2025.08.13

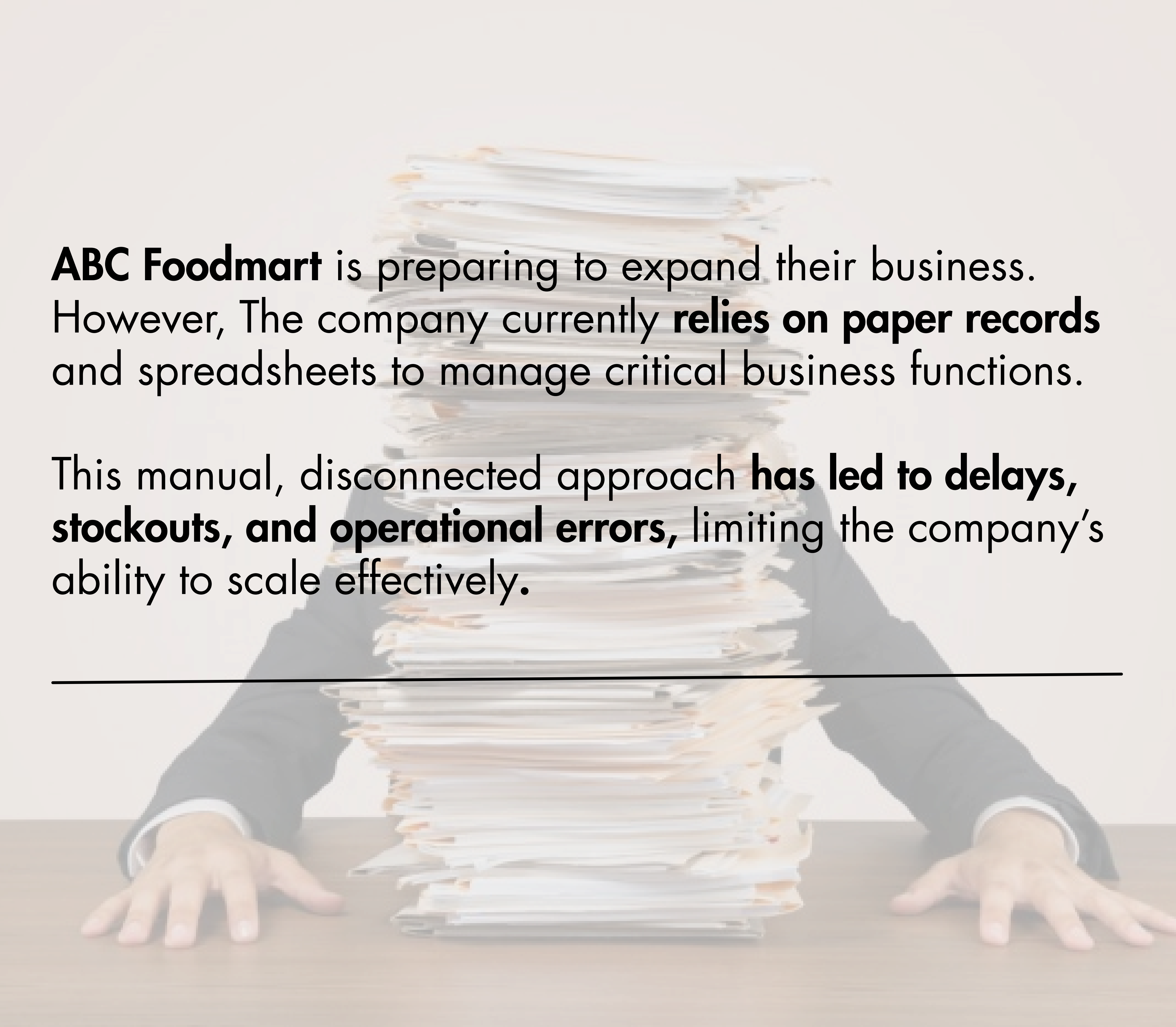
ABC Foodmart Database Implementation

Group 7
Minkyung Kim, Boni Vasius Rosen,
Theodore Zaphiris, Peter Zhang

Problem Statement

ABC Foodmart is preparing to expand their business. However, The company currently **relies on paper records** and spreadsheets to manage critical business functions.

This manual, disconnected approach **has led to delays, stockouts, and operational errors,** limiting the company's ability to scale effectively**.**



Solution Offered

Our team will design and implement a centralized relational database system. The solution will meet ABC Foodmart’s specific **business requirements in the following areas**:

Core Operation

Staffing & HR

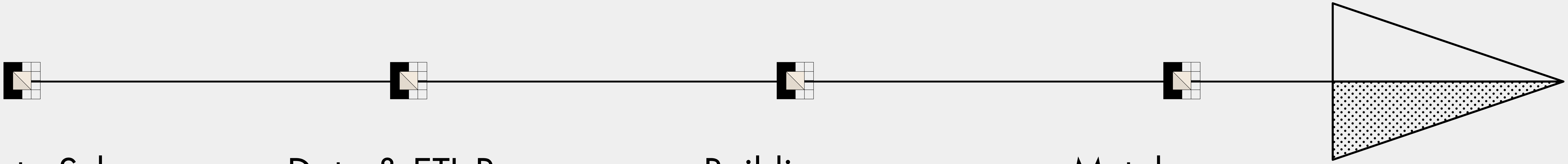
Product & Inventory Management

Vendors & Deliveries

Sales & Financial

Manager Reporting & Insights

Project Overview

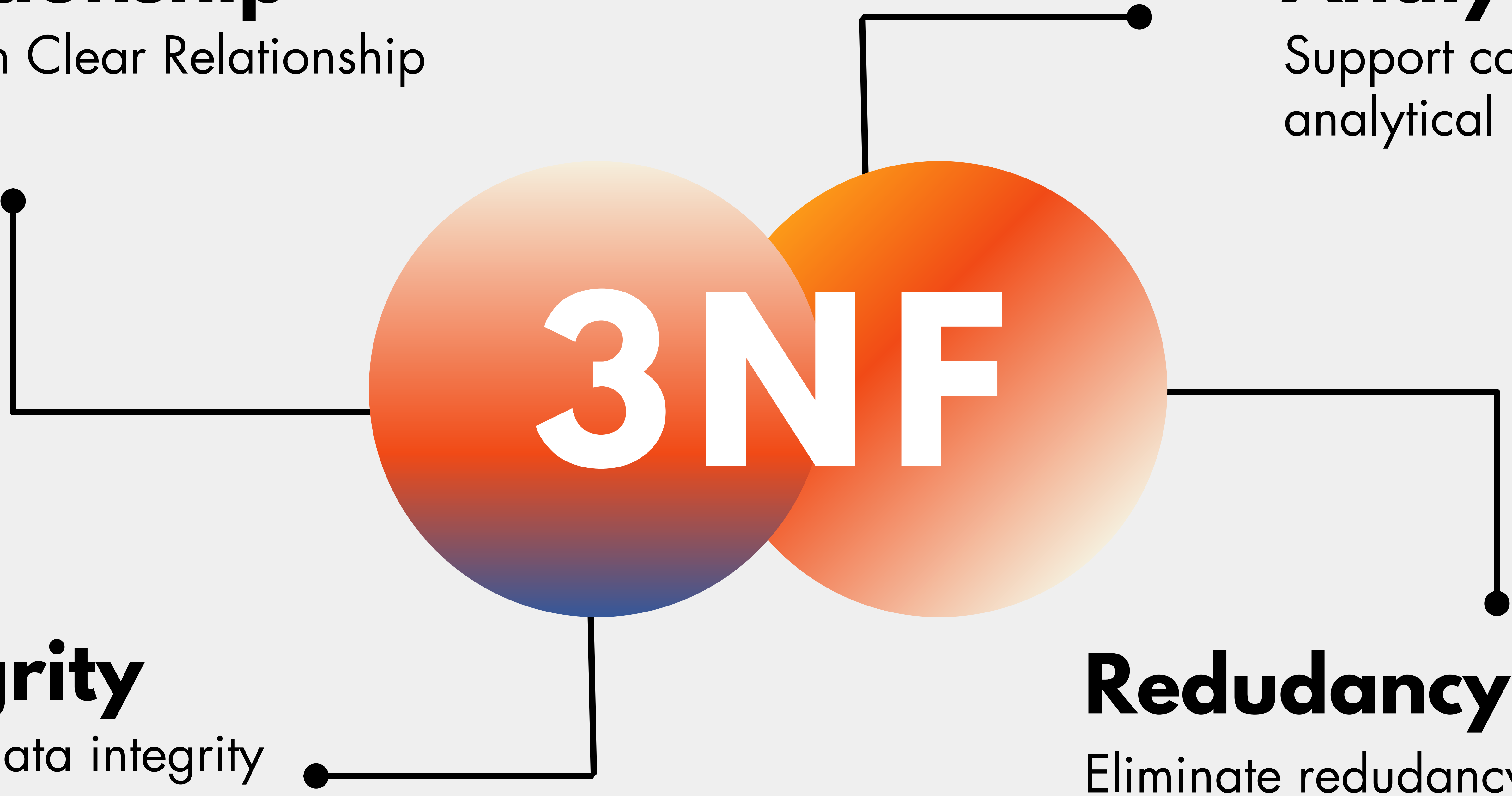


Data Scheme Design

Data & ETL Process

Building Complex Queries

Metabase Dashboard



Relationship

Maintain Clear Relationship

Analytical

Support complex analytical queries

Integrity

Ensure data integrity

Redudancy

Eliminate redudancy

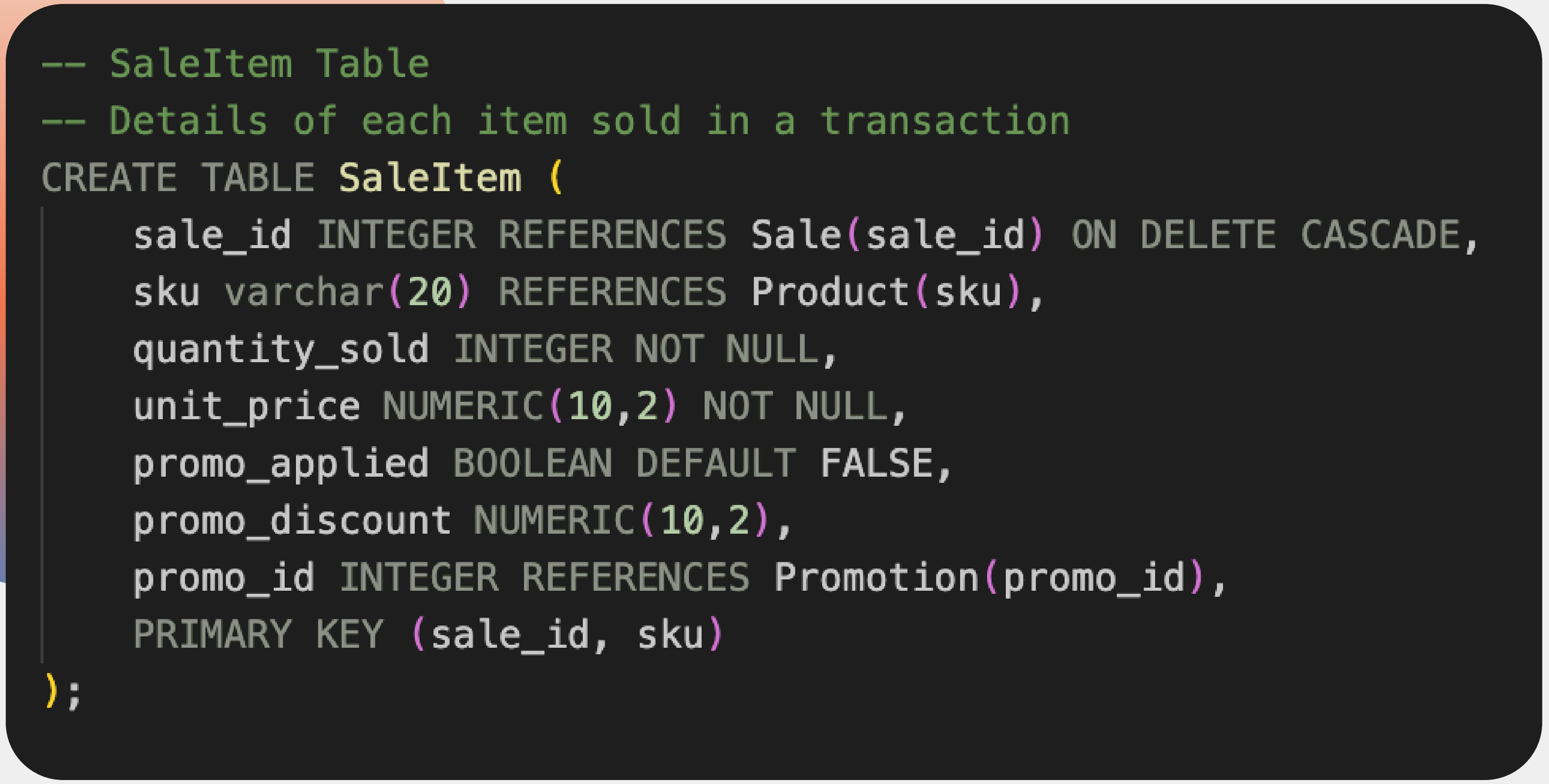
Database Scheme Design

- The ABC Foodmart database consist of **18 tables** which grouped by **4 business domains (Operation, Sales, Products, and Supply Chain)**

- This design includes **4 trigger functions** (restock, deduct, return, delivery) designed to keep inventory levels and status in sync automatically for our client and performance efficiency.

