ABC Foodmart

Group 11

By: Mingce Bi, Charles Jester, Benno Zhang,
Aminah Al-Jaber

GitHub Link:
https://github.com/MingceBi/APAN5310_Group11_Document.git

Overview

Objectives	03
Database Design	04
ETL Process	06
Analytical Process	09
Dashboard Demo	

Objectives

To improve data quality by designing a database for ABC Foodmart that optimizes storage, while increasing integrity and security of corporate data. All while allowing for it's consumers to have a user friendly experience and take a way important insights.

Scalability

Using one system for all ABC Foodmart locations

Consistency

Being able to make important business decisions based off consistent data

User Friendly

Providing straightforward and intelligible insights

Our E-R diagram is organized into two main groupings of relations.

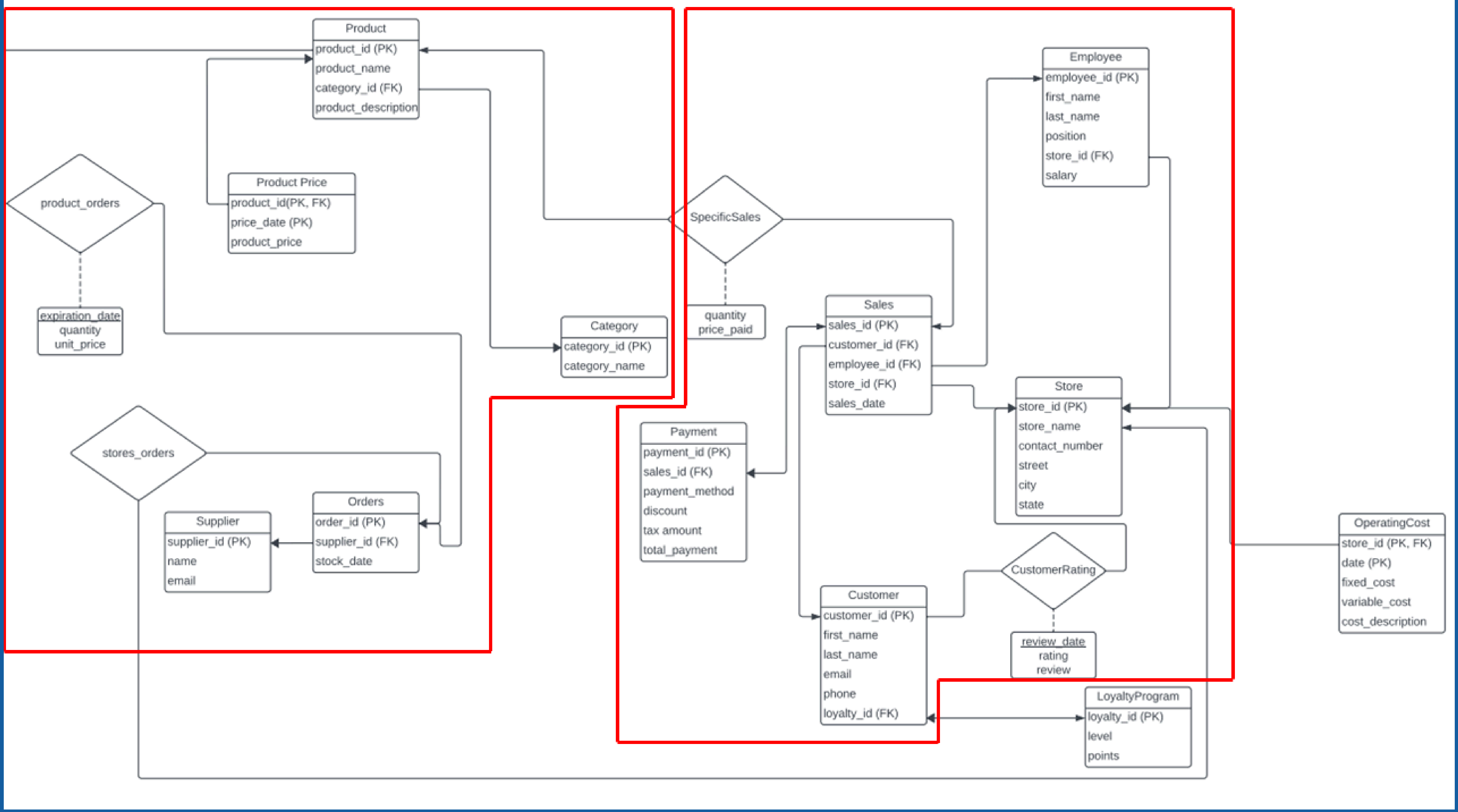
Region 1 tables are related to products, vendors, and inventory. In region 2, we have tables related to employee and customer records, sales and payments, and store operating costs.

The two relations connecting these regions are store orders, which maps inventory orders to stores, and specific sales, which breaks down sales transactions by product.

The organization of our diagram is meant to facilitate the generation of insights in the areas enumerated on the right.

