Group 1 APAN5310 Final Project Report

Pedro Flores, Tingting Fang, Kanyarat Suwannama, and Matthew Michael

Consultant Client Scenario

Our team chose to work with the ABC Foodmart grocery store scenario. ABC Foodmart is a grocery store that has been operating in Queens, New York for over 30 years. The grocery store is looking to continue its expansion to three additional locations in Brooklyn, New York. Their challenge is that all company data has been stored manually in paper binders or at best, in spreadsheets. This limits the company's ability to track key performance indicators that can be used for continuous improvement. Moreover, they are unable to monitor the health of the business across all stores and take corrective actions as needed.

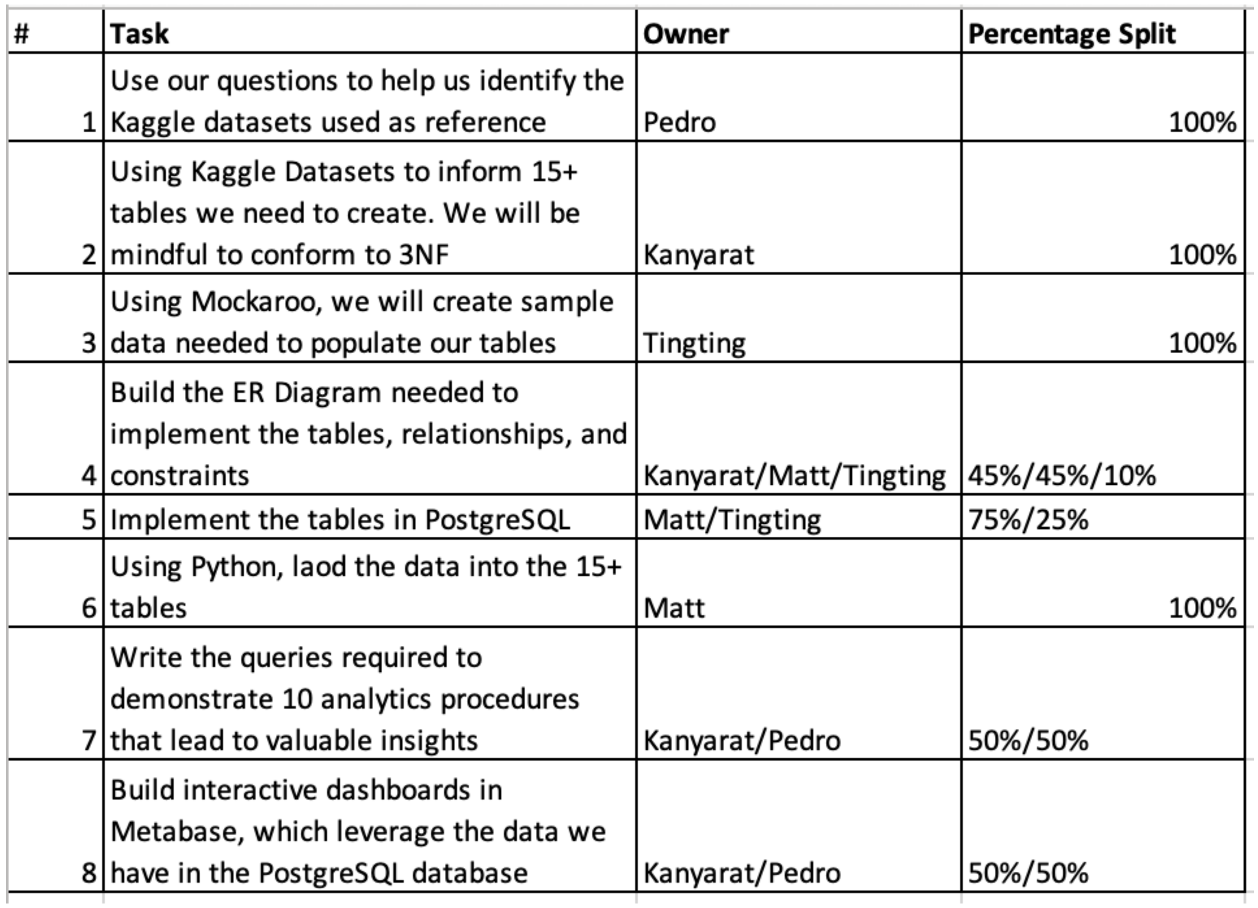
We chose to tackle the ABC Foodmart scenario due to the significant challenges that the company is facing and the positive impact that a relational database management system will have on the company's potential for success. Going from manual documentation storage to a RDBMS will not only improve the company's operation, but it will enable the company to better compete in today's very competitive grocery store market.

Our initial plan of action was to perform research on the key performance indicators used in the supermarket business that best-in-class companies use in order to be successful. Then, knowing the important information that we needed to make available for ABC Foodmart's management to run the business, we set out to design a RDBMS that contained all relevant information and better guaranteed data atomicity, consistency, isolation, and durability to ensure the best performance possible and that ABC Foodmart will survive the test of time.

The robust RDBMS will make the following key performance indicators readily available to the company's management: employee turnover, sales, inventory, vendors, loyalty programs, marketing campaigns, among others. The company manages multiple locations, so ensuring management can monitor and continuously improve its business operations will be key to ensuring the successful expansion into the Brooklyn market.

Team Contract

Our team divided up the eight tasks required of us. Below is a table describing the eight tasks, who owns each, and percentage splits where applicable.



#	Task	Owner	Percentage Split
1	Use our questions to help us identify the Kaggle datasets used as reference	Pedro	100%
2	Using Kaggle Datasets to inform 15+ tables we need to create. We will be mindful to conform to 3NF	Kanyarat	100%
3	Using Mockaroo, we will create sample data needed to populate our tables	Tingting	100%
4	Build the ER Diagram needed to implement the tables, relationships, and constraints	Kanyarat/Matt/Tingting	45%/45%/10%
5	Implement the tables in PostgreSQL	Matt/Tingting	75%/25%
6	Using Python, laod the data into the 15+ tables	Matt	100%
7	Write the queries required to demonstrate 10 analytics procedures that lead to valuable insights	Kanyarat/Pedro	50%/50%
8	Build interactive dashboards in Metabase, which leverage the data we have in the PostgreSQL database	Kanyarat/Pedro	50%/50%

Sample Data

We started researching what Grocery Store management personnel would be interested in by performing some preliminary research about Grocery store chains like Walmart. Once we had an idea about some of the most important areas of the business, we explored Kaggle data across those areas. Our findings are below:

1. Sales/Account:

Example columns: transaction_id, transaction_timestamp, product_id, product_name, category, sub-category, brand, add_to_cart_order, reordered, department, membership_type, order_number, order_dow, order_hour_of_day, days_since_prior_order, unit_price, quantity, discount, total_sale, promotion_code, payment_type

Source:
https://www.kaggle.com/datasets/abhinayasaravanan/grocery-supply-chain-isuue?select=Grocery_Sales.csv,
https://www.kaggle.com/datasets/hunter0007/ecommerce-dataset-for-predictive-marketing-2023

2. Vendor:

Example columns: vendor_id, vendor, address, email, phone

Source:
https://www.kaggle.com/datasets/abhinayasaravanan/grocery-supply-chain-isuue?select=sensor_stock_levels.csv

3. Customer:

Example columns: customer_id, name, gender, address, membership_type, spending_score

Source:
https://www.kaggle.com/datasets/sindraanthony9985/marketing-data-for-a-supermarket-in-united-states

4. Delivery:

Example columns: trip_id, customer_id, delivery_address, delivery_start, delivery_end

Source:https://www.kaggle.com/datasets/viswajithkn/instacart-predict-shopping-time?select=train_trips.csv

5. Staff:

Example columns: employee_id, department, name, address, education, gender, age, position, salary, satisfaction, attrition

Source: https://www.kaggle.com/datasets/sonalishanbhag/employee-attrition

These Kaggle datasets informed our primary dataset, which was put together with Mockaroo. We have included a link to the dataset below:

Link:
https://drive.google.com/file/d/1XTtFaw_tIG0o0UABS2YjLLNv1yxj6UiJ/view?usp=sharing

Normalization Plan

With our research and Kaggle datasets guiding us, we started to build the entities, relationships, and entity attributes that would ultimately become our Lucidchart ERD. Given the Third Normal Form requirement, we were mindful of the fact that all attributes within each table must be functionally dependent on solely the primary key. So, rather than having a flat dataset with redundancy across entities, we broke the entities out across fifteen tables. Below is an overview of our thought process as we built out the tables:

Addresses: The addresses table contains ID (primary key), street, city, state, and country columns. Each attribute is completely dependent on the primary key. It has no dependencies (foreign key relationships) and therefore served as a great starting point for our ERD. Many other tables, though, have references to addresses. As an example, Employees live at addresses, so we built our Employees table next.

Employees: The employees table contains ID (primary key), First Name, Last Name, Start Date, End Date, Status, and Address ID (foreign key that references the Addresses table) columns. Again, each of these columns is completely dependent on the primary key. Employees work at stores, but if we were to include in the employees table any attribute that is specific to the store, we would break the rules of third normal form. So, we built the Stores table and a Store Role junction table shortly thereafter.

Stores: The Stores table contains ID (primary key), Sq ft, Name, and Address ID (foreign key that references the Addresses table) columns. Stores are located at specific addresses, hence the foreign key relationship. These columns are all completely dependent on the primary key and have no direct relationship with the Employees table.

Store Role: The Store Role junction table enables the relationship between Employees and Stores. This table contains Employee ID, Store ID, Role Type, Role Start, and Role End columns. Employee ID, Store ID, and Role Type serve as the composite primary key. Employee ID references the Employees table and Store ID references the Stores table. Using the Store Role table, we can describe what role each employee has at a store (management, baker, cashier, etc.), when they started, and, if they are a former employee, when their role ended.

Vendors: The Vendors table contains ID (primary key), Name, Vendor Type, and Address ID (foreign key that references the Addresses table) columns. Vendors are located at particular Addresses, hence the foreign key relationship. These columns are all completely dependent on the primary key.

Deliveries: The Deliveries table contains ID (primary key), Store ID (foreign key that references the Stores table), Time, and Vendor ID (foreign key that references the Vendors table) columns. Deliveries are made to Stores by Vendors, hence the foreign key relationships. With two foreign table relationships, the Stores and Vendors tables must be created in the database first. All of these columns are fully dependent on the primary key.

Carts: The Carts table contains ID (primary key), Store ID (foreign key that references the Stores table), and Quantity columns. Shoppers use carts to shop for items at particular stores, hence the relationship with Stores. These columns are all completely dependent on the primary key.

Inventory: Inventory represents a daily snapshot of the inventory stock. So, as of a certain date, how much inventory is in stock? This table has no foreign key relations. The columns are ID (primary key), Inventory Stock, and Date.

Items: The Items table contains ID (primary key), Delivery ID (foreign key that references the Deliveries table), Name, Purchase Price, Purchase Date, Cart ID (foreign key that references the Carts table), and Inventory ID (foreign key that references the Inventory table) columns. Items are delivered to stores, they are placed in carts, and they are sourced from Inventory, hence the foreign key relations. With three foreign table relationships, the Deliveries, Carts, and Inventory tables must be created in the database first. All of these columns are fully dependent on the primary key.

Departments: The Departments table contains ID (primary key), Store ID (foreign key that references the Stores table), and Name columns. Departments are within stores, hence the relationship with Stores. These columns are all completely dependent on the primary key.

Accounts: Accounts refer to bank accounts specific to ABF Foodmart. Accounts include an Account Number (primary key), an Account Type (checking, savings, etc), and an Account Balance. There are no foreign key relationships with other tables. All columns are completely dependent on the primary key.

Loyalty Member: ABC Foodmart's Loyalty program is the way in which we recommend they track customer relationships. In joining the loyalty program, customers provide to ABC Foodmart their First Name, Last Name, email address, Address (foreign key referencing Addresses), and Primary phone number, all of which are columns in the database. The database assigns each loyalty member an ID (primary key) and the Level is derived based on the customer's purchase volume. Marketing campaigns are Marketed To Loyalty Members (see below). All columns are completely dependent on the primary key.

Marketing Campaigns: Marketing Campaigns are assigned an ID (primary key), have a budget, and marketed a particular Department (foreign key referencing the Departments table). All columns are completely dependent on the primary key.

Marketed To: Marketed To is a junction table sitting between Loyalty Members (primary key and foreign key referencing the Loyalty Members table) and Marketing Campaigns (primary key and foreign key referencing the Marketing Campaigns table) for tracking purposes.