Schema Design - SaleItem table

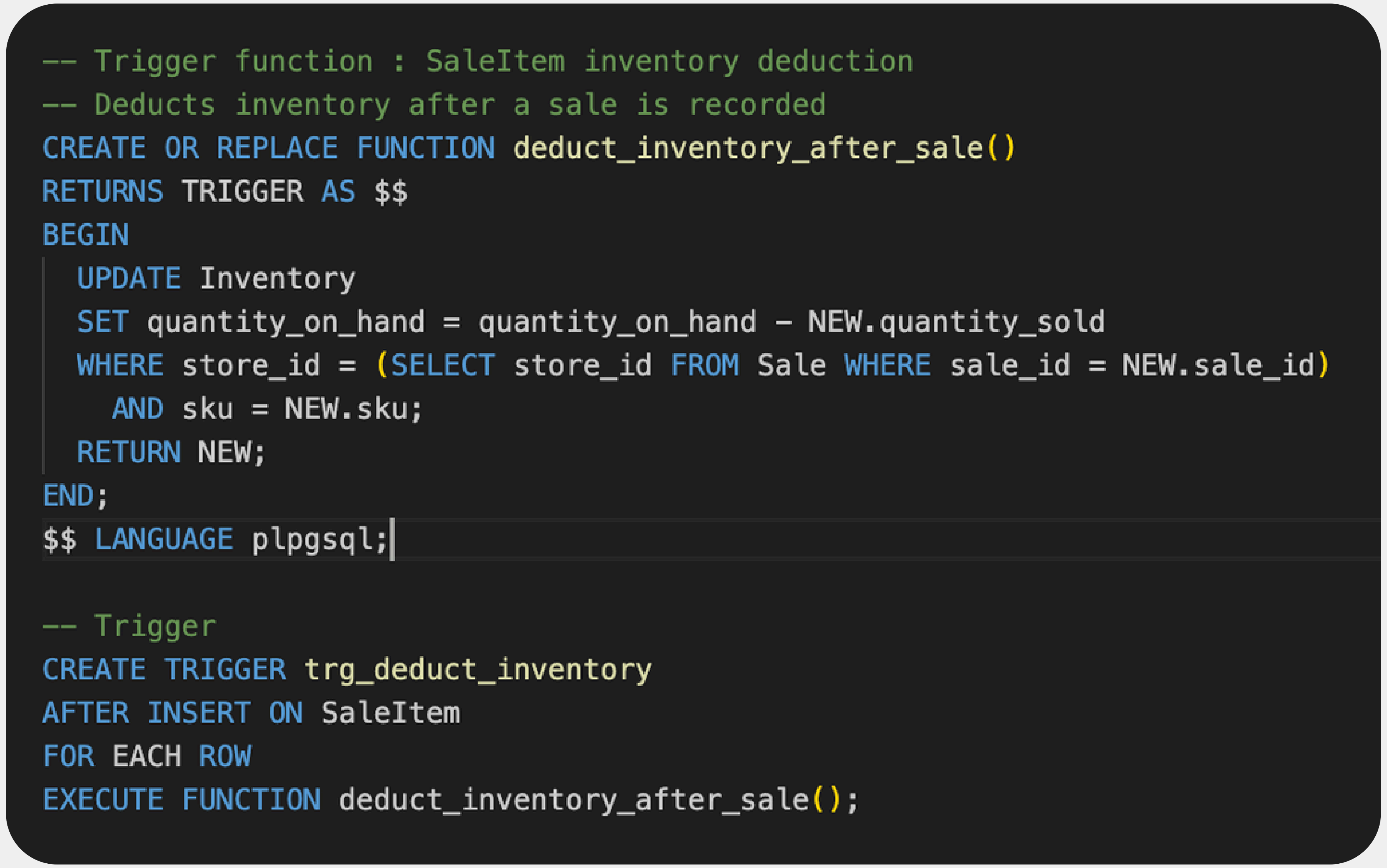
- Items within a sale, including promo details.
- Enables **profitability analysis**, **promotional impact tracking**



```sql
-- SaleItem Table
-- Details of each item sold in a transaction
CREATE TABLE SaleItem (
    sale_id INTEGER REFERENCES Sale(sale_id) ON DELETE CASCADE,
    sku varchar(20) REFERENCES Product(sku),
    quantity_sold INTEGER NOT NULL,
    unit_price NUMERIC(10,2) NOT NULL,
    promo_applied BOOLEAN DEFAULT FALSE,
    promo_discount NUMERIC(10,2),
    promo_id INTEGER REFERENCES Promotion(promo_id),
    PRIMARY KEY (sale_id, sku)
);
```

Trigger - Sale Item Inventory update

- Logic: **Finds the matching Inventory record** based on Sale.store_id and SKU and **subtracts quantity_sold.**
- Justification: **Maintains real-time inventory** accuracy after sales transactions.



```sql
-- Trigger function : SaleItem inventory deduction
-- Deducts inventory after a sale is recorded
CREATE OR REPLACE FUNCTION deduct_inventory_after_sale()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE Inventory
  SET quantity_on_hand = quantity_on_hand - NEW.quantity_sold
  WHERE store_id = (SELECT store_id FROM Sale WHERE sale_id = NEW.sale_id)
    AND sku = NEW.sku;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger
CREATE TRIGGER trg_deduct_inventory
AFTER INSERT ON SaleItem
FOR EACH ROW
EXECUTE FUNCTION deduct_inventory_after_sale();
```

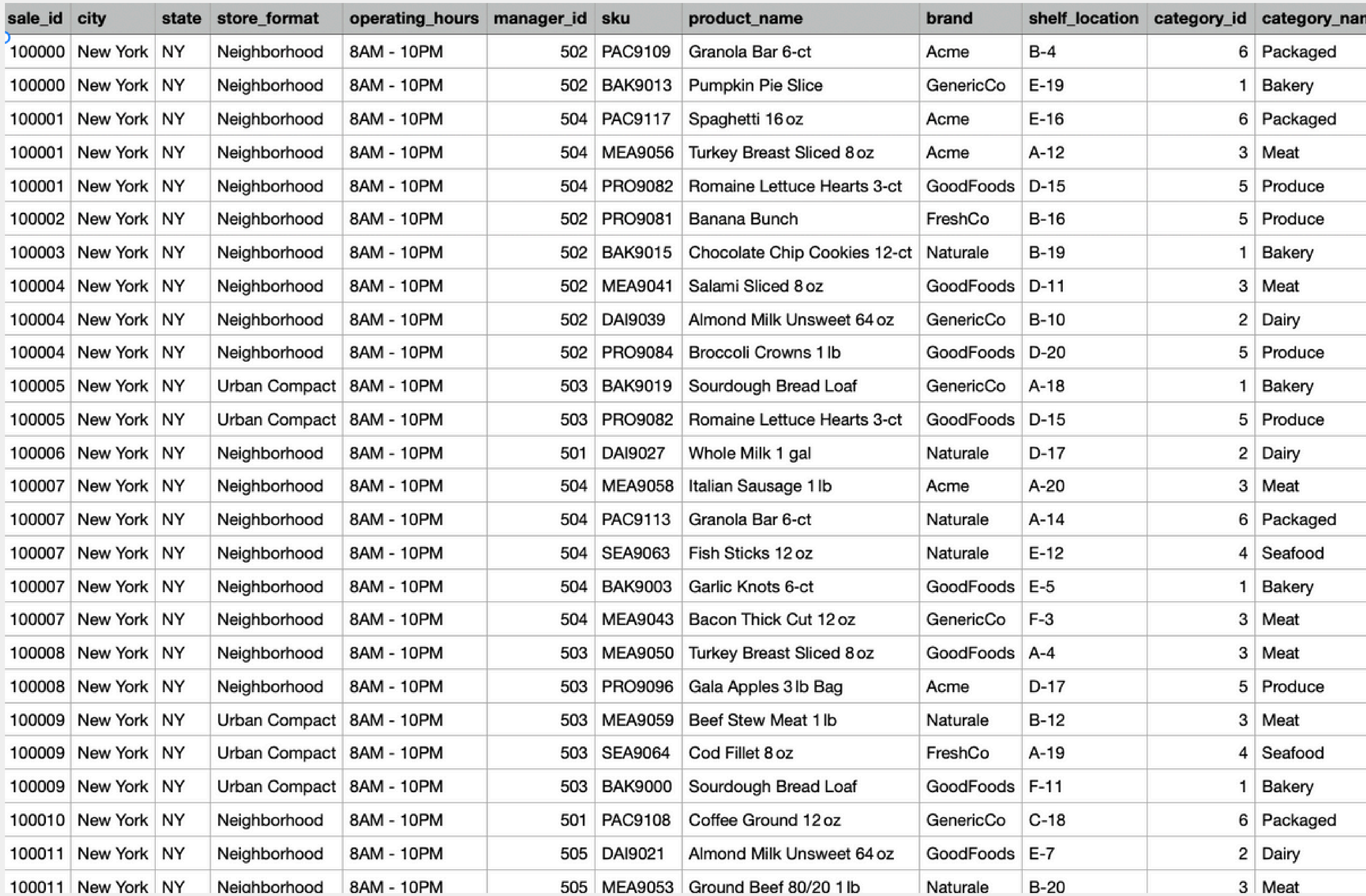
Data Generation

*Using LLMs - GPT-4, GPT-4o, GPT-3o

Four primary master tables

These datasets serve as the foundation for the ETL process, analytical queries, and dashboard visualizations.

