Final Project Report

Authors: Jie Chen, Jiarong Guo, Yuxin Li, Zhuoran Li, Chenyan Wu, Ennan Zhao

Date: December 3, 2025

Problem Statement

ABC Foodmart is a neighborhood grocery chain with two long-standing stores in Queens, NY. For more than thirty years, the company has operated using Excel spreadsheets, email attachments, and paper binders to manage critical business functions such as staff schedules, vendor information, product lists, sales logs, and accounting. Although this informal workflow has been good enough for two stores, it has led to slow decision making, inconsistencies in reporting, and occasional data entry errors.

To address these challenges, our team proposes the design and implementation of a fully centralized relational database system tailored to ABC Foodmart's operational and analytical needs. The system will consolidate all product, sales, customer, and employee information into a unified structure that eliminates spreadsheet fragmentation, reduces reporting inconsistencies, and supports data-driven decision making across the organization.

The database will provide a robust foundation for product and category management by maintaining a complete, consistently structured product catalog and enabling category-level organization and reporting. It will also store all sales transactions to support time-series analysis and detailed revenue reporting by store, category, and product.

In addition, the system will house customer demographic and geographic information, link customers to their purchase histories, and enable advanced analytics such as loyalty segmentation and customer lifetime value estimation. Employee data will also be integrated in a structured format, including personal profiles, employment details, store assignments, and demographic attributes. This will allow ABC Foodmart to conduct HR-focused analyses such as turnover trends, tenure patterns, workforce composition, and ultimately explore how employee performance relates to store-level outcomes.

The final deliverable is designed to serve the needs of two core user groups. Analysts will have full SQL and Python access to the underlying database, enabling them to run complex analyses and build custom models. Executives will interact with the system through interactive, automatically refreshed dashboards in Metabase, giving them clear, timely insights without requiring technical expertise.

Proposal

Our proposal to ABC Foodmart is to replace the fragmented system with a fully normalized 3NF PostgreSQL relational database. A reproducible Python ETL pipeline and a Metabase dashboard suite for executives also support it.

Our initial plan of action includes the following steps:

- Brainstorm and define a list of business requirements.
- Design an ERD with important tables and turn it into a relational database system in SQL.
- Build an ETL pipeline.
- Implement at least 10 analytical procedures. Tackle the business requirement and write SQL queries accordingly.
- Determine how our customers will interact with the database system. Publish executive dashboards in Metabase while enabling analysts to query in SQL.

The new database can help ABC Foodmart make faster and error-reduced decisions: real-time sales by store, category, and daypart monitoring; early detection of outliers; store inventory management; and customer repeat rate tracking. Over time, ABC Foodmart can extend this foundation to performance management (employee and store KPIs), demand forecasting, promotion effectiveness, and location planning. Our 3NF model, with a proper delivery structure for different uses of the customers, can turn isolated files into a living system for growth.

Team Structure & Timeline

The team has agreed to collaborate on all tasks. Each member contributes across design, coding, and documentation while serving as a lead in one primary area.

- Team Contract

Task	Description
Database Design	Develop a relational database in SQL. Define keys and constraints, and normalize tables to 3NF.
ETL Process	Build a Python ETL pipeline using pandas and SQLAlchemy.
SQL Development	Create complex SQL queries to meet the business requirements.
Visualization	Design interactive Metabase dashboards and compile the final presentation and report integrating ERD, ETL summary, and SQL findings
Project Coordination, Reporting, & Review	Manage timelines, synchronize files, conduct peer reviews, and ensure quality and cohesiveness of the final submission.

- Timeline

Phase	Week
Phase 1 – Business Problem Identification	Week 8 – 9
Phase 2 – Database Design & Implementation	Weeks 9 – 10
Phase 3 – ETL & Data Integration	Weeks 10 – 11
Phase 4 – Visualization & Final Reporting	Weeks 11 – 13

Sample Data, Full Dataset, & Rationale

We chose ABC Foodmart because a grocery chain expanding from two to five stores needs a robust, scalable database system to replace spreadsheets and paper. We reviewed several public retail datasets and selected a multi-file option (see the link below: customers.csv, products.csv, categories.csv, sales.csv, employees.csv, cities.csv, countries.csv) that maps cleanly to a grocery context and supports a 7–10 table relational design. Our initial decision criteria focused on: the existence of several important business entities (customers, products, categories, employees, locations, transactions) that fit the business scenario; time-stamped sales for trend analysis; join

keys we can standardize; and enough volume and variety to justify normalization. We validated that the files can be normalized to 3NF and loaded via a Python ETL (pandas and SQLAlchemy) into PostgreSQL.

- Sample data

customer_id	name	email	gender	signup_date	country
1	Allison Hill	donaldgarcia@example.net	Other	2/15/25	Benin
2	Amanda Davi	williamjohnson@example.org	Male	3/30/24	Reunion
3	Connie Lawre	blakeerik@example.com	Male	5/13/22	Cape Verde
4	Phillip Garcia	wdavis@example.net	Other	12/26/21	Aruba
5	Kimberly Dud	smiller@example.net	Female	10/4/23	Tokelau
6	Corey Jones	kendragalloway@example.org	Male	12/30/24	Hong Kong
7	Mark Diaz	michael41@example.net	Male	11/3/23	Martinique
8	Carla Gray	gabriellecameron@example.org	Male	10/22/21	Antigua and Barbuda
9	Jennifer Jone	jason76@example.net	Other	12/16/22	Estonia
10	Patrick Ryan	zhurst@example.com	Male	10/4/21	Burundi

Figure 1: sample data from customers.csv

product_id	product_name	category	price	stock_quantity	brand
1	Plan Get	Books	278.27	533	BrandA
2	Field Yet	Beauty	479.56	5	BrandA
3	Technology Her	Books	332.59	268	BrandA
4	Decade Pick	Toys	198.16	944	BrandB
5	Finally Develop	Beauty	307.6	862	BrandC
6	Play Several	Electronics	406.48	231	BrandA
7	Various Account	Electronics	200.95	267	BrandC
8	Build Of	Beauty	262.73	198	BrandA
9	Save Keep	Clothing	155.47	355	BrandD

Figure 2: Sample data from products.csv

- Dataset: https://www.kaggle.com/datasets/andrexibia/grocery-sales-dataset?select=categories.csv

Database Schema

Entity-Relationship Design

Firstly, our database design and ER diagram organizes the dataset into dimensional and stable tables. The 6 tables that contain mostly stable data are countries, cities, categories, products, customers, and employees. The last one, the sales table, serves as the central fact table that records transactions. This structure separates slow-changing descriptive data from high-volume operational data.

Then, we established one-to-many relationships in the schema. They are: countries to cities (1:N), cities to both customers and employees (1:N each), categories to products (1:N), and employees, customers, and products each to sales (1:N). We use foreign keys to connect and secure each of these relationships. For example, cities.country_id references countries.country_id, so that every city belongs to exactly one country. At the same time, countries can have multiple cities. We made all these relationships mandatory by making foreign keys NOT NULL. This prevents error records like a city without a country or a sale without a customer.