1. Documentation system on revenue and operation cost
2. Sales Analytics: peak sales periods, sales amount, total revenue gained, major product
3. Vendor profiles (vendor names, contact information, types of products supplied)
4. Inventory data keep track on the products ordered from suppliers
5. Customer Loyalty Tracking: Keep track of customer loyalty program, customer purchases
6. Customer rating and/or reviews on social media for each store
7. Store data on staffing, with details on salary, position

E-R Diagram



Product
product_id (PK)
product_name
category_id (FK)
product_description

Product Price
product_id(PK, FK)
price_date (PK)
product_price

Category
category_id (PK)
category_name

Orders
order_id (PK)
supplier_id (FK)
stock_date

Supplier
supplier_id (PK)
name
email

Employee
employee_id (PK)
first_name
last_name
position
store_id (FK)
salary

Sales
sales_id (PK)
customer_id (FK)
employee_id (FK)
store_id (FK)
sales_date

Payment
payment_id (PK)
sales_id (FK)
payment_method
discount
tax amount
total_payment

Store
store_id (PK)
store_name
contact_number
street
city
state

Customer
customer_id (PK)
first_name
last_name
email
phone
loyalty_id (FK)

LoyaltyProgram
loyalty_id (PK)
level
points

OperatingCost
store_id (PK, FK)
date (PK)
fixed_cost
variable_cost
cost_description

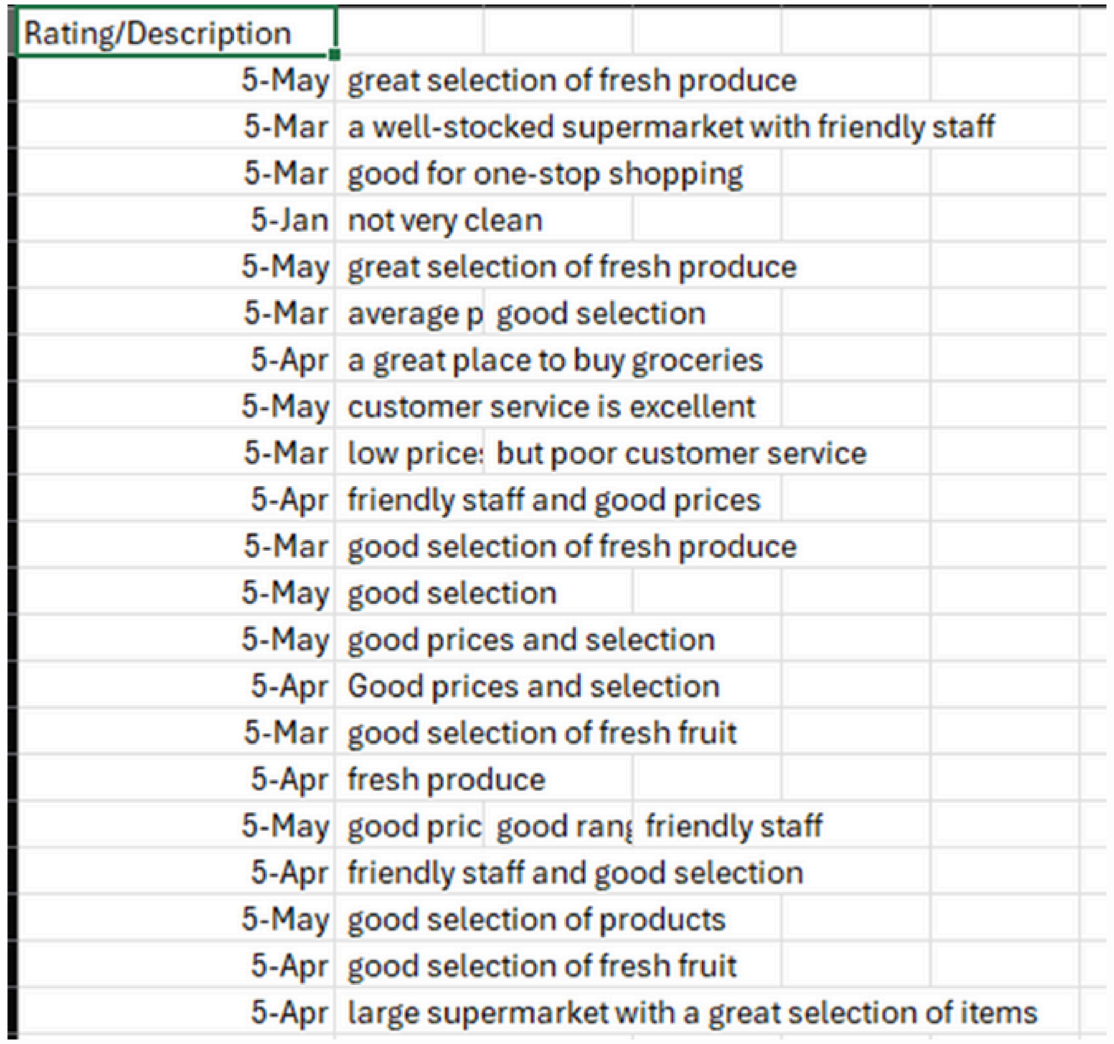
Extract

Datasets: 5 datasets

we combined five datasets:

- customers_sales.csv
- employees_shifts.csv
- products_vendor_orders.csv
- stores.csv
- sample_reviews.csv

We added sample_review that includes the rating and review values for relationship set (CustomerRatings) between Customer and Store entity sets