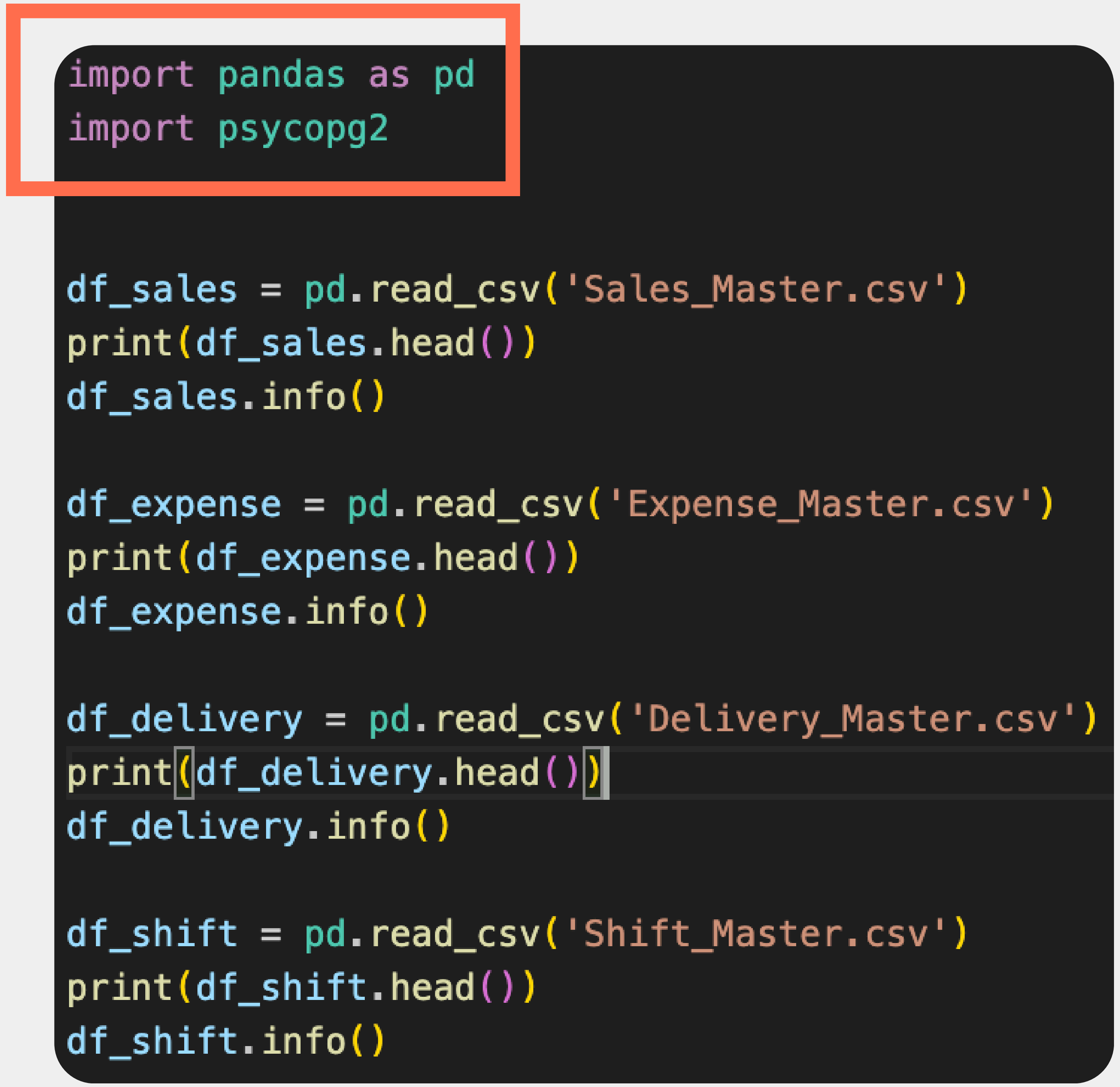
- Sales_Master.csv
- Shift_Master.csv
- Expense_Master.csv
- Delivery_Master.csv



sale_id	city	state	store_format	operating_hours	manager_id	sku	product_name	brand	shelf_location	category_id	category_nar
100000	New York	NY	Neighborhood	8AM - 10PM	502	PAC9109	Granola Bar 6-ct	Acme	B-4	6	Packaged
100000	New York	NY	Neighborhood	8AM - 10PM	502	BAK9013	Pumpkin Pie Slice	GenericCo	E-19	1	Bakery
100001	New York	NY	Neighborhood	8AM - 10PM	504	PAC9117	Spaghetti 16 oz	Acme	E-16	6	Packaged
100001	New York	NY	Neighborhood	8AM - 10PM	504	MEA9056	Turkey Breast Sliced 8 oz	Acme	A-12	3	Meat
100001	New York	NY	Neighborhood	8AM - 10PM	504	PRO9082	Romaine Lettuce Hearts 3-ct	GoodFoods	D-15	5	Produce
100002	New York	NY	Neighborhood	8AM - 10PM	502	PRO9081	Banana Bunch	FreshCo	B-16	5	Produce
100003	New York	NY	Neighborhood	8AM - 10PM	502	BAK9015	Chocolate Chip Cookies 12-ct	Naturale	B-19	1	Bakery
100004	New York	NY	Neighborhood	8AM - 10PM	502	MEA9041	Salami Sliced 8 oz	GoodFoods	D-11	3	Meat
100004	New York	NY	Neighborhood	8AM - 10PM	502	DAI9039	Almond Milk Unsweet 64 oz	GenericCo	B-10	2	Dairy
100004	New York	NY	Neighborhood	8AM - 10PM	502	PRO9084	Broccoli Crowns 1 lb	GoodFoods	D-20	5	Produce
100005	New York	NY	Urban Compact	8AM - 10PM	503	BAK9019	Sourdough Bread Loaf	GenericCo	A-18	1	Bakery
100005	New York	NY	Urban Compact	8AM - 10PM	503	PRO9082	Romaine Lettuce Hearts 3-ct	GoodFoods	D-15	5	Produce
100006	New York	NY	Neighborhood	8AM - 10PM	501	DAI9027	Whole Milk 1 gal	Naturale	D-17	2	Dairy
100007	New York	NY	Neighborhood	8AM - 10PM	504	MEA9058	Italian Sausage 1 lb	Acme	A-20	3	Meat
100007	New York	NY	Neighborhood	8AM - 10PM	504	PAC9113	Granola Bar 6-ct	Naturale	A-14	6	Packaged
100007	New York	NY	Neighborhood	8AM - 10PM	504	SEA9063	Fish Sticks 12 oz	Naturale	E-12	4	Seafood
100007	New York	NY	Neighborhood	8AM - 10PM	504	BAK9003	Garlic Knots 6-ct	GoodFoods	E-5	1	Bakery
100007	New York	NY	Neighborhood	8AM - 10PM	504	MEA9043	Bacon Thick Cut 12 oz	GenericCo	F-3	3	Meat
100008	New York	NY	Neighborhood	8AM - 10PM	503	MEA9050	Turkey Breast Sliced 8 oz	GoodFoods	A-4	3	Meat
100008	New York	NY	Neighborhood	8AM - 10PM	503	PRO9096	Gala Apples 3 lb Bag	Acme	D-17	5	Produce
100009	New York	NY	Urban Compact	8AM - 10PM	503	MEA9059	Beef Stew Meat 1 lb	Naturale	B-12	3	Meat
100009	New York	NY	Urban Compact	8AM - 10PM	503	SEA9064	Cod Fillet 8 oz	FreshCo	A-19	4	Seafood
100009	New York	NY	Urban Compact	8AM - 10PM	503	BAK9000	Sourdough Bread Loaf	GoodFoods	F-11	1	Bakery
100010	New York	NY	Neighborhood	8AM - 10PM	501	PAC9108	Coffee Ground 12 oz	GenericCo	C-18	6	Packaged
100011	New York	NY	Neighborhood	8AM - 10PM	505	DAI9021	Almond Milk Unsweet 64 oz	GoodFoods	E-7	2	Dairy
100011	New York	NY	Neighborhood	8AM - 10PM	505	MEA9053	Ground Beef 80/20 1 lb	Naturale	B-20	3	Meat