We did consider implementing many-to-many relationships for sales transactions, since one purchase might have multiple products. However, to be more clear and straightforward, we decided to treat every item purchased as a separate sale. In this way, each sales row represents one product in one transaction, and transaction_number uniquely identifies that transaction.

Normalization

All tables are in their third normal form. For the first normal form, each column contains atomic values. We split names into first_name, middle_initial, and last_name rather than directly using full names. Product attributes like resistant and is_allergic are separate columns instead of multi-valued fields. For the second normal form, we made sure that every non-key attribute depends on the complete primary key. For example, in the sales table, total_price depends on sales_id, not on partial key components. In products, price depends on product_id, not just category_id.

For the third normal form, we made sure that every non-key attribute must depend directly on the primary key and not indirectly on any other non-key attribute. For example, customer locations do not store country information directly. Instead, they refer to the city table using city_id as foreign key, then refer to the country table. This means that country names appear only in the countries table. If we stored country names directly in the customers table, we would have a transitive dependency (customer_id → city_id → country_name), and this violates the rule of 3NF. The same rule applies to the rest of the database design as well. Employees reference cities for location data. Products reference categories for classification. Sales references employees, customers, and products rather than repeating their attributes.

Schema Implementation

All of our primary keys are non-meaningful integers (IDs), rather than real-world customer or product codes. This protects the schema from business identifier changes. If a product name changes or a customer's details are changed, we do not need to make updates through foreign key relationships. The integer keys stay constant while descriptive attributes can change freely.

Our foreign key constraints use ON DELETE RESTRICT and ON UPDATE CASCADE uniformly. The RESTRICT policy prevents accidental deletion of “parent records” that still “have children”. For example, you cannot delete a country that has cities, or a product is recorded in sales history. The CASCADE policy allows key corrections to propagate automatically. If we update a category_id, the change flows to all products in that category. This combination keeps the data accurate while also making the system easy to update.

Data type choices reflect business requirements and technical constraints. We use NUMERIC(12,2) for all prices instead of the less precise FLOAT type. This ensures we have exact decimal precision for currency, which is critical for avoiding potential rounding errors in future financial calculations. For high-volume transaction IDs, like sales_id, we use the larger BIGINT data type. This predicts rapid growth, since a standard INTEGER can only handle up to about 2.1 billion records before it runs out of capacity. For slower-growing fields, like IDs in lookup tables, the standard INTEGER is enough. We also specify reasonable lengths for text fields (e.g., VARCHAR(60) for names and VARCHAR(200) for product names).
CHECK constraints enforce business rules at the database level. The quantity > 0 CHECK prevents any negative sales. The discount >= 0 AND discount <= 1 constraint makes sure that discounts are between 0 and 100%. The price >= 0 CHECK prevents negative pricing. The gender IN ('Male','Female','Other','Unspecified') constraint makes sure inputs are valid, inclusive genders. The UNIQUE constraints on country_name, country_code, category_name, and transaction_number prevent repetitions. The UNIQUE NULLS NOT DISTINCT on country_code makes sure that each country_code value (including NULL) appears at most one time.

Lucidchart Link:
https://lucid.app/lucidchart/2340fff3-fa95-48c0-bfc6-c74d398bdd5d/edit?viewport_loc=-212%2C179%2C3264%2C1603%2C0_0&invitationId=inv_5a185eb4-6a8b-4f24-be71-67ec78dacade

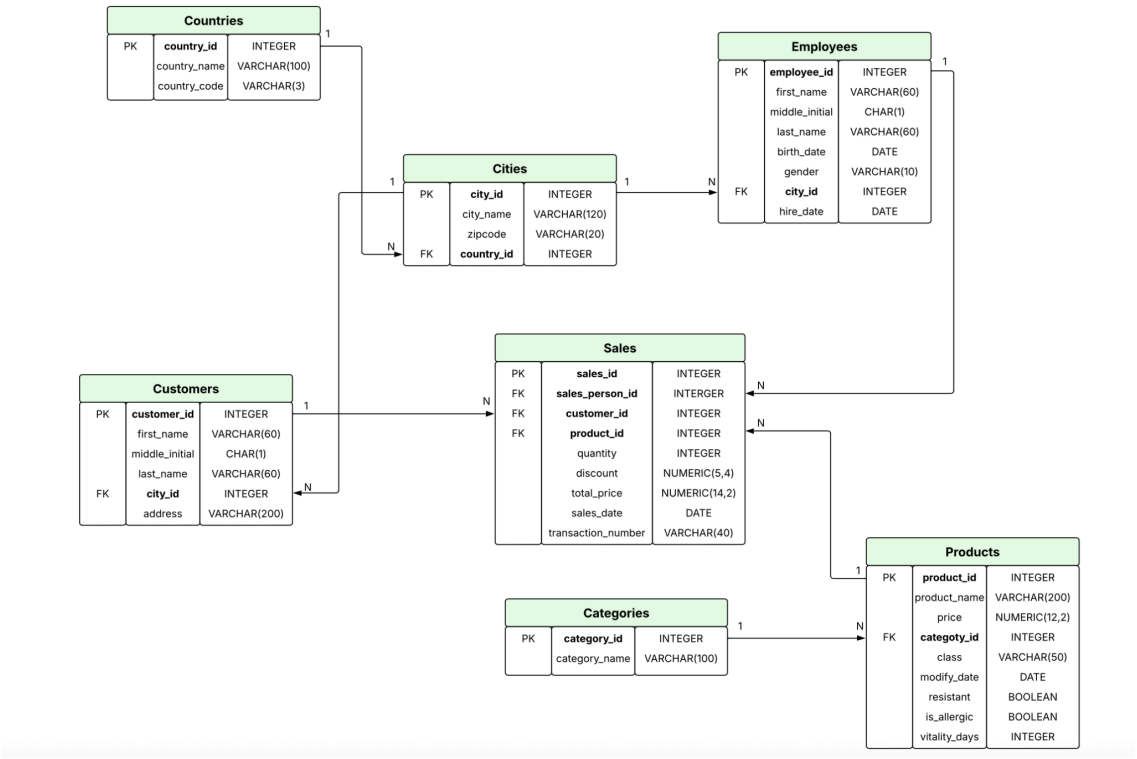


Figure 3: ERD Diagram

ETL Process

The ETL process for our project was designed to reliably convert a set of disparate CSV files into a clean, fully normalized PostgreSQL database aligned with our seven-table 3NF schema: countries, cities, customers, employees, categories, products, and sales. The workflow began by extracting each dataset into pandas DataFrames and standardizing column names to match the database schema. Before any loading occurred, all target tables in PostgreSQL were truncated with the RESTART IDENTITY CASCADE command. This approach ensured idempotency, prevented duplicate primary keys, and guaranteed that each ETL run would produce the same deterministic result.