Transactions: The Transactions table contains ID (primary key), Store ID (foreign key that references the Stores table), Time, Payment Type, Transaction Type, Loyalty Member ID (foreign key that references the Loyalty Member table), Cart ID (foreign key that references the Carts table), and Account ID (foreign key that references the Accounts table) columns. Transactions are completed by Loyalty Members (assuming a customer is part of the loyalty program), at stores and from carts. The proceeds from the transaction are stored in the Store Account. This is why we created the foreign key relations with Stores, Loyalty Members, Carts, and Accounts. With four foreign table relationships, the Stores, Loyalty Members, Carts, and Account tables must be created in the database first. All of these columns are fully dependent on the primary key.

We hope the above explanation conveys how we approached the relationships between entities and how we best ensured adherence to the rules of third normal form. Below is a link to our Lucidchart.

https://lucid.app/lucidchart/3d49d586-f91d-42e8-aefc-5c5ee1b3ef41/edit?viewport_loc=-314%2C-574%2C4353%2C2108%2C0_0&invitationId=inv_4aa4c971-18d1-4806-a54d-13765084afdf

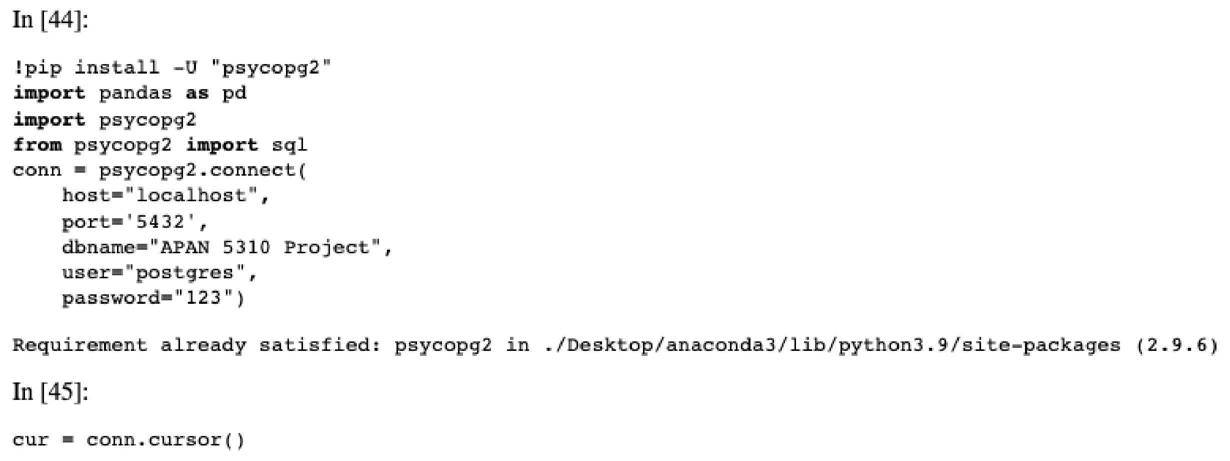
Additionally, you will find our SQL code below:

```
create table inventory (
        ID serial primary key,
        inventory_stock integer,
        date date
);

create type account_type as enum ('Checking', 'Growth', 'Reserve');

create table accounts (
        account_number varchar(255) primary key,
        account_type account_type,
        balance decimal
);

create type state as enum (
   'Alabama',
   'Alaska',
   'Arizona',
   'Arkansas',
   'California',
   'Colorado',
   'Connecticut',
   'Delaware',
   'Florida',
   'Georgia',
   'Hawaii',
   'Idaho',
   'Illinois',
   'Indiana',
   'Iowa',
   'Kansas',
   'Kentucky',
   'Louisiana',
   'Maine',
   'Maryland',
   'Massachusetts',
   'Michigan',
   'Minnesota',
   'Mississippi',
   'Missouri',
   'Montana',
   'Nebraska',
   'Nevada',
   'New Hampshire',
   'New Jersey',
   'New Mexico',
   'New York',
   'North Carolina',
   'North Dakota',
   'Ohio',
   'Oklahoma',
   'Oregon',
   'Pennsylvania',
   'Rhode Island',
   'South Carolina',
   'South Dakota',
   'Tennessee',
   'Texas',
   'Utah',
   'Vermont',
   'Virginia',
   'Washington',
   'West Virginia',
   'Wisconsin',
   'Wyoming'
);

create type country as enum (
        'United States',
        'Canada',
        'Mexico'
);
```

```
create table addresses (
        ID serial primary key,
        street varchar(255),
        city varchar(255),
        state state,
        country country
);

create table stores (
        ID serial,
        sq_ft integer,
        name varchar(255),
        address_id integer,
        primary key (id),
        foreign key (address_id) references addresses (id)
);

create type status as enum ('Current', 'Former');

create table employees (
        id serial,
        first_name varchar(255),
        last_name varchar(255),
        start_date date,
        end_date date,
        status status,
        address_id integer,
        primary key (id),
        foreign key (address_id) references addresses (id)
);

create type role_type as enum (
        'Cashier',
        'Bakery',
        'Deli',
        'Inventory Stocking',
        'Fish',
        'Management',
        'Accounting',
        'Technology',
        'Produce'
);

create table store_role (
        employee_id integer,
        store_id integer,
        role_type role_type,
        role_start date,
        role_end date,
        primary key (employee_id, store_id, role_type),
        foreign key (store_id) references stores (id),
        foreign key (employee_id) references employees (id)
);

create table departments (
        id serial primary key,
        name varchar(255),
        store_id integer,
        foreign key (store_id) references stores (id)
);

create table marketing_campaigns (
        id serial,
        campaign_budget decimal,
        department_id integer,
        primary key (id),
        foreign key (department_id) references departments (id)
);

create type level as enum ('Platinum','Gold','Silver','Bronze');

create table loyalty_member (
        id serial,
        first_name varchar(255),
        last_name varchar(255),
        level level,
        email varchar(320),
	primary_number varchar(20),
	address_id integer,
	primary key (id),
	foreign key (address_id) references addresses(id)
);

create table marketed_to (
	loyalty_member_id integer,
	marketing_campaign_id integer,
	primary key (loyalty_member_id, marketing_campaign_id),
	foreign key (loyalty_member_id) references loyalty_member (id),
	foreign key (marketing_campaign_id) references marketing_campaigns (id)
);

create type vendor_type as enum (
	'Produce',
	'Poultry',
	'Equipment',
	'Fish',
	'Electronics',
	'Packaged Goods',
	'Security');

create table vendors (
	id serial,
	name varchar(255),
	vendor_type vendor_type,
	address_id integer,
	primary key (id),
	foreign key (address_id) references addresses(id)
);

create table deliveries (
	id serial,
	datetime time,
	store_id integer,
	vendor_id integer,
	primary key (id),
	foreign key (store_id) references stores (id),
	foreign key (vendor_id) references vendors (id)
);

create table carts (
	id serial,
	quantity integer,
	store_id integer,
	primary key (id),
	foreign key (store_id) references stores (id)
);

create table items (
	id serial,
	name varchar(255),
	purchase_price decimal,
	purchase_date date,
	delivery_id integer,
	cart_id integer,
	inventory_id integer,
	primary key (id),
	foreign key (delivery_id) references deliveries (id),
	foreign key (cart_id) references carts (id),
	foreign key (inventory_id) references inventory (id)
);

create type payment_type as enum ('Credit Card','Cash','Check','Digital Payment');
create type transaction_type as enum ('Credit','Debit');

create table transactions (
	id serial,
	datetime time,
	payment_type payment_type,
	transaction_type transaction_type,
	loyalty_member_id integer,
	cart_id integer,
	account_id varchar(255),
	store_id integer,
	primary key (id),
	foreign key (loyalty_member_id) references loyalty_member (id),
        foreign key (cart_id) references carts (id),
        foreign key (account_id) references accounts (account_number),
        foreign key (store_id) references stores (id)
);
```