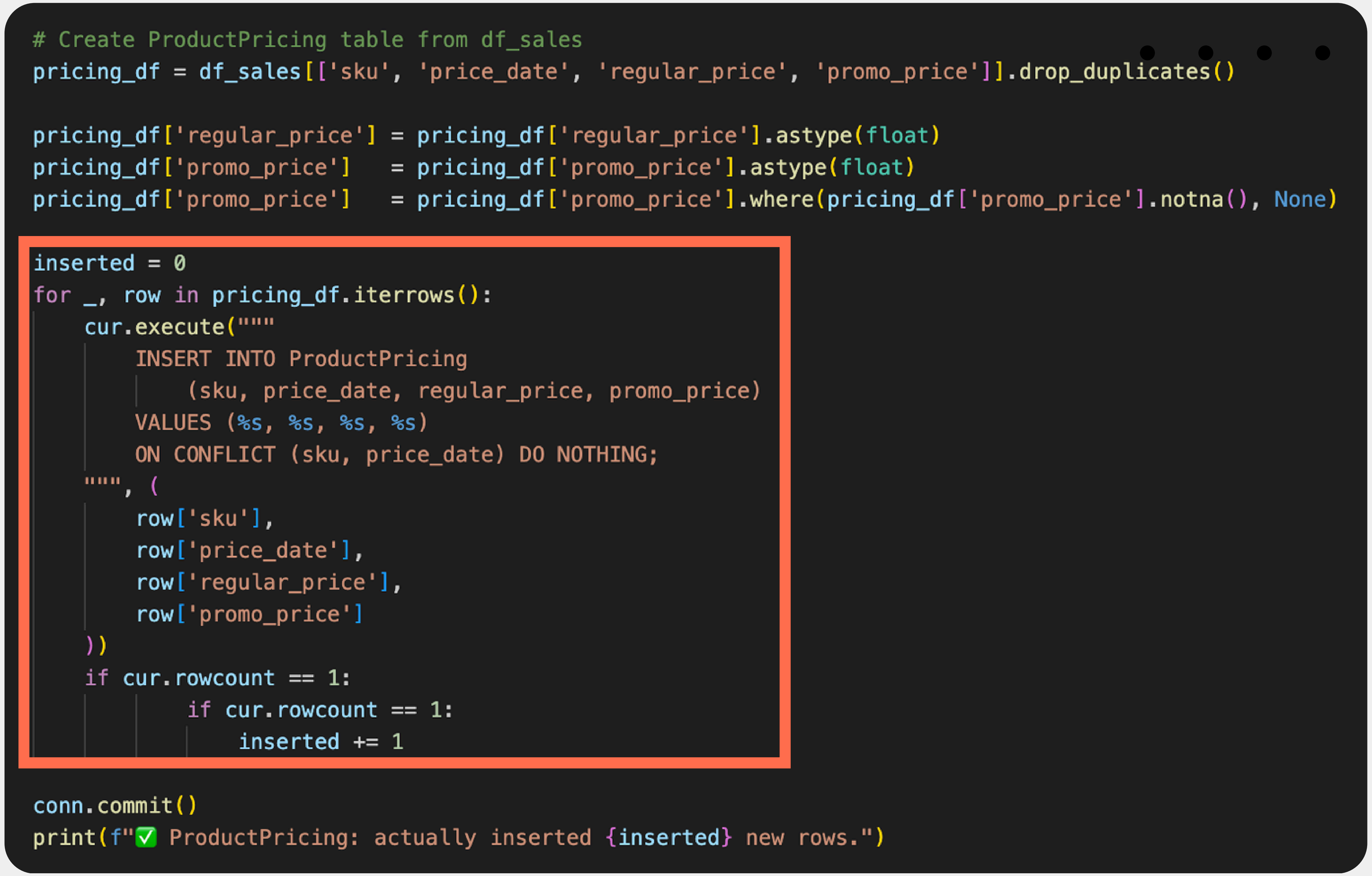
*Created dataset with detailed sales transactions at the item level (99612 rows)!

ETL Process



```python
import pandas as pd
import psycopg2


df_sales = pd.read_csv('Sales_Master.csv')
print(df_sales.head())
df_sales.info()

df_expense = pd.read_csv('Expense_Master.csv')
print(df_expense.head())
df_expense.info()

df_delivery = pd.read_csv('Delivery_Master.csv')
print(df_delivery.head())
df_delivery.info()

df_shift = pd.read_csv('Shift_Master.csv')
print(df_shift.head())
df_shift.info()
```