During transformation, we applied a series of systematic cleaning and normalization steps to align the raw data with the constraints defined in our schema. Names were sanitized and split into first, middle initial, and last components. Gender fields were standardized into the four accepted values of Male, Female, Other, and Unspecified. Dates were parsed from multiple inconsistent formats and converted into ISO-standard DATE values. Boolean-like fields were mapped into true boolean values, and malformed or ambiguous entries were safely converted to NULL where appropriate. Numeric fields, including price, quantity, discount, and vitality_days, were stripped of extraneous characters and validated against business rules such as ensuring that quantities were strictly positive, discounts fell between zero and one, and prices were

non-negative. These steps were essential for meeting CHECK constraints and avoiding insertion failures during the load phase.

A key part of the transformation process involved preparing foreign key relationships. Country and city names were mapped to their corresponding IDs, categories were assigned numeric category identifiers, and all references in the customers, employees, products, and sales tables were reconciled to these parent tables. This relational mapping ensured that every row inserted into child tables would satisfy foreign key enforcement in PostgreSQL. The sales table required additional attention because the raw input contained inconsistent values for discounts and total_price. As part of our design choice, we intentionally set total_price to zero and delegated its computation to a PostgreSQL trigger to guarantee consistent and accurate calculation based on quantity, unit price, and discount logic.

Once all transformations were complete, the cleaned DataFrames were loaded into PostgreSQL in an order that respected foreign key dependencies: countries first, followed by cities, customers, employees, categories, products, and finally sales. High-volume tables, especially sales, were written in chunks using pandas to_sql to maintain performance and avoid memory strain. This loading strategy ensured both reliability and scalability while maintaining strict referential integrity across the database.

Throughout the development process, our ETL design emphasized robustness, maintainability, and alignment with the conceptual model of the database. By enforcing data consistency early, structuring a dependency-aware loading sequence, and building repeatability into the workflow, the pipeline provides a dependable mechanism for refreshing or expanding the database as new data becomes available. We also documented several questions for future refinement, such as whether transaction_number should be enforced as a unique business key and whether certain malformed fields should default to NULL or require imputation. These considerations reflect our intention to design a system that not only functions correctly but mirrors realistic data-governance practices.

All ETL code, implemented in Python using pandas, SQLAlchemy, and psycopg2, has been uploaded to our GitHub repository: https://github.com/jc6413-ui/5310-Group2-code

Analytical Applications

To demonstrate how the relational database supports business decision-making, we implemented 12 analytical procedures across three themes: product and category performance, salesperson effectiveness, and geographic opportunities. Each analysis is linked to a specific client need, uses multiple normalized tables, and is fully operational through SQL queries executed directly on the database.

1. Product & Category Analysis

1.1 Monthly Revenue Trends by Category

```
SELECT
    cat.category_name,
    EXTRACT(MONTH FROM s.sales_date) AS month,
    SUM(s.total_price) AS monthly_revenue
FROM sales s
JOIN products p ON s.product_id = p.product_id
JOIN categories cat ON p.category_id = cat.category_id
GROUP BY cat.category_name, month
ORDER BY cat.category_name, month;
```

category_name character varying (100)	month numeric	monthly_revenue numeric
Beverages	1	265010.48
Beverages	2	243361.08
Beverages	3	220100.45
Beverages	4	241889.08
Beverages	5	74865.80

Client Need: Understand category contributions and detect demand shifts.
Tables Used: *sales, products, categories*
Summary: We calculate monthly revenue per category by joining products to categories and extracting the month from sales_date. The

1.2 Category Seasonality Index

```
WITH monthly_rev AS (
    SELECT
        cat.category_name,
        EXTRACT(MONTH FROM s.sales_date) AS month,
        SUM(s.total_price) AS revenue
    FROM sales s
    JOIN products p ON s.product_id = p.product_id
    JOIN categories cat ON p.category_id = cat.category_id
    GROUP BY cat.category_name, month
),
category_avg AS (
    SELECT
        category_name,
        AVG(revenue) AS avg_revenue
    FROM monthly_rev
    GROUP BY category_name
)
SELECT
    m.category_name,
    m.month,
    m.revenue,
    ROUND(m.revenue / c.avg_revenue, 2) AS seasonality_index
FROM monthly_rev m
JOIN category_avg c
    ON m.category_name = c.category_name
ORDER BY m.category_name, m.month;
```

category_name character varying (100)	month numeric	revenue numeric	seasonality_index numeric
Beverages	1	265010.48	1.51
Beverages	2	243361.08	1.39
Beverages	3	220100.45	1.25
Beverages	4	241889.08	1.38
Beverages	5	74865.80	0.43

Client Need: Identify which categories exhibit stable vs. seasonal patterns.
Tables Used: *sales, products, categories*
Summary: Revenue is aggregated by category–month and normalized against the category's own average to produce a

resulting time-series reveals growth patterns, off-season periods, and category mix changes that support both planning and revenue forecasting.

seasonality index. Values above 1 indicate peak months, guiding assortment, procurement, and promotional timing.

1.3 Peak Month Identification

```
WITH monthly AS (
    SELECT
        cat.category_name,
        EXTRACT(MONTH FROM s.sales_date) AS month,
        SUM(s.total_price) AS revenue
    FROM sales s
    JOIN products p ON s.product_id = p.product_id
    JOIN categories cat ON p.category_id = cat.category_id
    GROUP BY cat.category_name, month
)
SELECT category_name, month, revenue
FROM (
    SELECT
        category_name,
        month,
        revenue,
        RANK() OVER (PARTITION BY category_name ORDER BY revenue DESC) AS rnk
    FROM monthly
) ranked
WHERE rnk = 1
ORDER BY category_name;
```

category_name character varying (100)	month numeric	revenue numeric
Beverages	1	265010.48
Cereals	1	292262.81
Confections	1	419263.41
Dairy	1	266983.76
Grain	1	238437.17

1.4 Top-Grossing Product Within Each Category

```
WITH product_revenue AS (
    SELECT
        cat.category_name,
        p.product_id,
        p.product_name,
        SUM(s.total_price) AS revenue
    FROM sales s
    JOIN products p ON s.product_id = p.product_id
    JOIN categories cat ON p.category_id = cat.category_id
    GROUP BY cat.category_name, p.product_id, p.product_name
),
ranked AS (
    SELECT
        category_name,
        product_id,
        product_name,
        revenue,
        RANK() OVER (PARTITION BY category_name ORDER BY revenue DESC) AS rnk
    FROM product_revenue
)
SELECT
    category_name,
    product_id,
    product_name,
    revenue AS total_revenue
FROM ranked
WHERE rnk = 1
ORDER BY category_name;
```

	category_name character varying (100)	product_id integer	product_name character varying (200)	total_revenue numeric
1	Beverages	104	Tia Maria	60895.37
2	Cereals	98	Shrimp - 31/40	59438.61
3	Confections	223	Pail With Metal Handle 16l W...	77733.58

Client Need: Determine optimal timing for marketing and inventory allocation.
Tables Used: *sales, products, categories*
Summary: Using a monthly revenue CTE and a RANK() window function, we identify the single highest-revenue month for each category. Leadership can immediately see when each category performs best.

Client Need: Highlight hero products that drive category performance.
Tables Used: *sales, products, categories*
Summary: Total revenue per product is computed and ranked using RANK() by category. The top performer in each category is surfaced to support promotion, sourcing, and inventory prioritization.