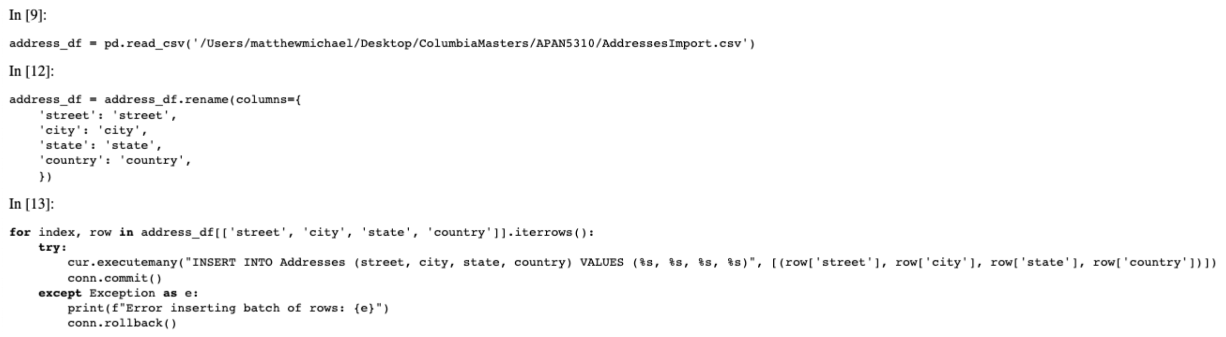
ETL Process

For the data import, we imported the pandas and psycopg2 python packages. Leveraging Jupyter Notebook, we established a connection with our database. While screenshots and explanations are included below, the entirety of our code is available in a Github repository: https://github.com/mmichael12/APAN5310-Group-1-Project



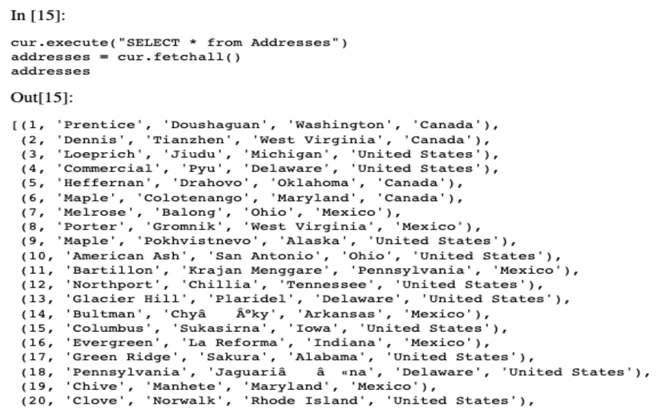
```
In [44]:

!pip install -U "psycopg2"
import pandas as pd
import psycopg2
from psycopg2 import sql
conn = psycopg2.connect(
    host="localhost",
    port='5432',
    dbname="APAN 5310 Project",
    user="postgres",
    password="123")

Requirement already satisfied: psycopg2 in ./Desktop/anaconda3/lib/python3.9/site-packages (2.9.6)

In [45]:

cur = conn.cursor()
```

After establishing the connection, we created a pandas dataframe for the Addresses table and renamed the columns in the file to correspond to those without our PostgreSQL database. With the data frame created, we then iterated over each of the rows within the file and inserted the column values into the database. To ensure we could identify and resolve any exceptions, we included a try/except block, which printed eros and rolled back the connection should an error occur.



```
In [9]:

address_df = pd.read_csv('/Users/matthewmichael/Desktop/ColumbiaMasters/APAN5310/AddressesImport.csv')

In [12]:

address_df = address_df.rename(columns={
    'street': 'street',
    'city': 'city',
    'state': 'state',
    'country': 'country',
    })

In [13]:

for index, row in address_df[['street', 'city', 'state', 'country']].iterrows():
    try:
        cur.executemany("INSERT INTO Addresses (street, city, state, country) VALUES (%s, %s, %s, %s)", [(row['street'], row['city'], row['state'], row['country'])])
        conn.commit()
    except Exception as e:
        print(f"Error inserting batch of rows: {e}")
        conn.rollback()
```

To ensure the success of the import, we executed a SELECT * query and printed the addresses in the database.