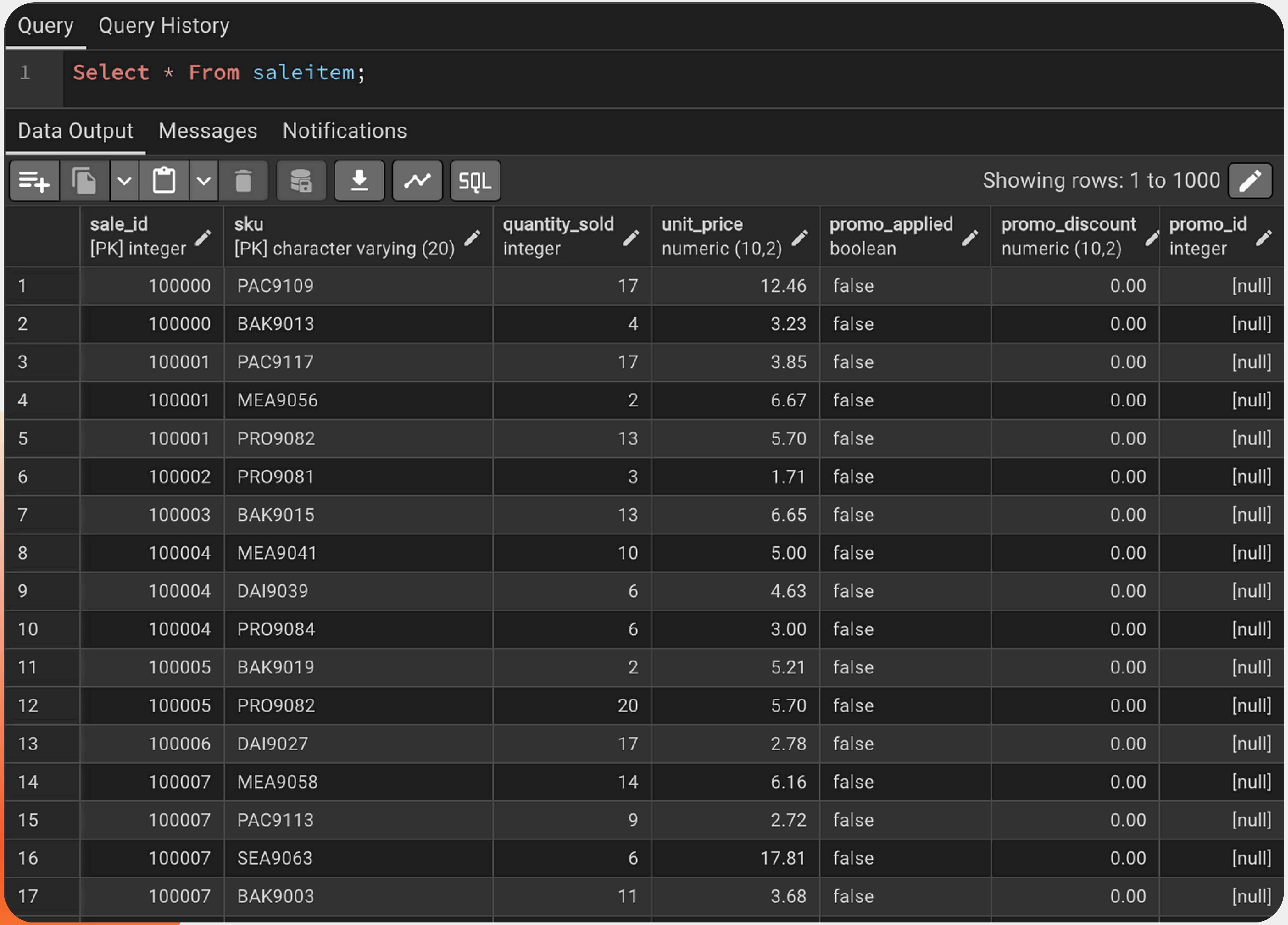
- **Loading** four master datasets



```python
# Create ProductPricing table from df_sales
pricing_df = df_sales[['sku', 'price_date', 'regular_price', 'promo_price']].drop_duplicates()

pricing_df['regular_price'] = pricing_df['regular_price'].astype(float)
pricing_df['promo_price']   = pricing_df['promo_price'].astype(float)
pricing_df['promo_price']   = pricing_df['promo_price'].where(pricing_df['promo_price'].notna(), None)

inserted = 0
for _, row in pricing_df.iterrows():
    cur.execute("""
        INSERT INTO ProductPricing
            (sku, price_date, regular_price, promo_price)
        VALUES (%s, %s, %s, %s)
        ON CONFLICT (sku, price_date) DO NOTHING;
    """, (
        row['sku'],
        row['price_date'],
        row['regular_price'],
        row['promo_price']
    ))
    if cur.rowcount == 1:
            if cur.rowcount == 1:
                inserted += 1

conn.commit()
print(f"✅ ProductPricing: actually inserted {inserted} new rows.")
```

- **Extract** required columns for each target table
- Rows iterated using Python **for loops**, and insert statements executed for each record.

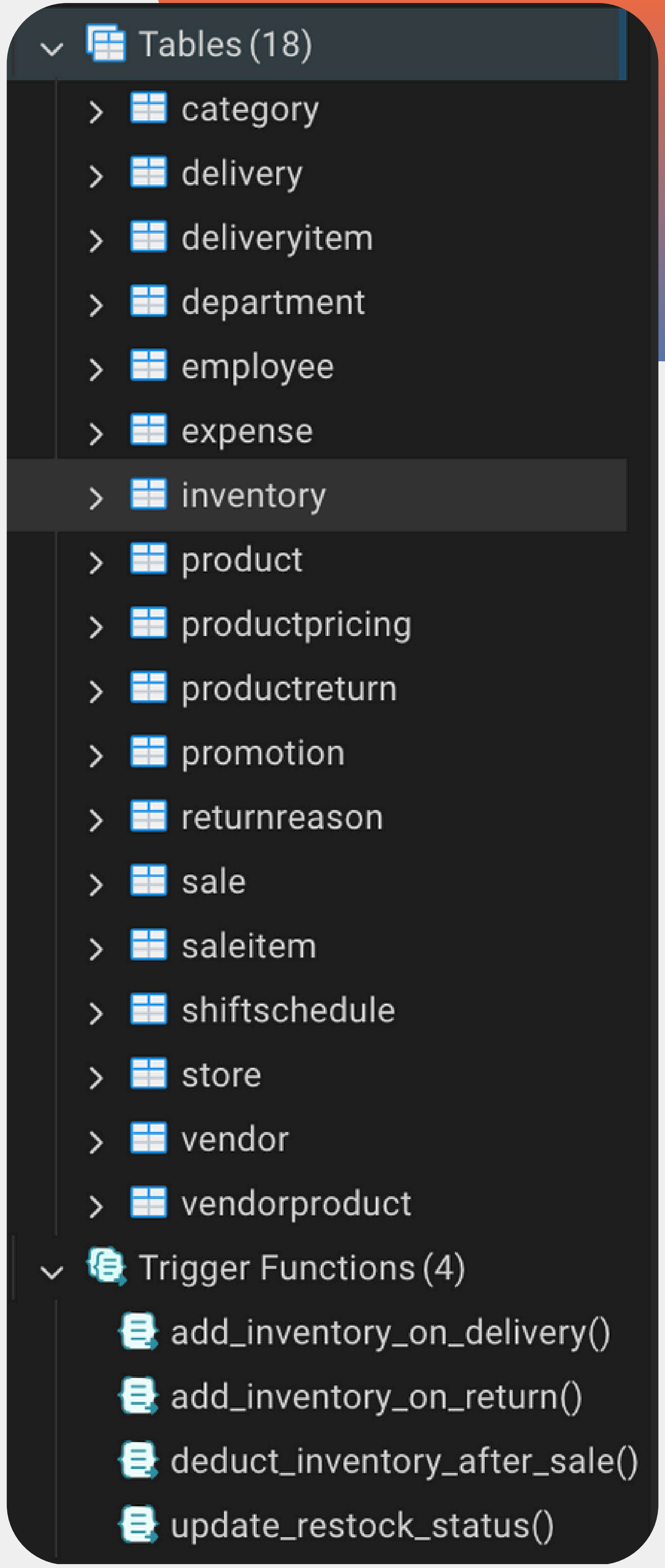
Loading Completed - Pgadmin



```sql
Select * From saleitem;
```

