s.store_name
ORDER BY
    total_revenue DESC;</td><td><table><tr><th>store_id
[PK] integer</th><th>store_name
character varying (100)</th><th>total_revenue
numeric</th></tr><tr><td>4</td><td>ValuePlus Market</td><td>17687.15</td></tr><tr><td>2</td><td>FreshMart Downtown</td><td>16980.37</td></tr><tr><td>5</td><td>GreenGrocer Plaza</td><td>16539.76</td></tr><tr><td>3</td><td>Corner Grocery</td><td>16017.11</td></tr><tr><td>1</td><td>MegaMart Westside</td><td>14812.92</td></tr></table></td></tr>
<tr><td>Which stores generate the highest revenue after discounts?</td><td>“StoreRevenueAfterDiscount.sql”

SQL
SELECT
    s.store_id,
    s.store_name,</td><td><table><tr><th>store_id
[PK] integer</th><th>store_name
character varying (100)</th><th>revenue_after_discount
numeric</th></tr><tr><td>4</td><td>ValuePlus Market</td><td>15896.97</td></tr><tr><td>2</td><td>FreshMart Downtown</td><td>15131.58</td></tr><tr><td>5</td><td>GreenGrocer Plaza</td><td>14773.07</td></tr><tr><td>3</td><td>Corner Grocery</td><td>14198.28</td></tr><tr><td>1</td><td>MegaMart Westside</td><td>13187.62</td></tr></table></td></tr>
</table>

<table>
<tr>
<td></td>
<td>

```
    SUM(ti.final_amount -
COALESCE(ti.discount_amount,
0)) AS revenue_after_discount
FROM stores s
JOIN transactions tr
    ON tr.store_id =
s.store_id
JOIN transaction_items ti
    ON ti.transaction_id =
tr.transaction_id
GROUP BY
    s.store_id,
    s.store_name
ORDER BY
    revenue_after_discount
DESC;
```

</td>
<td></td>
</tr>
<tr>
<td>What is the average order value at each store?</td>
<td>

“AverageOrderValueByStore.sql”

```
SQL
SELECT
    s.store_name,
    SUM(ti.final_amount) /
COUNT(DISTINCT
t.transaction_id) AS
average_order_value
FROM stores s
JOIN transactions t ON
t.store_id = s.store_id
JOIN transaction_items ti ON
ti.transaction_id =
t.transaction_id
GROUP BY s.store_name
ORDER BY average_order_value
DESC;
```

</td>
<td>

store_name character varying (100)	average_order_value numeric
ValuePlus Market	44.2178750000000000
GreenGrocer Plaza	42.7383979328165375
FreshMart Downtown	41.3147688564476886
MegaMart Westside	39.7129222520107239
Corner Grocery	39.1616381418092910

</td>
</tr>
<tr>
<td>Which stores give out the highest total discount amounts?</td>
<td>

“StorewiseDiscount.sql”

```
SQL
SELECT
    s.store_id,
```

</td>
<td>

store_id [PK] integer	store_name character varying (100)	total_discount_amount numeric
2	FreshMart Downtown	1848.79
3	Corner Grocery	1818.83
4	ValuePlus Market	1790.18
5	GreenGrocer Plaza	1766.69
1	MegaMart Westside	1625.30

</td>
</tr>
</table>

```
    s.store_name,
SUM(COALESCE(ti.discount_amount, 0)) AS total_discount_amount
FROM stores s
JOIN transactions tr
    ON tr.store_id = s.store_id
JOIN transaction_items ti
    ON ti.transaction_id = tr.transaction_id
GROUP BY
    s.store_id,
    s.store_name
ORDER BY
    total_discount_amount DESC;
```

Which aisles generate the highest net revenue?

“NetRevenueByAisle.sql”

```sql
SELECT
    a.aisle_id,
    a.aisle_name,
    SUM(ti.final_amount - COALESCE(ti.discount_amount, 0)) AS net_revenue
FROM aisles a
JOIN products p
    ON p.aisle_id = a.aisle_id
JOIN transaction_items ti
    ON ti.product_id = p.product_id
GROUP BY
    a.aisle_id,
    a.aisle_name
ORDER BY
    net_revenue DESC;
```

aisle_id [PK] integer	aisle_name character varying (100)	net_revenue numeric
1	Produce	24962.14
2	Dairy	15179.85
8	Meat & Seafood	11879.79
5	Canned Goods	8416.84
3	Bakery	4773.78
11	Beverages	4054.57
4	Snacks & Candy	3920.55

Which products generate the most revenue?