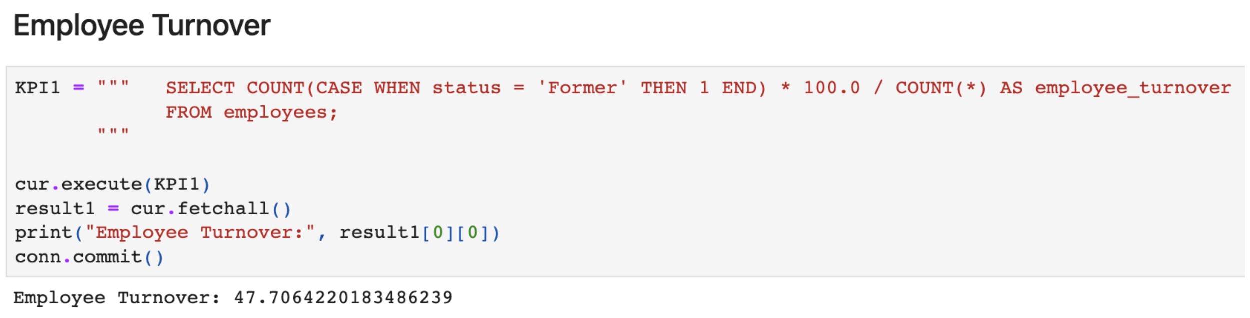
```
In [15]:

cur.execute("SELECT * from Addresses")
addresses = cur.fetchall()
addresses

Out[15]:

[(1, 'Prentice', 'Doushaguan', 'Washington', 'Canada'),
 (2, 'Dennis', 'Tianzhen', 'West Virginia', 'Canada'),
 (3, 'Loeprich', 'Jiudu', 'Michigan', 'United States'),
 (4, 'Commercial', 'Pyu', 'Delaware', 'United States'),
 (5, 'Heffernan', 'Drahovo', 'Oklahoma', 'Canada'),
 (6, 'Maple', 'Colotenango', 'Maryland', 'Canada'),
 (7, 'Melrose', 'Balong', 'Ohio', 'Mexico'),
 (8, 'Porter', 'Gromnik', 'West Virginia', 'Mexico'),
 (9, 'Maple', 'Pokhvistnevo', 'Alaska', 'United States'),
 (10, 'American Ash', 'San Antonio', 'Ohio', 'United States'),
 (11, 'Bartillon', 'Krajan Menggare', 'Pennsylvania', 'Mexico'),
 (12, 'Northport', 'Chillia', 'Tennessee', 'United States'),
 (13, 'Glacier Hill', 'Plaridel', 'Delaware', 'United States'),
 (14, 'Bultman', 'Chyâ   Â°ky', 'Arkansas', 'Mexico'),
 (15, 'Columbus', 'Sukasirna', 'Iowa', 'United States'),
 (16, 'Evergreen', 'La Reforma', 'Indiana', 'Mexico'),
 (17, 'Green Ridge', 'Sakura', 'Alabama', 'United States'),
 (18, 'Pennsylvania', 'Jaguariâ   â «na', 'Delaware', 'United States'),
 (19, 'Chive', 'Manhete', 'Maryland', 'Mexico'),
 (20, 'Clove', 'Norwalk', 'Rhode Island', 'United States'),
```

We repeated this process for the other tables while being mindful of the sequence in which we imported the data. Given we could only import data that had foreign key relations after those foreign key relations had been imported, junction table inserts had to follow the associated tables.

10 Analytical Procedures that Led to Valuable Insights

1. Employee Turnover: calculated as the count of employees that leave the supermarket over a certain period. The success of the business highly depends on low employee turnover as it measures how successful the company is to onboard employees, train them, be successful, and contribute. High employee turnover results in talent losses and higher onboarding costs that are ultimately detrimental to the company's success. Figure 1 below shows the ratio of employee turnover for the current year.

Figure 1 – Employee turnover in calendar year 2023



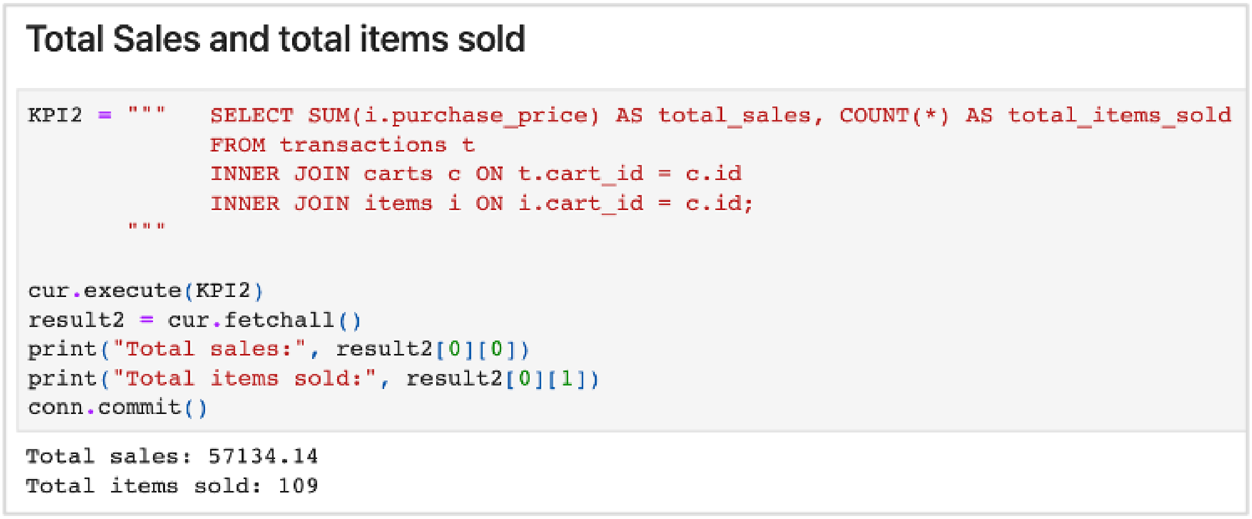
Employee Turnover

```
KPI1 = """   SELECT COUNT(CASE WHEN status = 'Former' THEN 1 END) * 100.0 / COUNT(*) AS employee_turnover
             FROM employees;
       """

cur.execute(KPI1)
result1 = cur.fetchall()
print("Employee Turnover:", result1[0][0])
conn.commit()
```

```
Employee Turnover: 47.7064220183486239
```

2. Average price: calculated as the average price per item sold as an indicator of sales volume by supermarket customers. It helps the supermarket understand general customer purchasing trends over time. Figure 2 below shows the average price for products sold.