1.5 Lowest-Grossing Product Within Each Category

```
WITH product_revenue AS (
    SELECT
        cat.category_name,
        p.product_id,
        p.product_name,
        SUM(s.total_price) AS revenue
    FROM sales s
    JOIN products p ON s.product_id = p.product_id
    JOIN categories cat ON p.category_id = cat.category_id
    GROUP BY cat.category_name, p.product_id, p.product_name
),
ranked AS (
    SELECT
        category_name,
        product_id,
        product_name,
        revenue,
        RANK() OVER (PARTITION BY category_name ORDER BY revenue ASC) AS rnk
    FROM product_revenue
)
SELECT
    category_name,
    product_id,
    product_name,
    revenue AS total_revenue
FROM ranked
WHERE rnk = 1
ORDER BY category_name;
```

	category_name character varying (100)	product_id integer	product_name character varying (200)	total_revenue numeric
1	Beverages	325	Halibut - Fletches	1582.14
2	Cereals	390	Seedlings - Mix, Organic	505.96
3	Confections	107	Bread - French Baquette	3572.72
4	Dairy	283	Bread Fig And Almond	469.93
5	Grain	75	Chef Hat 20cm	2513.44

Client Need: Identify weak performers for rationalization or repositioning.
Tables Used: *sales, products, categories*
Summary: Similar to the previous analysis but ranking revenue in ascending order, this highlights products that may require discounting, assortment changes, or reconsideration.

1.6 Products Most Impacted by Discounts

```
WITH product_qty AS (
    SELECT
        p.product_id,
        p.product_name,
        CASE
            WHEN s.discount > 0 THEN 'discounted'
            ELSE 'no_discount'
        END AS price_type,
        AVG(s.quantity) AS avg_qty
    FROM sales s
    JOIN products p ON s.product_id = p.product_id
    GROUP BY p.product_id, p.product_name, price_type
),
pivoted AS (
    SELECT
        product_id,
        product_name,
        MAX(CASE WHEN price_type = 'discounted' THEN avg_qty END) AS avg_qty_discount,
        MAX(CASE WHEN price_type = 'no_discount' THEN avg_qty END) AS
avg_qty_no_discount
    FROM product_qty
    GROUP BY product_id, product_name
),
impact AS (
    SELECT
        product_id,
        product_name,
        avg_qty_discount,
        avg_qty_no_discount,
        (avg_qty_discount - avg_qty_no_discount) AS qty_increase_due_to_discount
    FROM pivoted
)
SELECT
    product_id,
    product_name,
    avg_qty_discount,
    avg_qty_no_discount,
    qty_increase_due_to_discount
FROM impact
ORDER BY qty_increase_due_to_discount DESC NULLS LAST
LIMIT 10;   -- top 10 most sensitive products
```

	product_id [PK] integer	product_name character varying (200)	avg_qty_discount numeric	avg_qty_no_discount numeric	qty_increase_due_to_discount numeric
1	364	Cattail Hearts	23.0000000000000000	11.1304347826086957	11.8695652173913043
2	214	French Pastry - Mini Chocol…	18.1000000000000000	10.1538461538461538	7.9461538461538462
3	399	Berry Brulee	18.0000000000000000	10.1388888888888889	7.8611111111111111
4	298	Pop Shoppe Cream Soda	19.6666666666666667	11.8085106382978723	7.8581560283687944
5	27	Chocolate - Compound Coat…	19.0000000000000000	11.6551724137931034	7.3448275862068966
6	111	Scallops 60/80 Iqf	17.8000000000000000	10.5142857142857143	7.2857142857142857
7	121	Muffin Batt - Blueberry Passi…	19.4285714285714286	12.1951219512195122	7.2334494773519164
8	290	Sauce - Rosee	18.0000000000000000	10.8285714285714286	7.1714285714285714
9	116	Steam Pan - Half Size Deep	18.5000000000000000	11.4814814814814815	7.0185185185185185
10	234	Tofu - Firm	20.2857142857142857	13.4000000000000000	6.8857142857142857

Client Need: Measure discount elasticity and avoid unnecessary margin loss.
Tables Used: *sales, products*
Summary: We compare average quantities sold with and without discounts by classifying each transaction line. The difference quantifies which products respond strongly to price reductions, enabling targeted promotions.

2. Salesperson Analysis

2.1 Revenue-Based Salesperson Ranking

```
SELECT
    emp.employee_id,
    emp.first_name || ' ' || emp.last_name AS employee_name,
    SUM(s.total_price) AS revenue,
    RANK() OVER (ORDER BY SUM(s.total_price) DESC) AS revenue_rank
FROM sales s
JOIN employees emp ON emp.employee_id = s.sales_person_id
GROUP BY emp.employee_id, employee_name
ORDER BY revenue DESC;
```

	employee_id [PK] integer	employee_name text	revenue numeric	revenue_rank bigint
1	16	Chadwick Walton	607337.21	1
2	14	Wendi Buckley	598446.45	2
3	7	Chadwick Cook	591659.32	3
4	3	Pablo Cline	589504.76	4
5	13	Katina Marks	589415.83	5

Client Need: Evaluate individual performance for bonuses and territory planning.
Tables Used: *sales, employees*
Summary: Total revenue per employee is aggregated and ranked using a window function, creating a dynamic leaderboard that managers can filter by region or time.

2.2 Top-Grossing Employee in Each City

```
WITH city_employee_revenue AS (
    SELECT
        city.city_name,
        emp.employee_id,
        emp.first_name || ' ' || emp.last_name AS employee_name,
        SUM(s.total_price) AS revenue
    FROM sales s
    JOIN employees emp ON s.sales_person_id = emp.employee_id
    JOIN customers cust ON s.customer_id = cust.customer_id
    JOIN cities city ON cust.city_id = city.city_id
    GROUP BY city.city_name, emp.employee_id, employee_name
),
ranked AS (
    SELECT
        city_name,
        employee_id,
        employee_name,
        revenue,
        RANK() OVER (PARTITION BY city_name ORDER BY revenue DESC) AS rnk
    FROM city_employee_revenue
)
SELECT
    city_name,
    employee_id,
    employee_name,
    revenue AS total_revenue
FROM ranked
WHERE rnk = 1
ORDER BY city_name;
```

Client Need: Compare performance in local markets with differing demand patterns.
Tables Used: *sales, employees, customers, cities*
Summary: City-level revenue per employee is computed and ranked within each city. This identifies local top performers and supports the design of city-specific incentives and training.

3. Geographic Analysis

3.1 Revenue Distribution Across Countries and Cities

3.2 Top Cities by Sales Volume

```
SELECT
    ctry.country_name,
    city.city_name,
    SUM(s.total_price) AS total_revenue
FROM sales s
JOIN customers cust ON s.customer_id = cust.customer_id
JOIN cities city ON cust.city_id = city.city_id
JOIN countries ctry ON city.country_id = ctry.country_id
GROUP BY ctry.country_name, city.city_name
ORDER BY total_revenue DESC;
```

ry_name cter varying (100)	city_name character varying (120)	total_revenue numeric
d States	St. Petersburg	166774.35
d States	Philadelphia	157442.27
d States	Aurora	156522.37
d States	Albuquerque	155400.82
d States	Dallas	155382.13

Client Need: Allocate marketing and operational resources geographically.
Tables Used: *sales, customers, cities, countries*
Summary: We aggregate revenue across all countries and cities to reveal which markets contribute most to total sales. This guides investment in coverage, marketing, and inventory.