“TopProductRevenue.sql”

```sql
SELECT
    p.product_name,
    SUM(ti.final_amount) AS
total_revenue
FROM products p
JOIN transaction_items ti
    ON ti.product_id =
p.product_id
GROUP BY
    p.product_name
ORDER BY
    total_revenue DESC;
```

product_name character varying (100)	total_revenue numeric
Tomatoes	5381.71
Bread	5349.36
Potatoes	5306.25
Chicken Breast	5172.88
Eggs	4825.59
Bananas	4710.76

Which products sell the most units?

“Top20SellingProducts.sql”

```sql
SELECT
    p.product_name,
    SUM(ti.quantity) AS
total_units
FROM products p
JOIN transaction_items ti ON
ti.product_id = p.product_id
GROUP BY p.product_name
ORDER BY total_units DESC
LIMIT 20;
```

product_name character varying (100)	total_units bigint
Chicken Breast	379
Tomatoes	366
Bread	359
Potatoes	353
Onions	342

Which customers spend the most in total?

“TopCustomerSpend.sql”

```sql
SELECT
    c.customer_id,
    SUM(ti.final_amount) AS
total_spend
FROM customers c
JOIN transactions t
    ON t.customer_id =
c.customer_id
JOIN transaction_items ti
    ON ti.transaction_id =
t.transaction_id
GROUP BY
```

customer_id [PK] integer	total_spend numeric
1725	1549.79
1829	1542.64
1858	1453.21
1478	1411.19
1472	1364.00

<table>
<tr>
<td></td>
<td>

```
    c.customer_id
ORDER BY
    total_spend DESC;
```

</td>
<td></td>
</tr>
<tr>
<td>Which customers visit the store most frequently?</td>
<td>

“CustomerRepeatVisits.sql”

```SQL
SELECT
    c.customer_id,
    COUNT(*) AS
number_of_visits
FROM customers c
JOIN transactions t ON
t.customer_id = c.customer_id
GROUP BY c.customer_id
ORDER BY number_of_visits
DESC;
```

</td>
<td>
<table>
<tr><th>customer_id
[PK] integer</th><th>number_of_visits
bigint</th></tr>
<tr><td>8381</td><td>4</td></tr>
<tr><td>9056</td><td>3</td></tr>
<tr><td>8933</td><td>3</td></tr>
<tr><td>2020</td><td>3</td></tr>
</table>
</td>
</tr>
<tr>
<td>How do total sales change over time (by day)?</td>
<td>

“SalesTrendByDay.sql”

```SQL
SELECT
    t.transaction_date,
    SUM(ti.final_amount) AS
total_sales
FROM transactions t
JOIN transaction_items ti
    ON ti.transaction_id =
t.transaction_id
GROUP BY
    t.transaction_date
ORDER BY
    t.transaction_date;
```

</td>
<td>
<table>
<tr><th>transaction_date
date</th><th>total_sales
numeric</th></tr>
<tr><td>2025-06-27</td><td>368.72</td></tr>
<tr><td>2025-06-30</td><td>603.68</td></tr>
<tr><td>2025-07-05</td><td>1356.96</td></tr>
<tr><td>2025-07-10</td><td>476.96</td></tr>
</table>
</td>
</tr>
</table>

These SQL scripts allow analysts to explore customer behavior, sales trends, product performance, and discount usage in a structured and efficient way. Analyses like these cannot be done easily in Excel or Google Sheets because they require operations such as joins, filtering, and aggregation on very large tables, which spreadsheets are not designed to handle at scale.

The analysts can connect to PostgreSQL through pgAdmin4. They just need to open the pgAdmin_SQL_Query folder (pgAdmin_SQL_Query) and run the 16 SQL procedures. They can adjust the date filters, modify conditions, or join additional tables depending on the analysis they want to perform. If they require faster performance, they also have the option to use materialized views. Overall, analysts have full access to the tables in the database and can write their own queries to answer specific business questions

For executives and non-technical users, we use Metabase to create the dashboard. The dashboard is designed to address all business requirements and provide an easy way to explore key metrics. Users can open the ABC Foodmart dashboard through the web link provided. They can select a start date, end date, store, or aisle to customize their view. This allows them to monitor daily revenue by store, drill down into product or aisle performance, view aisle traffic and promotion usage, and track daily real-time sales. They can also receive scheduled email reports for regular updates. Sample dashboard views are presented in Figures 3, 4, and 5.