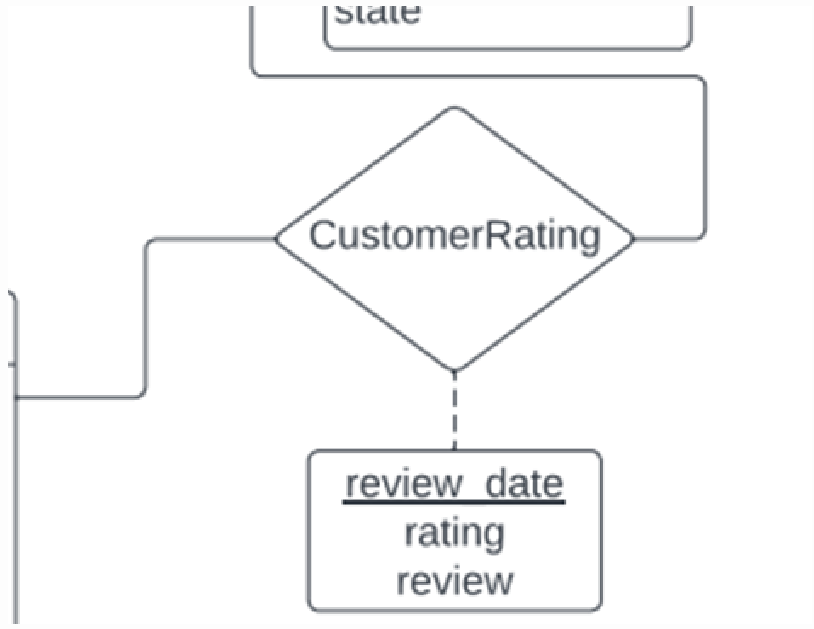
Rating/Description	
5-May	great selection of fresh produce
5-Mar	a well-stocked supermarket with friendly staff
5-Mar	good for one-stop shopping
5-Jan	not very clean
5-May	great selection of fresh produce
5-Mar	average p good selection
5-Apr	a great place to buy groceries
5-May	customer service is excellent
5-Mar	low price: but poor customer service
5-Apr	friendly staff and good prices
5-Mar	good selection of fresh produce
5-May	good selection
5-May	good prices and selection
5-Apr	Good prices and selection
5-Mar	good selection of fresh fruit
5-Apr	fresh produce
5-May	good pric good rang friendly staff
5-Apr	friendly staff and good selection
5-May	good selection of products
5-Apr	good selection of fresh fruit
5-Apr	large supermarket with a great selection of items

sample_reviews.csv (left)

- Rating in range 1-5
- Description (review)

CustomerRating relationship set (right)

- review_date
- rating
- review



Transform

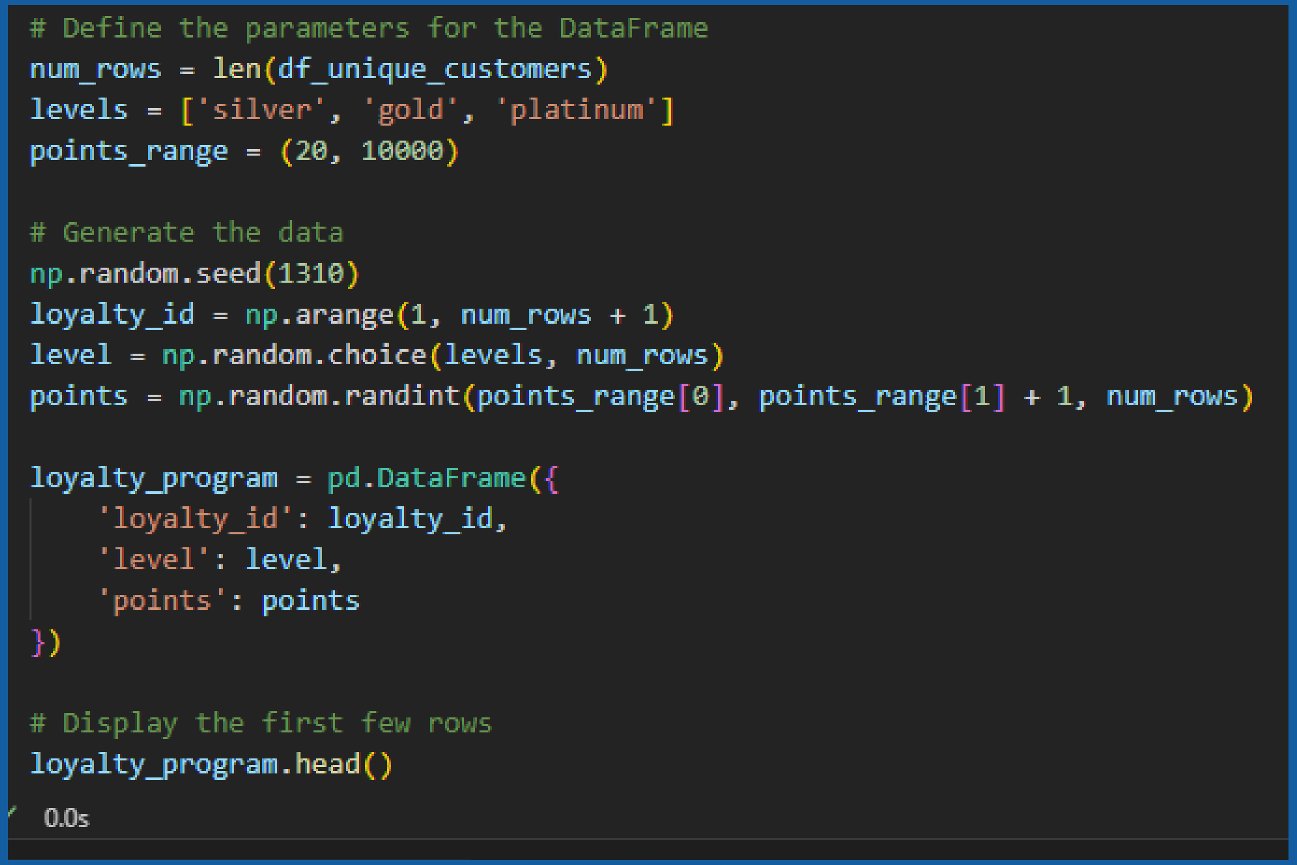
Create Two Dataframes

Grouped transformation tasks into two general parts:

- Combine datasets to create two dataframes
- Dataframes include variables matching schema attributes (right)