3.3 Bottom Cities by Sales Revenue

```
SELECT
    city.city_name,
    SUM(s.total_price) AS total_revenue,
    SUM(s.quantity) AS total_quantity
FROM sales s
JOIN customers cust ON s.customer_id = cust.customer_id
JOIN cities city ON cust.city_id = city.city_id
GROUP BY city.city_name
ORDER BY total_revenue DESC
LIMIT 5;
```

name acter varying (120)	total_revenue numeric	total_quantity bigint
etersburg	166774.35	3378
adelphia	157442.27	2994
ra	156522.37	3132
querque	155400.82	3148
as	155382.13	2940

Client Need: Identify high-value markets for targeted strategies.
Tables Used: *sales, customers, cities*
Summary: Cities are ranked by revenue and quantity, distinguishing premium markets from high-volume, low-price regions.

3.4 Top-Grossing Category in Each City

```sql
SELECT
    city.city_name,
    SUM(s.total_price) AS total_revenue,
    SUM(s.quantity) AS total_quantity
FROM sales s
JOIN customers cust ON s.customer_id = cust.customer_id
JOIN cities city ON cust.city_id = city.city_id
GROUP BY city.city_name
ORDER BY total_revenue ASC
LIMIT 5;
```

ıame ıcter varying (120)	total_revenue numeric	total_quantity bigint
and	100115.81	2099
side	100179.26	2114
ha	104208.96	2261
ingham	106156.70	2195
lo	110272.60	2254

Client Need: Spot markets that require improvement—or strategic de-prioritization.
Tables Used: *sales, customers, cities*
Summary: The weakest-performing cities are identified through ascending revenue rankings, highlighting areas where penetration, product fit, or operations may be insufficient.

```sql
WITH city_category_revenue AS (
    SELECT
        city.city_name,
        cat.category_name,
        SUM(s.total_price) AS revenue
    FROM sales s
    JOIN customers cust ON s.customer_id = cust.customer_id
    JOIN cities city ON cust.city_id = city.city_id
    JOIN products p ON s.product_id = p.product_id
    JOIN categories cat ON p.category_id = cat.category_id
    GROUP BY city.city_name, cat.category_name
),
ranked AS (
    SELECT
        city_name,
        category_name,
        revenue,
        RANK() OVER (PARTITION BY city_name ORDER BY revenue DESC) AS rnk
    FROM city_category_revenue
)
SELECT
    city_name,
    category_name AS top_category,
    revenue AS total_revenue
FROM ranked
WHERE rnk = 1
ORDER BY city_name;
```

varying (120)	top_category character varying (100)	total_revenue numeric
	Poultry	21087.67
ue	Cereals	24840.13
	Confections	23321.72
e	Confections	21997.42
	Dairy	14020.53

Client Need: Tailor assortments and marketing to local preferences.
Tables Used: *sales, customers, cities, products, categories*
Summary: Using a city–category revenue CTE and window ranking, we extract each city's top category, showing how preferences differ across markets and enabling geographic segmentation.

Customer Interaction Plan

1. Analyst

Direct SQL Access in PostgreSQL

Analysts will connect to the PostgreSQL environment using a SQL client such as pgAdmin or the Codio interface. Once connected, they can run analytical queries directly against the fully normalized third normal form schema. The separation of customers, products, categories, employees, cities, countries, and sales into distinct but relationally linked tables enables analysts to perform efficient joins, common table expressions, window functions, and aggregate operations. This capability supports a wide range of analytical tasks such as evaluating store profitability, monitoring inventory turnover, identifying category-level trends, and analyzing customer repeat-purchase behavior. Direct SQL access ensures transparency and enables flexible, ad-hoc analysis.

SQL and Python Integration with SQLAlchemy and pandas

For more complex analytical workflows, including automation, scheduled reporting, and statistical or predictive modeling, analysts will interact with the database through Python. With SQLAlchemy and psycopg2, analysts can establish secure connections to PostgreSQL, execute SQL statements programmatically, and load results directly into pandas DataFrames.
This integration combines the efficiency of SQL with the versatility of Python, allowing analysts to build multi-step analyses, transform large datasets, run forecasting models, and prepare data for machine learning applications.

Interaction with the ETL Pipeline

Analysts may also initiate ETL refreshes or reload specific tables as new data becomes available. The loading sequence of countries, cities, customers, employees, categories, products, and sales ensures that referential integrity is always maintained and analysts are consistently working with accurate and complete data. Since the ETL workflow is structured and repeatable, updates can be incorporated without disrupting dashboards or downstream analytical processes.

Performance and Redundancy Considerations

The system includes several measures to ensure strong performance during analytical workloads. Indexing is applied to primary keys, foreign keys, and commonly filtered columns to accelerate query execution. Analysts who import large tables into Python can rely on chunked reads to avoid memory overload, and in-memory DataFrames help reduce repeated database calls during multi-step analysis. PostgreSQL's ACID properties preserve consistency when multiple analysts access the system at the same time. As the data volume grows, the architecture can expand through the use of read replicas, caching layers, or materialized views to support heavier reporting demands.

Dashboard Integration

Analysts will also maintain the SQL queries that power executive dashboards in Metabase. These dashboards provide leaders with intuitive, real-time insights without requiring any technical

knowledge. Analysts ensure that the underlying queries and data models remain accurate and aligned with business priorities.

2. C-Level Officers

Executive Summary Metrics

To support strategic decision making at the leadership level, the Executive Summary panel presents the most important financial indicators based on current performance. The metrics include month-to-date revenue, month-over-month growth, quarter-to-date revenue, quarter-over-quarter growth, year-to-date revenue, and year-over-year growth. These measures allow executives to evaluate performance across multiple time horizons and to understand whether the business is accelerating or slowing down. By emphasizing real-time revenue trends rather than historical aggregates, the Executive Summary enables leadership to monitor progress, assess the impact of strategic initiatives, and determine whether the organization is on track to meet monthly, quarterly, and annual targets.

Order and Customer Behavior Insights

In addition to revenue metrics, executives benefit from a clear view of customer demand and transaction behavior. The dashboard therefore includes month-to-date total orders, order growth compared to the previous month, and changes in average order value. Together, these indicators illustrate whether customer demand is increasing or decreasing, whether average transaction size is shifting, and how marketing activities, pricing strategies, or operational changes are influencing purchasing patterns. These insights help leadership interpret revenue fluctuations with greater context and support decisions in pricing, product bundling, marketing investment, and inventory planning.

Product Portfolio Performance

To provide meaningful product insights without overwhelming users with SKU-level detail, the product analysis panel focuses on aggregated and high-impact performance measures. This section displays the top ten and bottom ten products by monthly revenue, total revenue by product category, and category lifecycle trends. The lifecycle view captures how categories evolve over time by showing monthly revenue trajectories, quarterly growth, peak-season months, and broader seasonal behavior. These insights help leadership identify which product segments are expanding, stabilizing, or declining. This supports decisions related to assortment planning, product development, supplier negotiation, and long-term investment across the product portfolio.