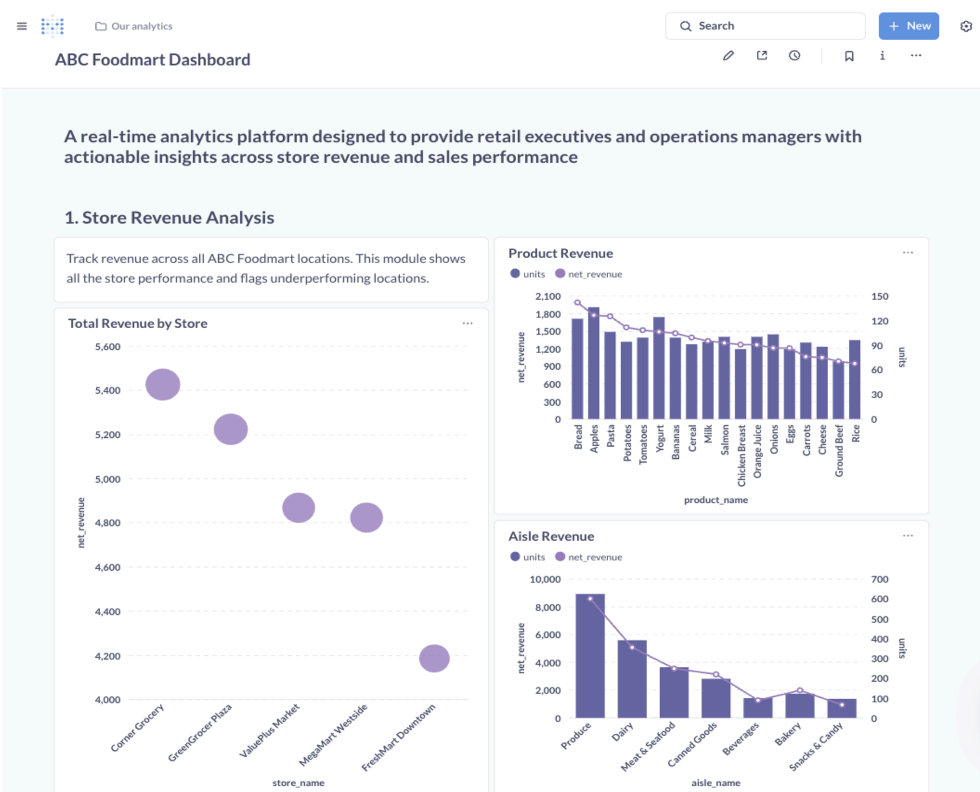

Figure 3: *Store Revenue Analysis Dashboard Example*

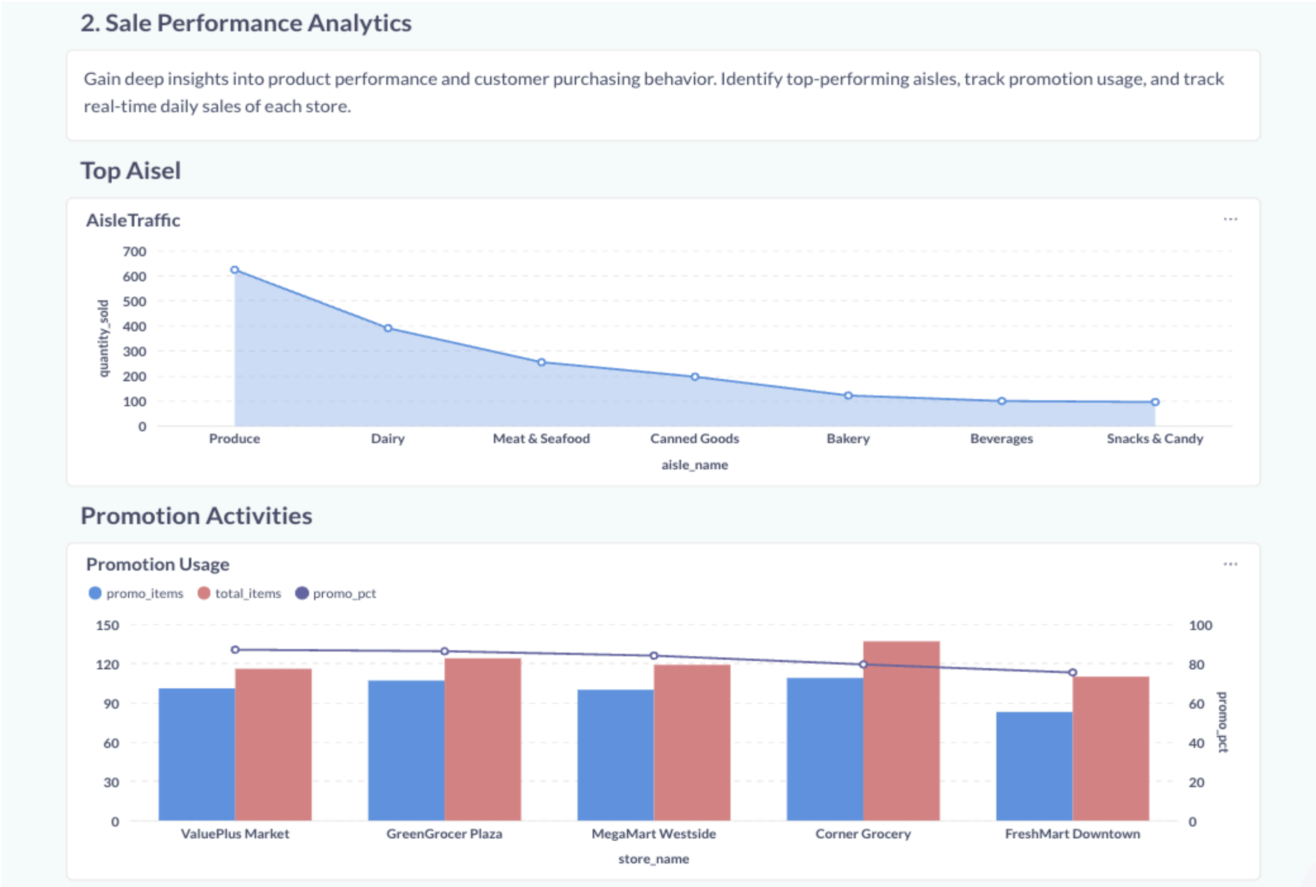

Figure 4: *Sales Performance Analytics Dashboard*

Figure 5: *Real-time Daily Sales Dashboard*

The dashboard provides several important benefits. It gives executives a quick overview of store performance, allowing them to see all key numbers in one place without having to run SQL queries. It shows trends and potential problem areas, such as daily and monthly sales patterns, which helps with planning for inventory, staffing, and budgeting. The drilldown feature makes it easy to identify which products contribute to low profits. Aisle traffic insights show which areas are crowded and which are quiet, allowing store managers to introduce strategies such as adding promotions to low-traffic aisles. Promotion usage helps evaluate whether discounts are effective, showing whether a promotion increases sales or simply reduces profit. With this information, managers can decide whether to continue or stop a discount. Daily real-time sales give store managers better control over day-to-day operations by helping them adjust staffing and inventory as sales rise or fall throughout the day.

Executives do not need SQL skills to access these insights. By simply selecting filters for date, store, or aisle, they can quickly get the information they need. This saves time, increases visibility, and improves efficiency.

To ensure reliable performance and reduce redundancy in database operations, we use a straightforward design. The system includes one primary PostgreSQL server for write operations and a read replica dedicated to the Metabase dashboard to reduce load during busy