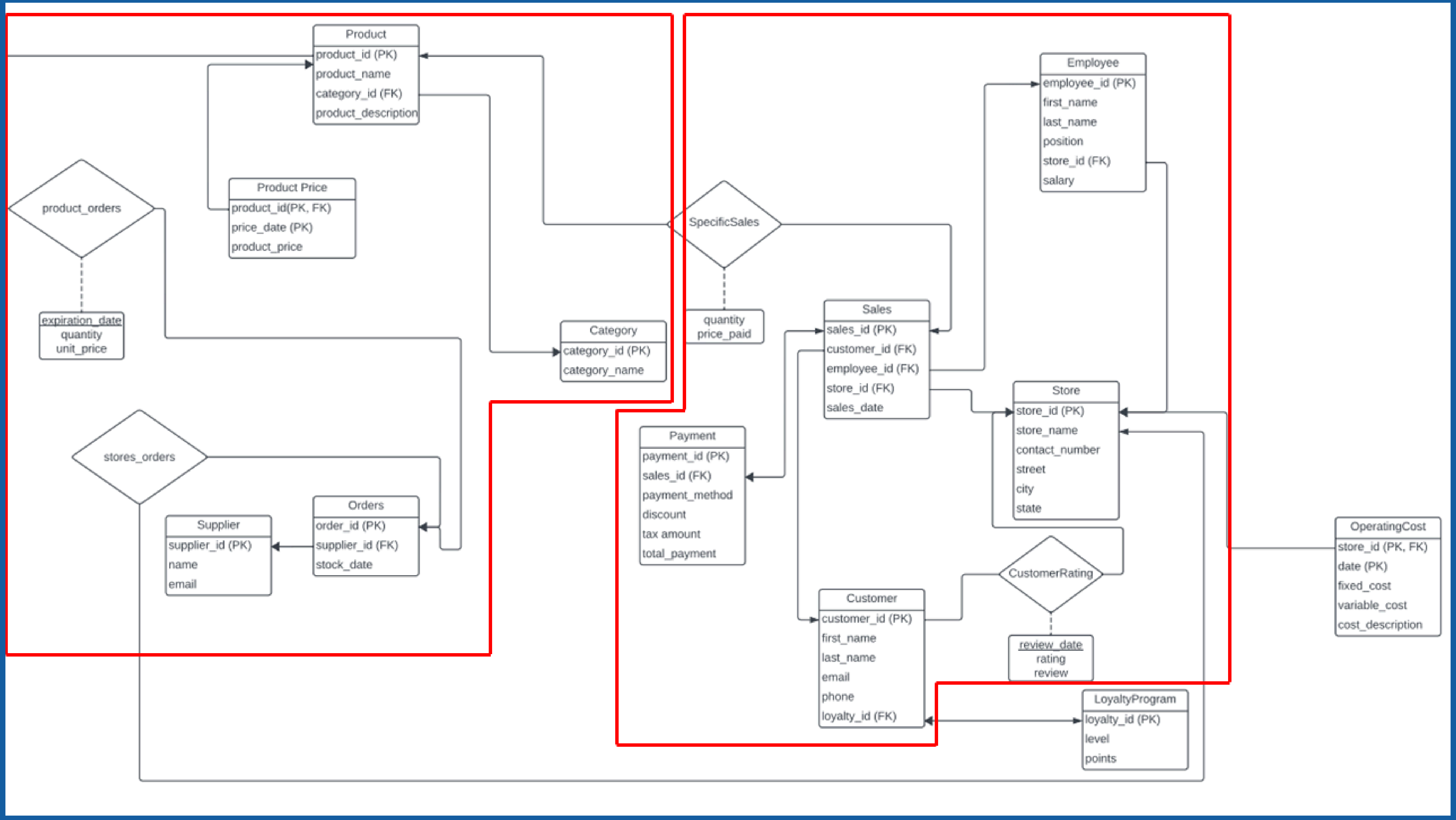
Benefits:

- Maintain relationship constraints between schemas
- Example: product_order relationship



```python
# Define the parameters for the DataFrame
num_rows = len(df_unique_customers)
levels = ['silver', 'gold', 'platinum']
points_range = (20, 10000)

# Generate the data
np.random.seed(1310)
loyalty_id = np.arange(1, num_rows + 1)
level = np.random.choice(levels, num_rows)
points = np.random.randint(points_range[0], points_range[1] + 1, num_rows)

loyalty_program = pd.DataFrame({
    'loyalty_id': loyalty_id,
    'level': level,
    'points': points
})

# Display the first few rows
loyalty_program.head()
```

✓ 0.0s



For Other Schemas

Generate dataframes for the other schemas:

- LoyaltyProgram (left)
- OperatingCost

Load

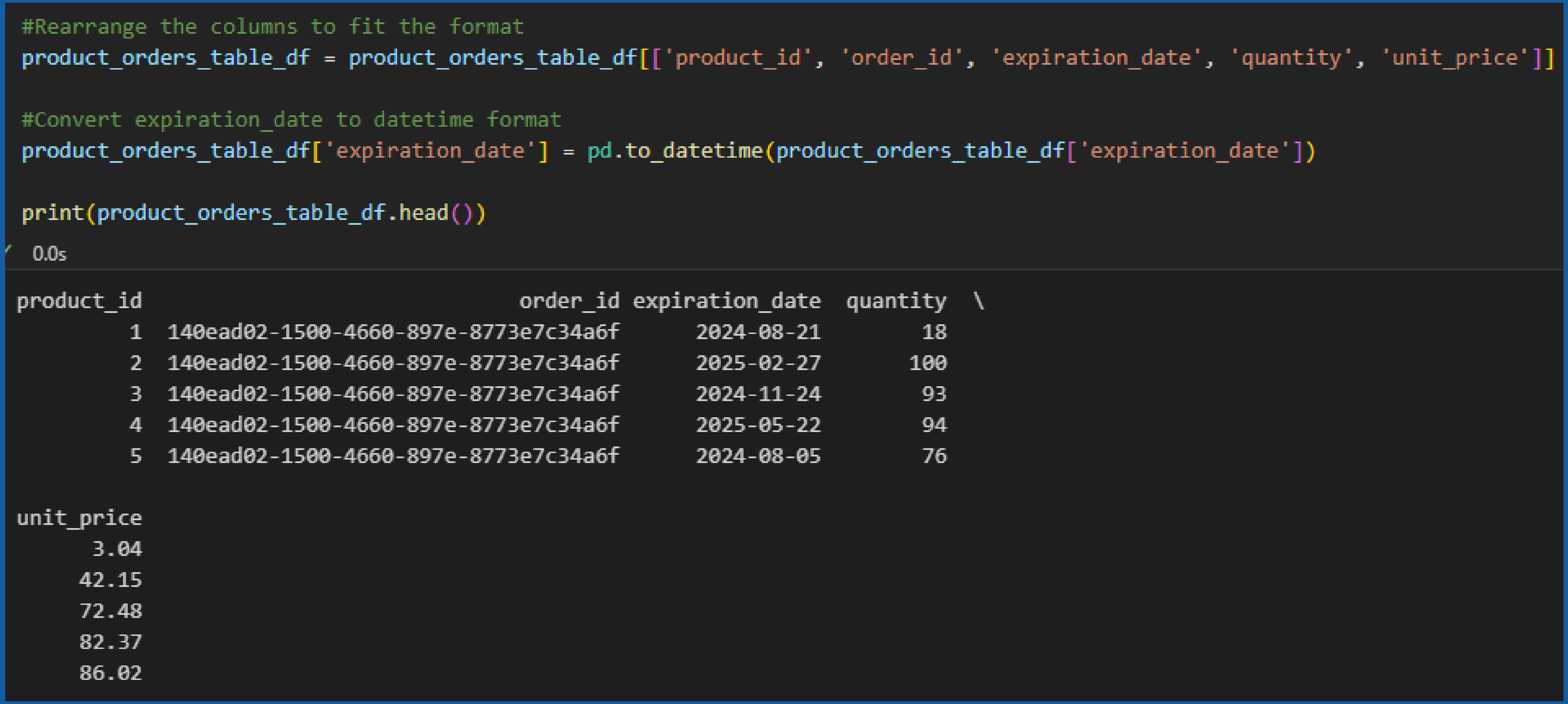
Important Considerations

1.The variable types and length should match the database schema design.

- Transform the date into the right DATETIME format for the schema

2.Make sure the sequence of the variables in the dataframe is matching the schema attributes sequennce.

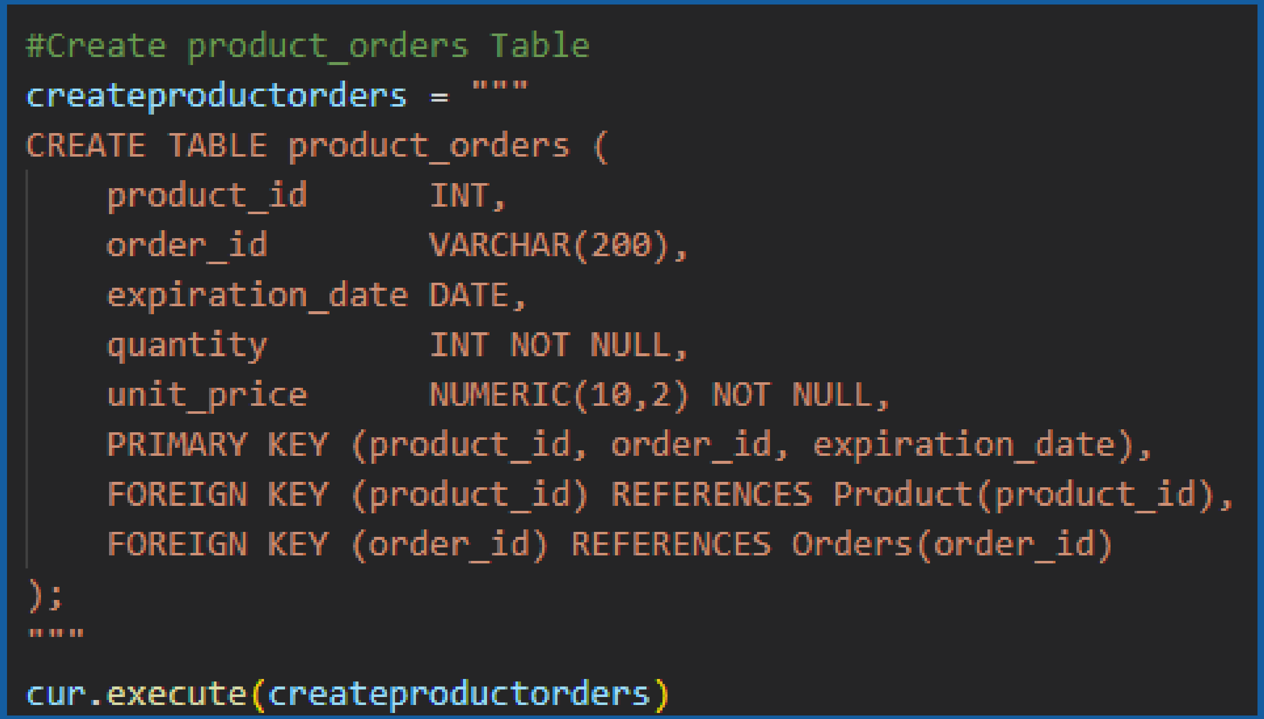
- Rearrange the column sequence before data insertion



```
#Rearrange the columns to fit the format
product_orders_table_df = product_orders_table_df[['product_id', 'order_id', 'expiration_date', 'quantity', 'unit_price']]

#Convert expiration_date to datetime format
product_orders_table_df['expiration_date'] = pd.to_datetime(product_orders_table_df['expiration_date'])

print(product_orders_table_df.head())
```