Geographic Insights

Since market performance varies across regions, the dashboard includes a geographic insights panel that highlights regional opportunities and risks. This section presents monthly revenue and

order volume across major United States regions, as well as customer distribution and category-level revenue differences by location. These geographic metrics help executives identify high-value markets, underpenetrated regions, and shifts in regional product demand. The insights directly support decisions regarding marketing strategy, regional expansion, logistics planning, and resource allocation. All geographic views update automatically through Metabase to ensure accuracy and timeliness.

Metabase Dashboards

The three KPI cards at the top display business scale metrics from the analysis period which show $4.3B total sales and 98K customers and 6.8M transactions. The fact table (sales) receives direct executive-level metric conversion through these numbers which prove our database operates at a practical level instead of being a simplified model (Figure 4).

The Sales by Category bar chart demonstrates that revenue distribution shows that few categories produce most of the business revenue. The revenue from Confections and Meat and Poultry segments exceeds all other product categories including Shellfish and Grain. The modeling decision for product categories as a fundamental dimension proved correct because these essential categories need priority attention for purchasing and shelf placement and promotional activities while lesser categories should focus on supporting roles.

Sales by City and Top 20 Cities by Revenue charts show that revenue is relatively evenly distributed across cities rather than being concentrated in just a few locations. While some cities perform slightly better than others, the gap between the top and bottom groups is modest. This suggests that geographic risk is fairly diversified and that many markets contribute meaningfully to total sales. The company should focus on monitoring city-level performance, identifying best practices from stronger cities, and selectively running local initiatives in markets with the largest upside rather than relying on a small set of “hero” cities.

The Sales Over Time (Monthly) chart displays total monthly revenue while Monthly Revenue by Category shows how different categories perform throughout the month. The business maintains high revenue levels throughout the first months of 2018 before showing a slight decline in May. The category-level chart demonstrates that the product mix between categories remains stable throughout the analyzed period without any major seasonal fluctuations. The business benefits from continuous price and promotion optimization because it maintains a balanced product range that does not depend on single seasonal events.

The Salesperson Revenue Ranking (in Millions) chart enables users to view which employees generate the highest revenue totals. The analysis remains basic at this stage because the client

plans to link our database to performance-based reward systems and performance evaluation systems in future development.

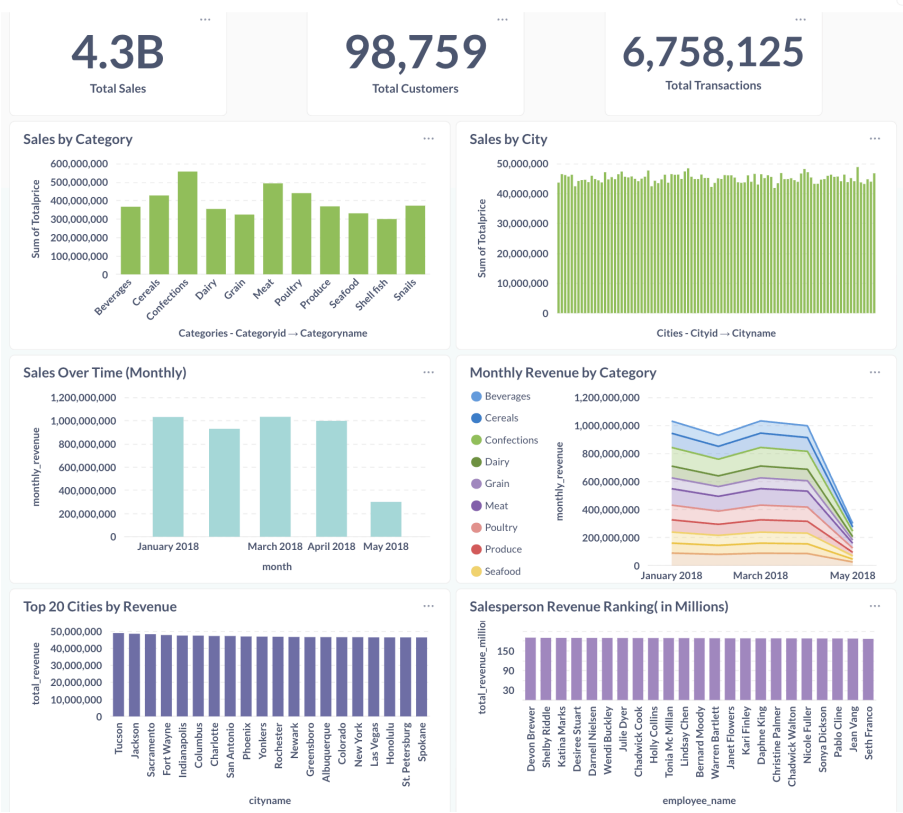


Figure 4: Executive Performance Overview

The Top Products per Category report together with Total Sales by Category provides dual insights about which categories produce the most revenue and which specific SKUs drive performance within each category. Each major product category contains multiple leading products which drive business success. The essential products of ABC Foodmart need protection during assortment planning and supplier negotiations and inventory management and store expansion initiatives.

The Products Most Impacted by Discount (Top 20) report evaluates promotion effectiveness through the calculated increase in average quantity sold during discounted periods. The products react differently to discount promotions because some show substantial sales growth while others remain unresponsive. The business should use price discounts to drive growth for products that show high elasticity but should avoid deep discounts for products with low elasticity because it will damage profit margins. The company should move away from universal discount promotions because specific product SKUs respond better to price reductions (Figure 5).

The 20 Cities by Revenue report identifies the cities which generate the lowest revenue. The company needs to handle these cities under different strategies because some present growth potential that needs marketing support and local funding but others face structural barriers that require conservative revenue projections (Figure 6). The Top Category per City report demonstrates that Confections stands as the leading revenue-generating category across most cities. The company should use this category as its foundation for national branding initiatives while allowing local adjustments to other product categories.The Top-Grossing Employee per City section reveals which salesperson generates the most revenue in each city. Management can use local champions as reference examples to implement best practices including training programs and staffing models and store operations across different locations.

The dashboards present our database and SQL work through a unified story that demonstrates how the company operates at different scales and with various structural elements and market potential. The dashboards reveal product and category value generation points through specific product lines and market areas and team performance which enables ABC Foodmart to create actionable plans for category improvement and promotional activities and regional business strategies.