Figure 2 – Average price per item sold



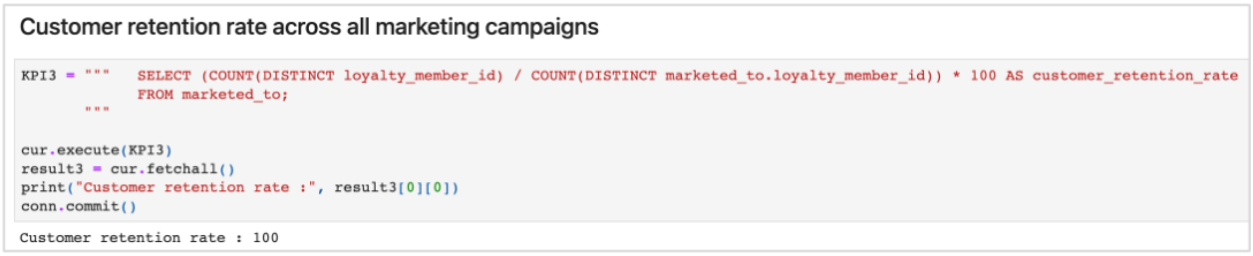
Total Sales and total items sold

```
KPI2 = """   SELECT SUM(i.purchase_price) AS total_sales, COUNT(*) AS total_items_sold
             FROM transactions t
             INNER JOIN carts c ON t.cart_id = c.id
             INNER JOIN items i ON i.cart_id = c.id;
       """

cur.execute(KPI2)
result2 = cur.fetchall()
print("Total sales:", result2[0][0])
print("Total items sold:", result2[0][1])
conn.commit()
```

```
Total sales: 57134.14
Total items sold: 109
```

3. Marketing Campaign Retention Rate: compared the number of customers that are currently part of the supermarket's loyalty program versus the number of customers that the supermarket has reached out to via marketing campaigns. This metric helps the supermarket understand if the marketing campaigns are targeting all loyal customers to ensure that resources are reaching the most loyal. Figure 3 below shows the customer Marketing Campaign Retention Rate.

Figure 3 - Marketing Campaign Retention Rate



Customer retention rate across all marketing campaigns

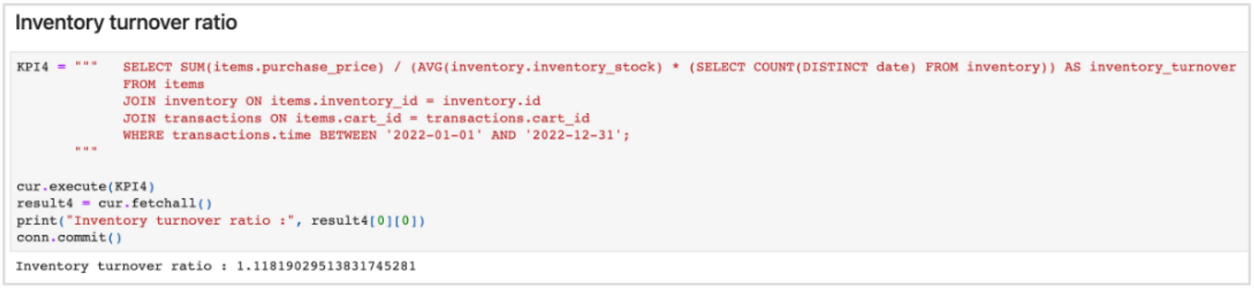
```
KPI3 = """   SELECT (COUNT(DISTINCT loyalty_member_id) / COUNT(DISTINCT marketed_to.loyalty_member_id)) * 100 AS customer_retention_rate
             FROM marketed_to;
       """

cur.execute(KPI3)
result3 = cur.fetchall()
print("Customer retention rate :", result3[0][0])
conn.commit()
```

```
Customer retention rate : 100
```

4. Inventory Turnover Ratio: calculated as the number of times inventory is sold during a given period. The higher the ratio, the stronger the signal for higher

sales. It also provides the supermarket an indication of overall sales over a period of time. Figure 4 below shows the Inventory Turnover Ratio for calendar year 2022.

Figure 4 – 2022 Inventory Turnover Ratio



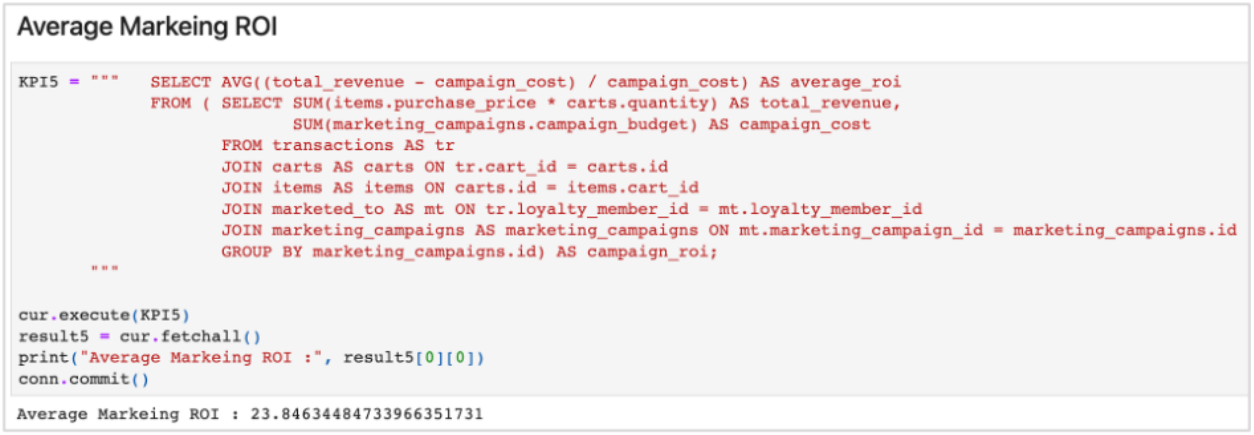
Inventory turnover ratio

```
KPI4 = """   SELECT SUM(items.purchase_price) / (AVG(inventory.inventory_stock) * (SELECT COUNT(DISTINCT date) FROM inventory)) AS inventory_turnover
             FROM items
             JOIN inventory ON items.inventory_id = inventory.id
             JOIN transactions ON items.cart_id = transactions.cart_id
             WHERE transactions.time BETWEEN '2022-01-01' AND '2022-12-31';
       """

cur.execute(KPI4)
result4 = cur.fetchall()
print("Inventory turnover ratio :", result4[0][0])
conn.commit()
```

```
Inventory turnover ratio : 1.11819029513831745281
```

5. <u>Marketing Budget Return on Investment (ROI):</u> calculated as the marketing campaign expenditures compared to the supermarket's total revenue for a given period of time. This gives the supermarket an indication of how effective a specific marketing campaign is in generating revenues. Figure 5 below shows the ROI for all marketing campaigns and the total revenue generated.