0.0s

```
product_id                              order_id expiration_date  quantity  \
         1  140ead02-1500-4660-897e-8773e7c34a6f      2024-08-21        18
         2  140ead02-1500-4660-897e-8773e7c34a6f      2025-02-27       100
         3  140ead02-1500-4660-897e-8773e7c34a6f      2024-11-24        93
         4  140ead02-1500-4660-897e-8773e7c34a6f      2025-05-22        94
         5  140ead02-1500-4660-897e-8773e7c34a6f      2024-08-05        76

unit_price
      3.04
     42.15
     72.48
     82.37
     86.02
```

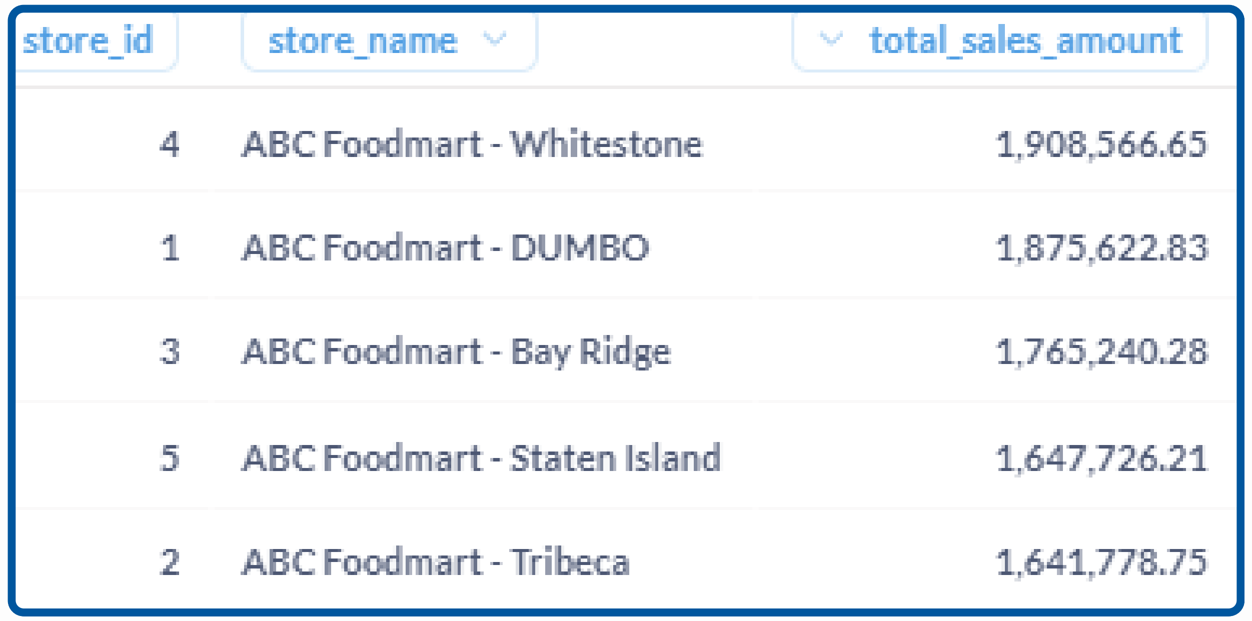


```
#Create product_orders Table
createproductorders = """
CREATE TABLE product_orders (
    product_id      INT,
    order_id        VARCHAR(200),
    expiration_date DATE,
    quantity        INT NOT NULL,
    unit_price      NUMERIC(10,2) NOT NULL,
    PRIMARY KEY (product_id, order_id, expiration_date),
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
"""
cur.execute(createproductorders)
```