Showing rows: 1 to 1000

	sale_id [PK] integer	sku [PK] character varying (20)	quantity_sold integer	unit_price numeric (10,2)	promo_applied boolean	promo_discount numeric (10,2)	promo_id integer
1	100000	PAC9109	17	12.46	false	0.00	[null]
2	100000	BAK9013	4	3.23	false	0.00	[null]
3	100001	PAC9117	17	3.85	false	0.00	[null]
4	100001	MEA9056	2	6.67	false	0.00	[null]
5	100001	PRO9082	13	5.70	false	0.00	[null]
6	100002	PRO9081	3	1.71	false	0.00	[null]
7	100003	BAK9015	13	6.65	false	0.00	[null]
8	100004	MEA9041	10	5.00	false	0.00	[null]
9	100004	DAI9039	6	4.63	false	0.00	[null]
10	100004	PRO9084	6	3.00	false	0.00	[null]
11	100005	BAK9019	2	5.21	false	0.00	[null]
12	100005	PRO9082	20	5.70	false	0.00	[null]
13	100006	DAI9027	17	2.78	false	0.00	[null]
14	100007	MEA9058	14	6.16	false	0.00	[null]
15	100007	PAC9113	9	2.72	false	0.00	[null]
16	100007	SEA9063	6	17.81	false	0.00	[null]
17	100007	BAK9003	11	3.68	false	0.00	[null]



Tables (18)
- category
- delivery
- deliveryitem
- department
- employee
- expense
- inventory
- product
- productpricing
- productreturn
- promotion
- returnreason
- sale
- saleitem
- shiftschedule
- store
- vendor
- vendorproduct

Trigger Functions (4)
- add_inventory_on_delivery()
- add_inventory_on_return()
- deduct_inventory_after_sale()
- update_restock_status()

Complex Query

To provide valuable insight to the client, 11 complex queries have been developed based on **business requirements.** For instances:

Top Return Reasons by $ value

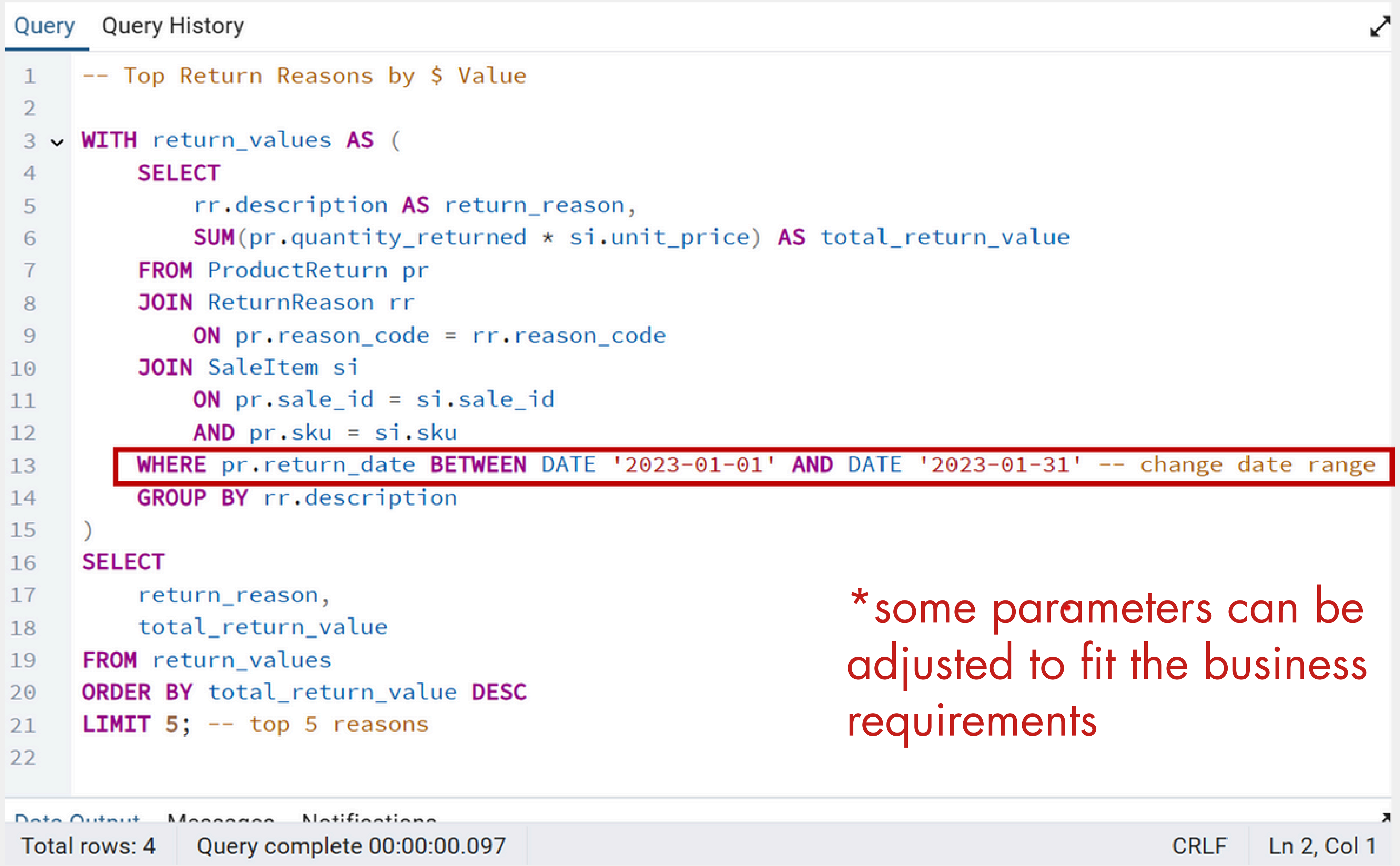
Client question:

"What's costing us the most in returns (defective/damaged/wrong item/etc)?"

Valuable insight:

To know what is the biggest contributor of financial loss and determine preventive action—tighten vendor SLAs, packaging, or fulfillment accuracy where it hurts most.

Complex Query



```sql
-- Top Return Reasons by $ Value

WITH return_values AS (
    SELECT
        rr.description AS return_reason,
        SUM(pr.quantity_returned * si.unit_price) AS total_return_value
    FROM ProductReturn pr
    JOIN ReturnReason rr
        ON pr.reason_code = rr.reason_code
    JOIN SaleItem si
        ON pr.sale_id = si.sale_id
        AND pr.sku = si.sku
    WHERE pr.return_date BETWEEN DATE '2023-01-01' AND DATE '2023-01-31' -- change date range
    GROUP BY rr.description
)
SELECT
    return_reason,
    total_return_value
FROM return_values
ORDER BY total_return_value DESC
LIMIT 5; -- top 5 reasons