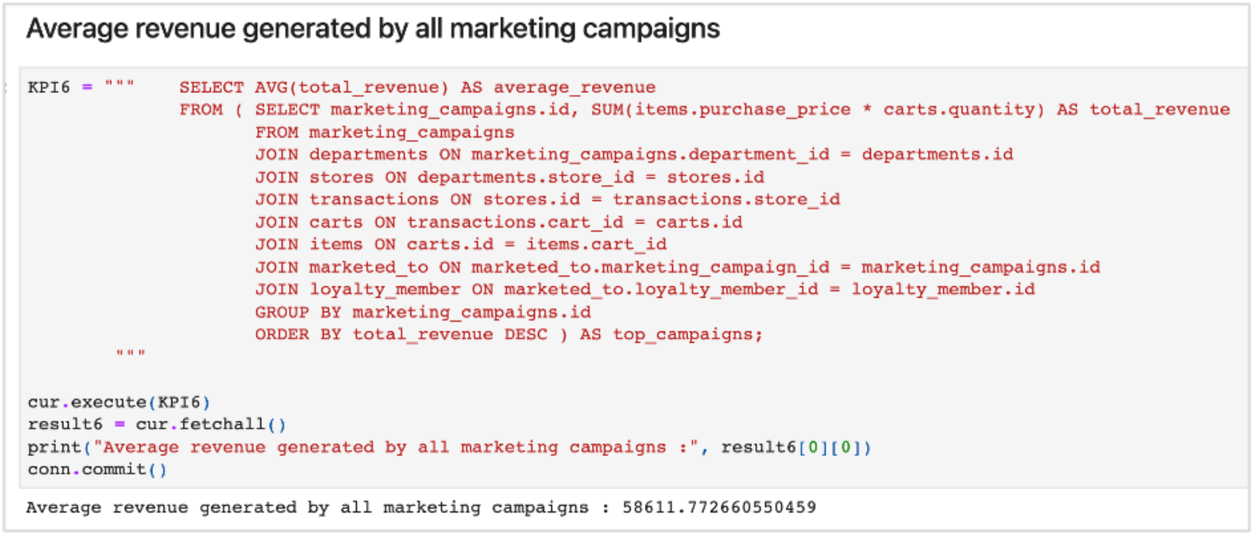
Figure 5 – Marketing Campaigns ROI



Average Markeing ROI

```
KPI5 = """   SELECT AVG((total_revenue - campaign_cost) / campaign_cost) AS average_roi
             FROM ( SELECT SUM(items.purchase_price * carts.quantity) AS total_revenue,
                           SUM(marketing_campaigns.campaign_budget) AS campaign_cost
                    FROM transactions AS tr
                    JOIN carts AS carts ON tr.cart_id = carts.id
                    JOIN items AS items ON carts.id = items.cart_id
                    JOIN marketed_to AS mt ON tr.loyalty_member_id = mt.loyalty_member_id
                    JOIN marketing_campaigns AS marketing_campaigns ON mt.marketing_campaign_id = marketing_campaigns.id
                    GROUP BY marketing_campaigns.id) AS campaign_roi;
       """

cur.execute(KPI5)
result5 = cur.fetchall()
print("Average Markeing ROI :", result5[0][0])
conn.commit()
```

```
Average Markeing ROI : 23.84634484733966351731
```



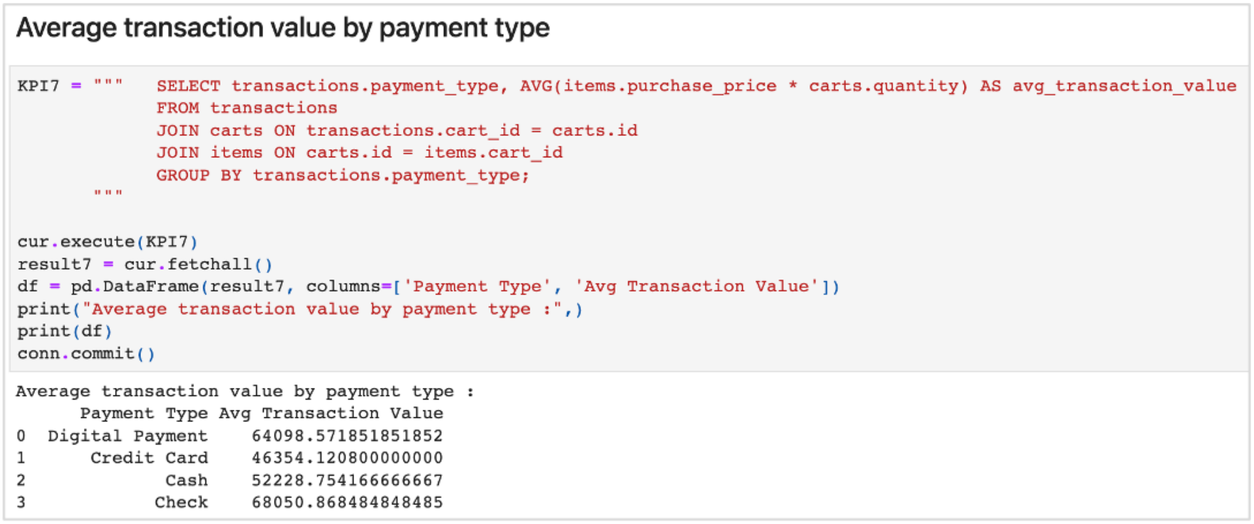
Average revenue generated by all marketing campaigns

```
KPI6 = """    SELECT AVG(total_revenue) AS average_revenue
              FROM ( SELECT marketing_campaigns.id, SUM(items.purchase_price * carts.quantity) AS total_revenue
                     FROM marketing_campaigns
                     JOIN departments ON marketing_campaigns.department_id = departments.id
                     JOIN stores ON departments.store_id = stores.id
                     JOIN transactions ON stores.id = transactions.store_id
                     JOIN carts ON transactions.cart_id = carts.id
                     JOIN items ON carts.id = items.cart_id
                     JOIN marketed_to ON marketed_to.marketing_campaign_id = marketing_campaigns.id
                     JOIN loyalty_member ON marketed_to.loyalty_member_id = loyalty_member.id
                     GROUP BY marketing_campaigns.id
                     ORDER BY total_revenue DESC ) AS top_campaigns;
        """

cur.execute(KPI6)
result6 = cur.fetchall()
print("Average revenue generated by all marketing campaigns :", result6[0][0])
conn.commit()
```

```
Average revenue generated by all marketing campaigns : 58611.772660550459
```

6. Revenue by Payment Type: calculated as the total revenue by each payment type. This metric serves as an indicator for the supermarket of how customers prefer to pay for their groceries. This information can help the supermarket to target marketing campaigns, negotiate with credit card companies or justify investments in new payment methods such as digital payment. Figure 6 below shows a breakdown of average revenue per payment type.