TARGET AUDIENCE

Data Analysts

Use **PgAdmin** to directly query and retrieve results in tabular data

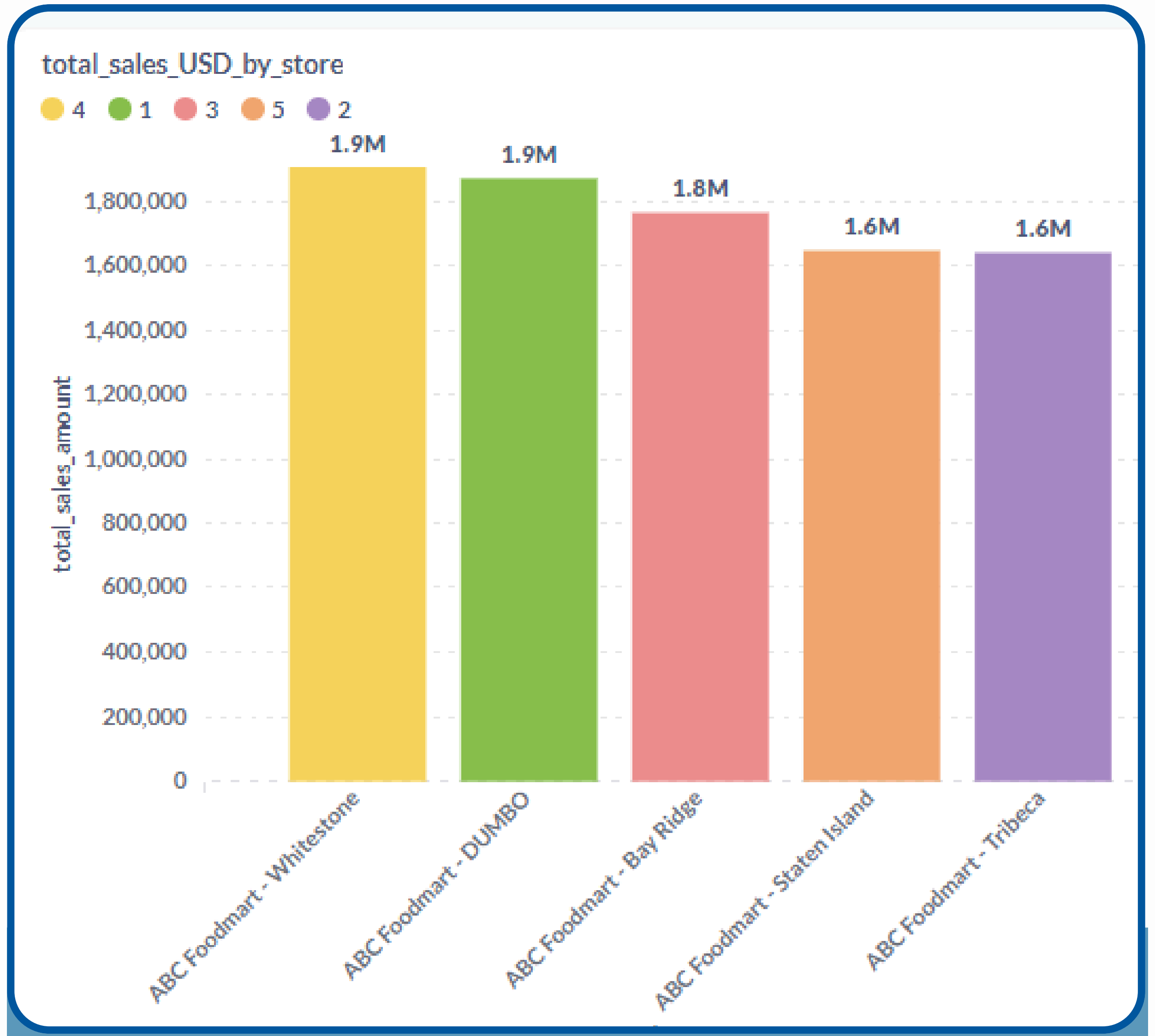


store_id	store_name	total_sales_amount
4	ABC Foodmart - Whitestone	1,908,566.65
1	ABC Foodmart - DUMBO	1,875,622.83
3	ABC Foodmart - Bay Ridge	1,765,240.28
5	ABC Foodmart - Staten Island	1,647,726.21
2	ABC Foodmart - Tribeca	1,641,778.75

Question: What is each store's total sales?

C-level Officers

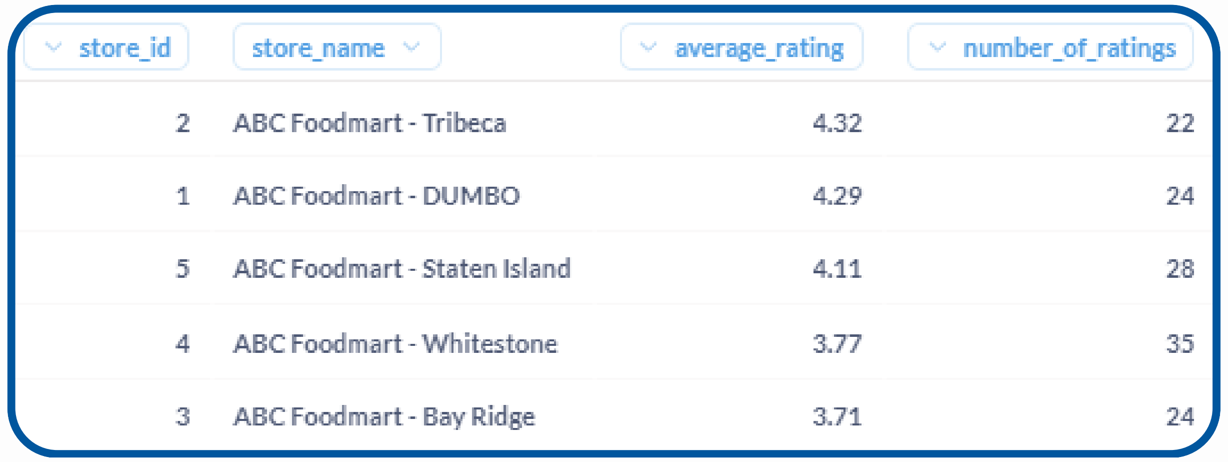
Use **Metabase** Dashboards to help understand quickly what the key insights are