hours. We also create materialized views for heavy queries such as revenue, aisle traffic, and daily sales to improve speed for both analysts and executives. In addition, we run daily automated backups to protect the data and allow fast recovery if anything goes wrong. Hosting the system on the cloud is recommended because it allows the client to scale up or down easily as the business grows.

Conclusion

Overall, the goal of the project is to replace ABC Foodmart's spreadsheet operations with a centralized relational database. This would help the business expand, improve data accuracy, and allow for better analysis across stores, customers, products, and transactions. We achieved this by designing a 3NF PostgreSQL schema with clearly defined tables and foreign-key relationships. Then, we built a Python-based ETL pipeline that cleaned the Kaggle dataset and loaded data into PostgreSQL in a dependency-aware order using SQLAlchemy. With this setup, analysts can run SQL procedures on sales, store performance, customer behaviour, product demand, discounts, and time-based trends. Executives and non-technical users can access a Metabase dashboard showing store revenue, aisle traffic, and promotion usage, with easy filters instead of having to run manual SQL. The combination of the database, ETL process, and analytics layers improves data reliability and reduces the risks of using spreadsheets. It also gives ABC Foodmart faster, more useful insights to support operational decisions and growth across multiple locations.

References:

Puri, P. (2025). Grocery Store Sales Dataset in 2025 - 1900+ Record. Kaggle.com. https://www.kaggle.com/datasets/pratyushpuri/grocery-store-sales-dataset-in-2025-1900-record/