```

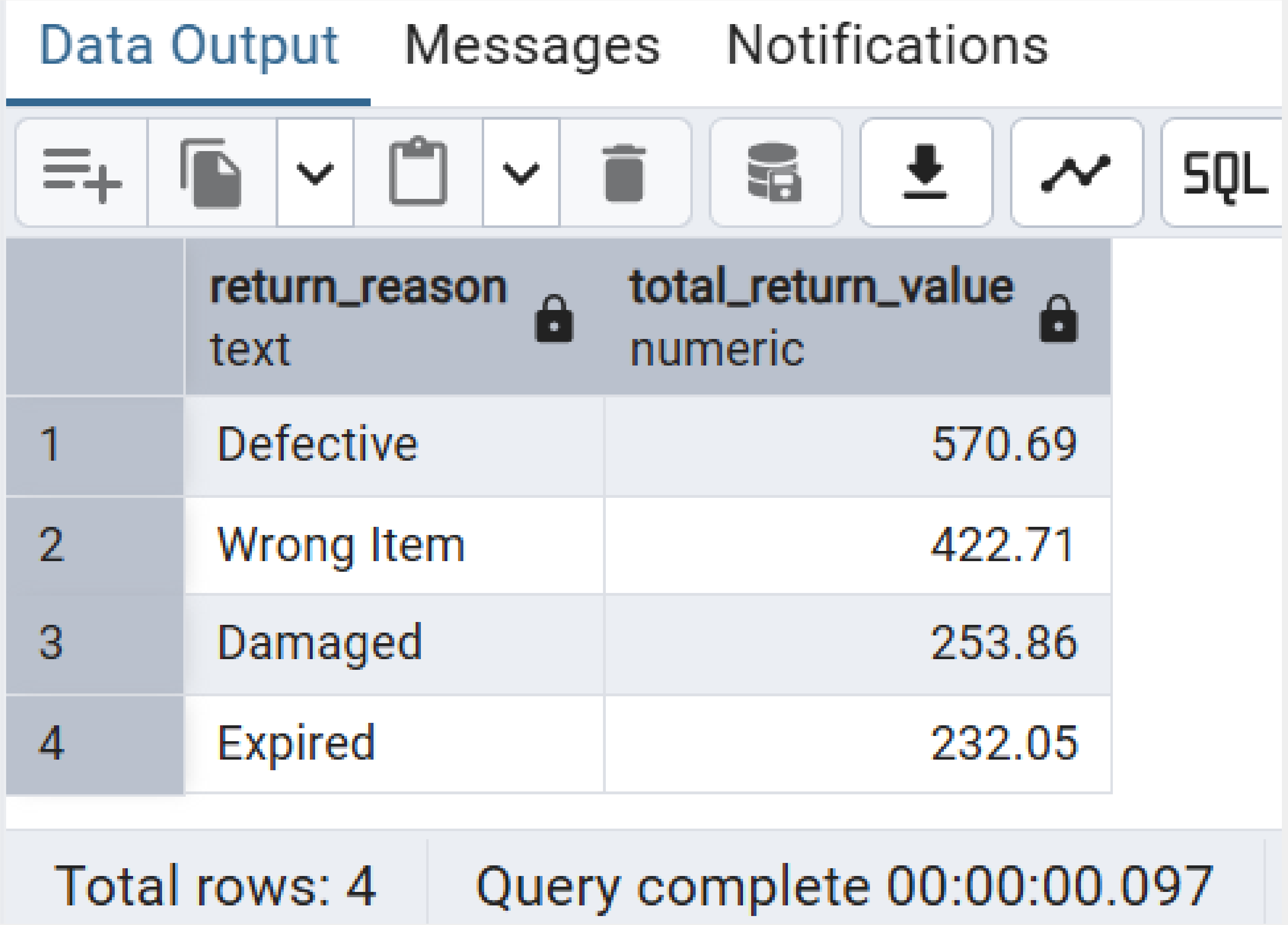
```
Total rows: 4    Query complete 00:00:00.097                    CRLF    Ln 2, Col 1
```

*some parameters can be adjusted to fit the business requirements

Data Analyst will build the script based on the business requirements with purpose to get data insights.

SQL script will be tested in pgAdmin before it will be launched.

Complex Query



	return_reason text	total_return_value numeric
1	Defective	570.69
2	Wrong Item	422.71
3	Damaged	253.86
4	Expired	232.05

Total rows: 4 Query complete 00:00:00.097

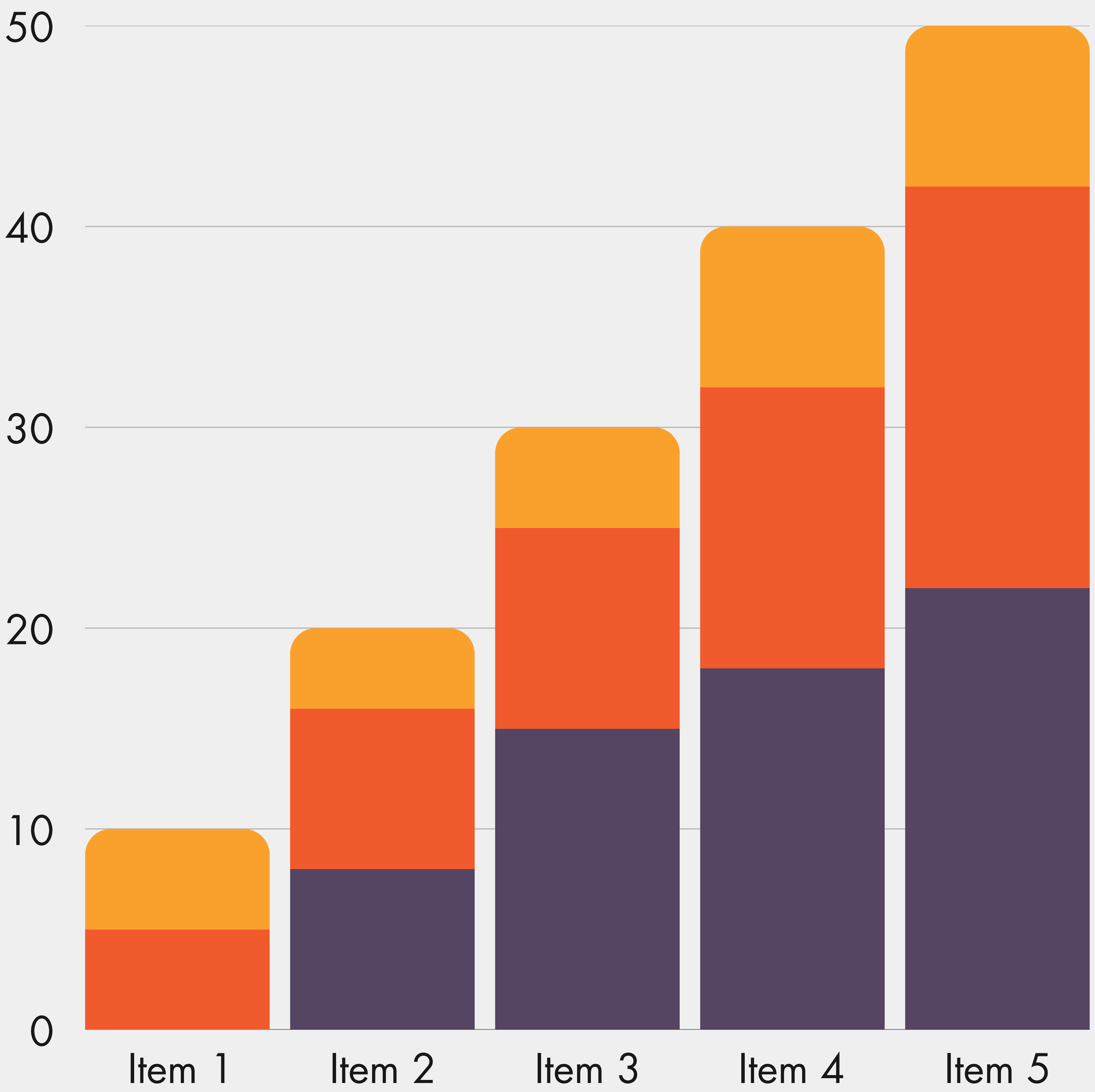
Complex query result will be visualized via Metabase Dashboard

This result can be used to determine actionable item for management to **reduce financial loss**, in this case due to defective item.

Interactive Dashboard

SQL/Metabase

Live Demonstration



Conclusion

- Our goal was to equip ABC Foodmart with a **centralized, scalable, and analytics-ready relational database** to replace fragmented manual systems, support expansion, and **enable data-driven decision-making**.

- Now, ABC Foodmart can make **faster, more informed decisions**, optimize stock levels, improve labor allocation, enhance vendor performance, and focus marketing spend where it delivers the highest ROI.

End

Thank you

Do you have any questions?