Figure 5: Product Performance and Market Insights

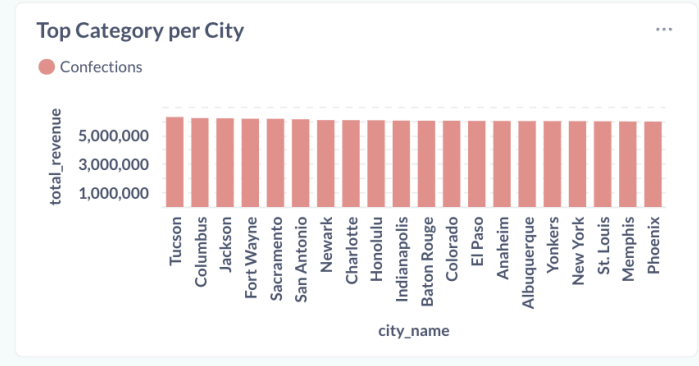


Figure 6: Revenue by City

Conclusion

Our goal in this project was to design a scalable relational database system that replaces ABC Foodmart's manual, inconsistent data practices with a centralized, reliable foundation for decision-making. Through our 3NF PostgreSQL schema, the Python-based ETL pipeline, and the analytical procedures developed across product, customer, salesperson, and geographic dimensions, we transformed fragmented CSV files into an integrated system that supports both operational reporting and advanced analytics. The ETL workflow ensures data accuracy, consistency, and repeatability, while the relational model eliminates redundancy and enables fast, flexible querying. Our SQL analyses uncovered meaningful insights such as category seasonality, discount responsiveness, regional performance differences, and employee productivity patterns, which include insights that were previously impossible under the company's spreadsheet-based workflow. Finally, the executive dashboards in Metabase give leaders real-time visibility into revenue, product mix, and market dynamics without requiring technical expertise. Together, the RDBMS design, ETL pipeline, analytics, and dashboards provide ABC Foodmart with a modern data infrastructure that supports strategic planning, operational efficiency, and long-term growth.