Insights

We can interact with the database to acquire insights

Rating per Store

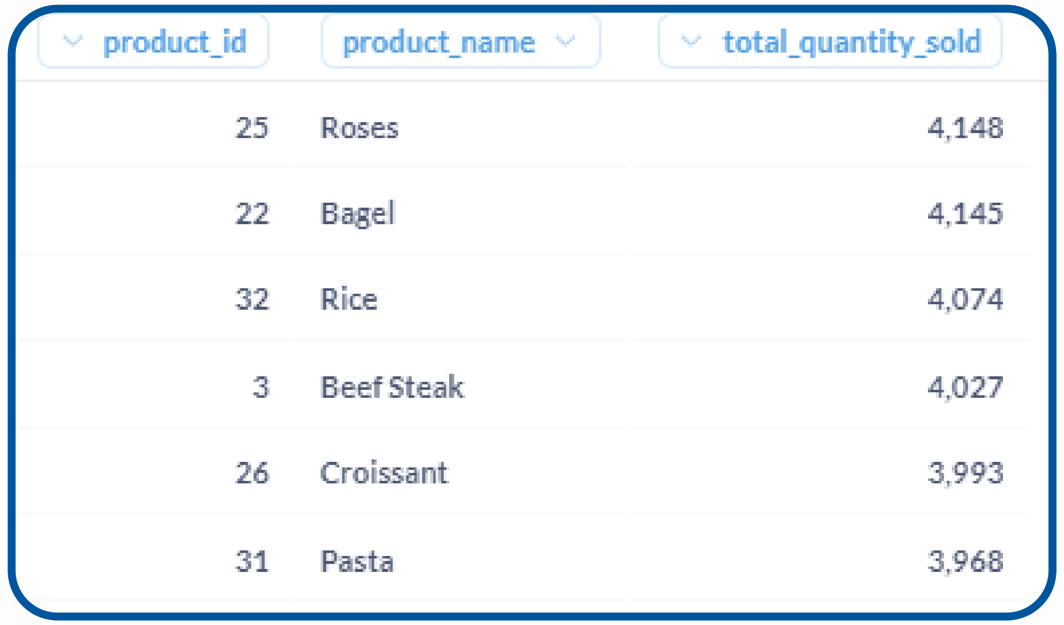


store_id	store_name	average_rating	number_of_ratings
2	ABC Foodmart - Tribeca	4.32	22
1	ABC Foodmart - DUMBO	4.29	24
5	ABC Foodmart - Staten Island	4.11	28
4	ABC Foodmart - Whitestone	3.77	35
3	ABC Foodmart - Bay Ridge	3.71	24

SQL:

```
SELECT
  Store.store_id, Store.store_name,
  AVG(CustomerRating.rating) AS average_rating,
  COUNT(CustomerRating.rating) AS number_of_ratings
FROM CustomerRating JOIN Store ON
  CustomerRating.store_id = Store.store_id
GROUP BY
  Store.store_id, Store.store_name
ORDER BY
  average_rating DESC;
```

Top Items Sold

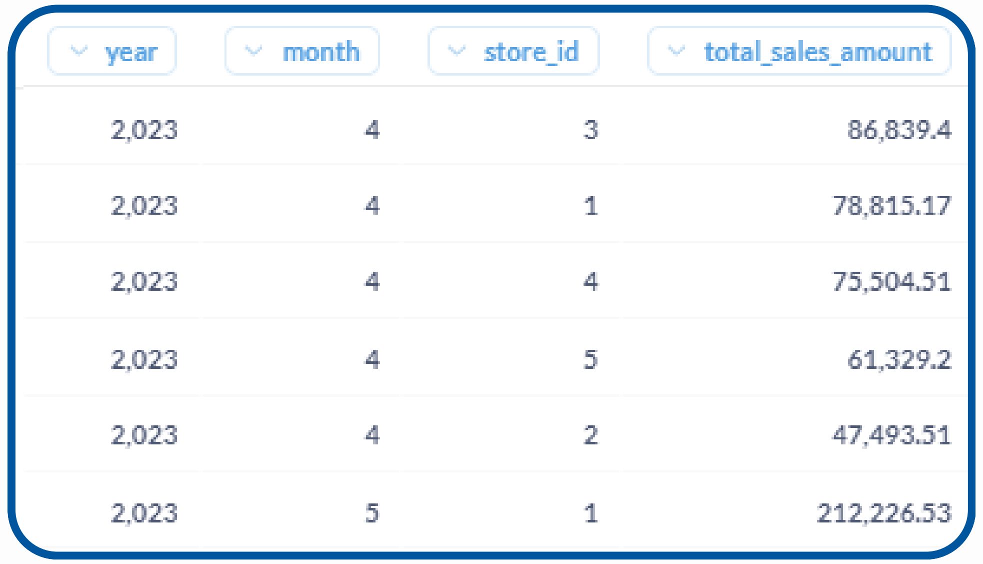


product_id	product_name	total_quantity_sold
25	Roses	4,148
22	Bagel	4,145
32	Rice	4,074
3	Beef Steak	4,027
26	Croissant	3,993
31	Pasta	3,968

SQL:

```
SELECT
  Product.product_id, Product.product_name,
  SUM(SpecificSales.quantity) AS total_quantity_sold
FROM SpecificSales JOIN Product ON
  SpecificSales.product_id = Product.product_id
GROUP BY Product.product_id, Product.product_name
ORDER BY total_quantity_sold DESC;
```

Revenue by Month



year	month	store_id	total_sales_amount
2,023	4	3	86,839.4
2,023	4	1	78,815.17
2,023	4	4	75,504.51
2,023	4	5	61,329.2
2,023	4	2	47,493.51
2,023	5	1	212,226.53

SQL:

```
SELECT
  EXTRACT(YEAR FROM Sales.sales_date) AS year,
  EXTRACT(MONTH FROM Sales.sales_date) AS month,
  Sales.store_id,
  SUM(Payment.total_payment) AS total_sales_amount
FROM
  Sales JOIN Payment ON Sales.sales_id = Payment.sales_id
GROUP BY year, month, Sales.store_id
ORDER BY year, month, total_sales_amount DESC;
```