Made with Metabase

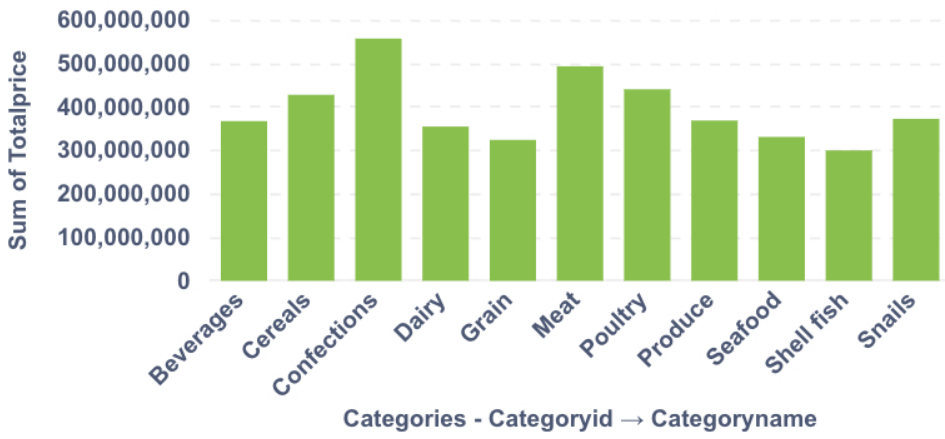
SQL Project Dashboard

4.3B
Total Sales

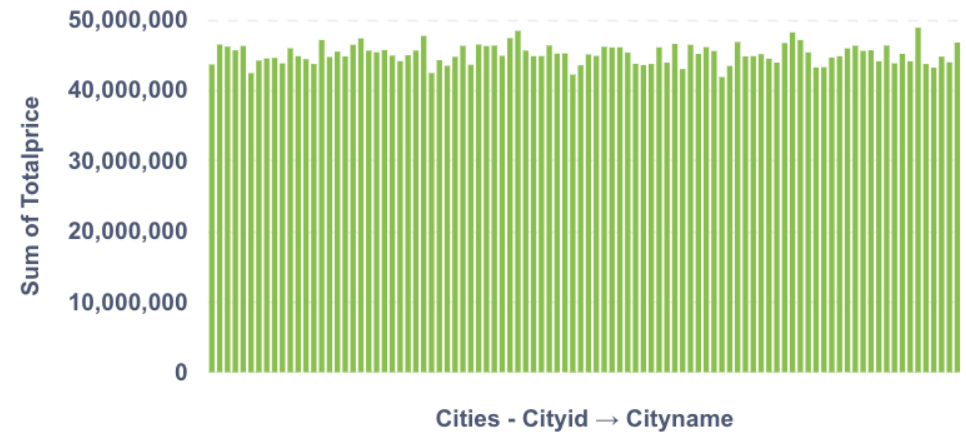
98,759
Total Customers

6,758,125
Total Transactions

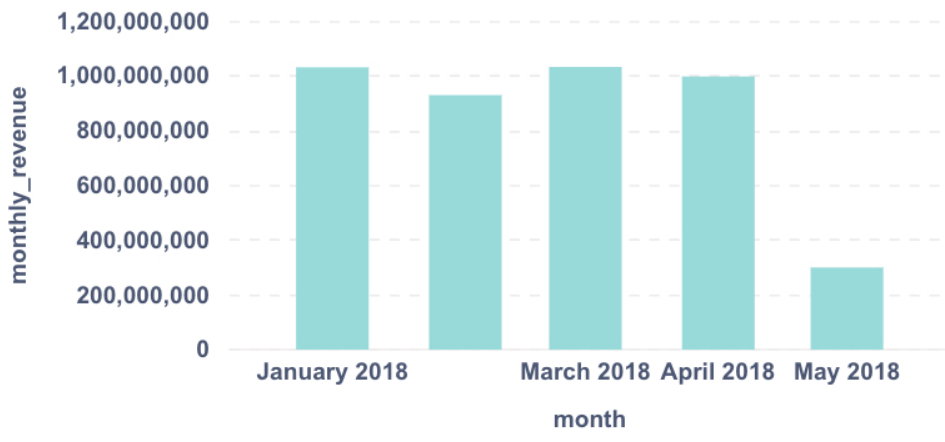
Sales by Category



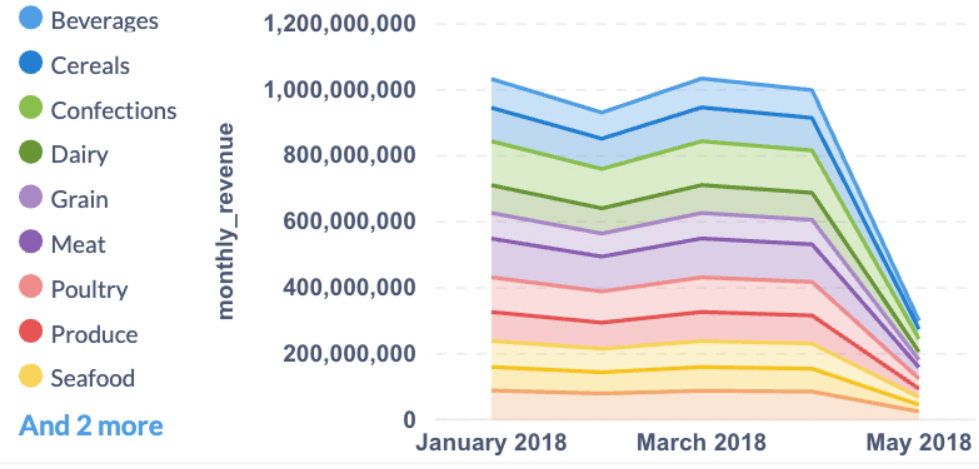
Sales by City



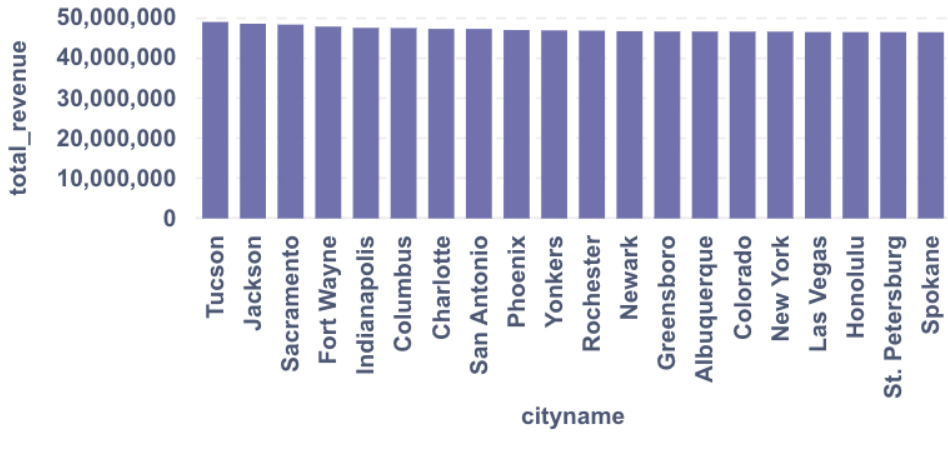
Sales Over Time (Monthly)



Monthly Revenue by Category



Top 20 Cities by Revenue



Salesperson Revenue Ranking(in Millions)

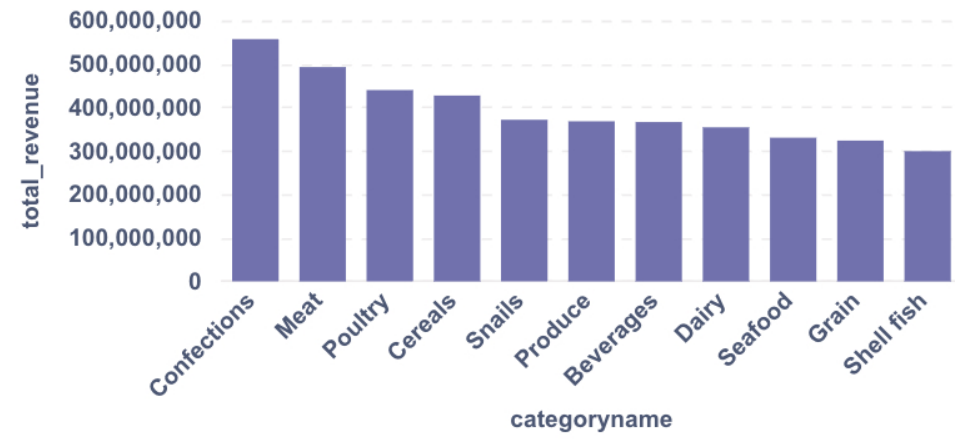
SQL Project Dashboard

Top Products per Category

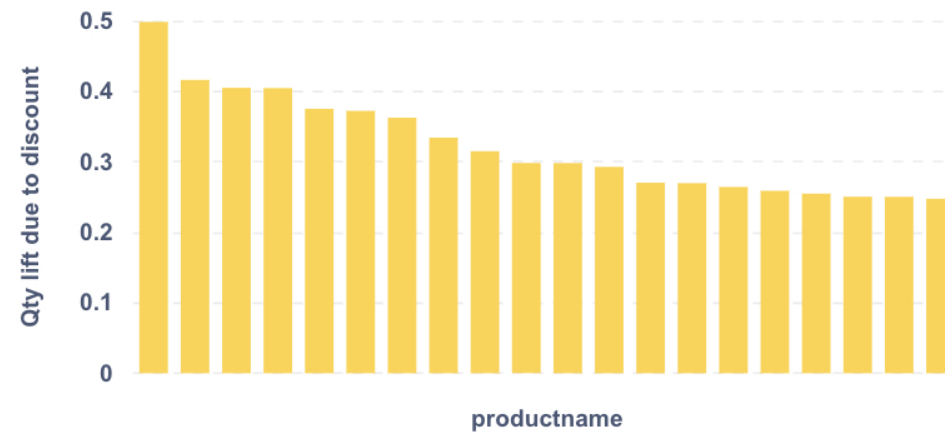
categoryname	top_product	total_revenue
Dairy	Bread - Calabrese Baguette	18,869,259.48
Cereals	Shrimp - 31/40	18,722,785.66
Beverages	Puree - Passion Fruit	18,704,180.36
Snails	Zucchini - Yellow	18,550,828.91
Poultry	Vanilla Beans	18,529,731.7
Meat	Beef - Inside Round	18,335,107.65
Grain	Grenadine	18,331,720.09
Seafood	Tuna - Salad Premix	18,216,317.34

11 rows

Total Sales by Category



Products Most Impacted by Discount (Top 20)



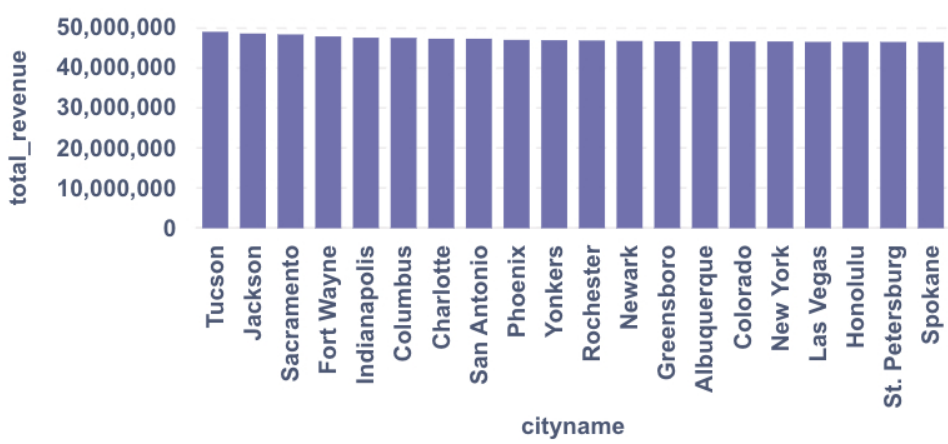
Top-Grossing Employee per City

city_name	employee_name	total_revenue
Tucson	Shelby Riddle	2,213,556.51
Sacramento	Desiree Stuart	2,204,439.57
Jackson	Jean Vang	2,197,194.37
San Antonio	Sonya Dickson	2,189,629.13
Phoenix	Devon Brewer	2,185,123.14
Fort Wayne	Sonya Dickson	2,176,930.3
Glendale	Desiree Stuart	2,158,205.45
Indianapolis	Lindsay Chen	2,153,604.79

96 rows

Top Category per City

Top 20 Cities by Revenue



Bottom 20 Cities by Revenue