Figure 6 - Average Revenue Per Payment Type



Average transaction value by payment type

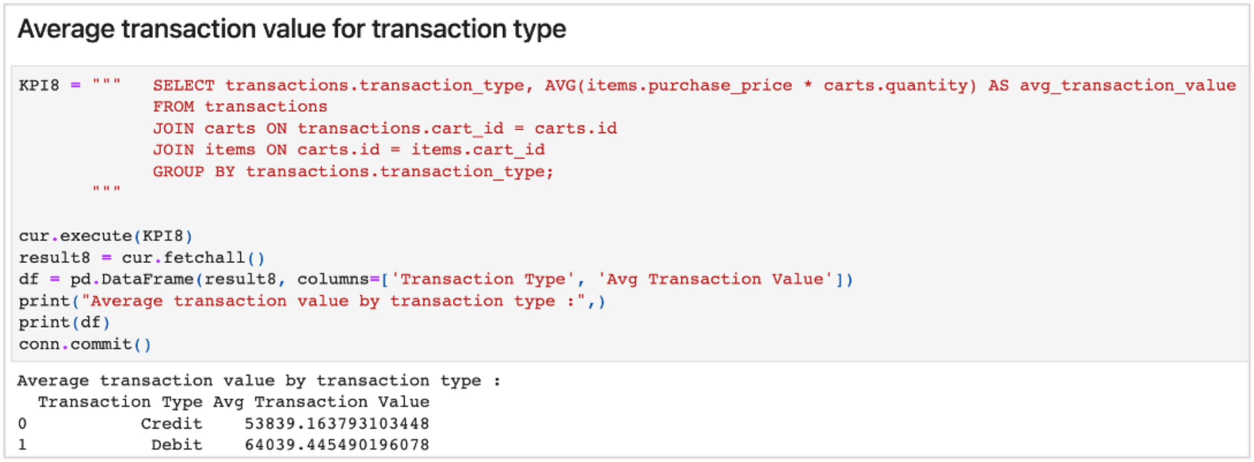
```
KPI7 = """   SELECT transactions.payment_type, AVG(items.purchase_price * carts.quantity) AS avg_transaction_value
             FROM transactions
             JOIN carts ON transactions.cart_id = carts.id
             JOIN items ON carts.id = items.cart_id
             GROUP BY transactions.payment_type;
       """

cur.execute(KPI7)
result7 = cur.fetchall()
df = pd.DataFrame(result7, columns=['Payment Type', 'Avg Transaction Value'])
print("Average transaction value by payment type :",)
print(df)
conn.commit()
```

```
Average transaction value by payment type :
     Payment Type Avg Transaction Value
0  Digital Payment    64098.571851851852
1      Credit Card    46354.120800000000
2             Cash    52228.754166666667
3            Check    68050.868484848485
```

7. Revenue by Transaction Type: calculated as the total revenue by each transaction type (debit or credit). This metric serves as an indicator for the supermarket of how customers prefer to pay for their groceries. This information

can help the supermarket to target marketing campaigns or negotiate with credit card companies. Figure 7 below shows a breakdown of average revenue per transaction type.

Figure 7 – Average Revenue By Transaction Type



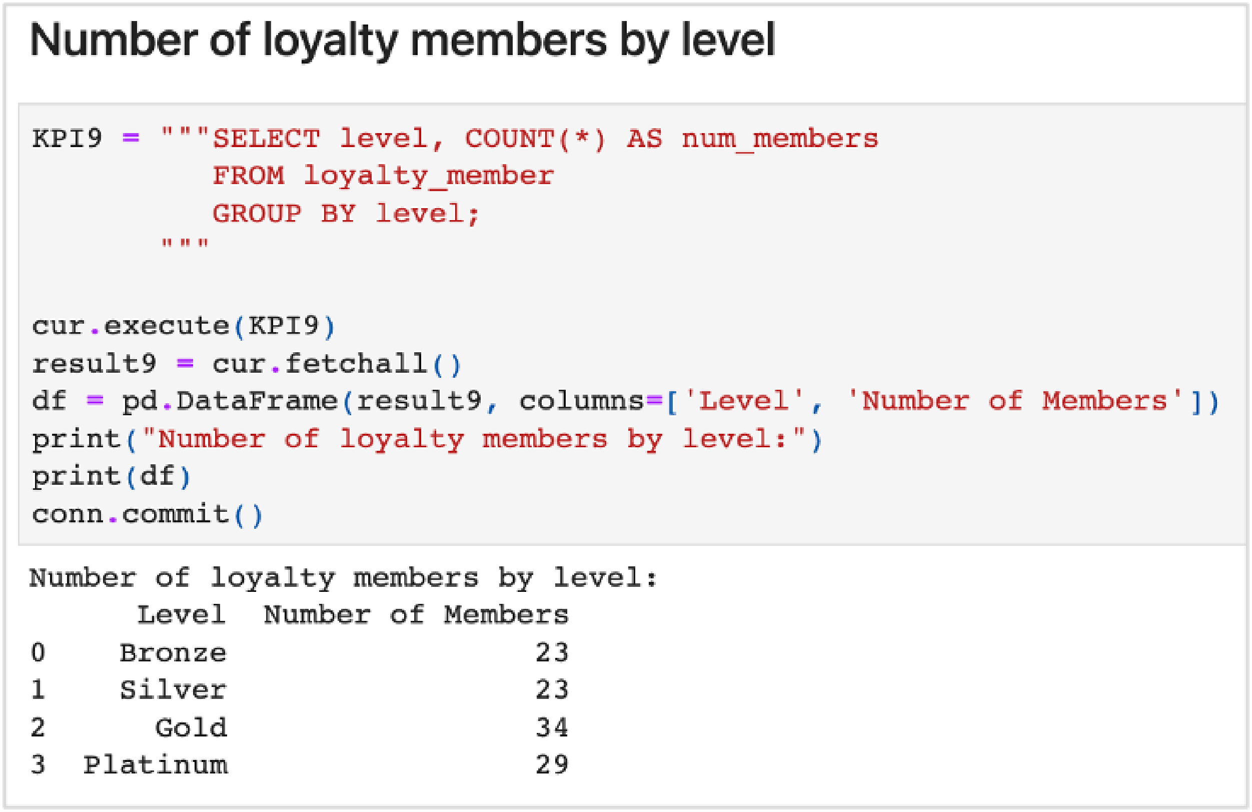
Average transaction value for transaction type

```
KPI8 = """   SELECT transactions.transaction_type, AVG(items.purchase_price * carts.quantity) AS avg_transaction_value
             FROM transactions
             JOIN carts ON transactions.cart_id = carts.id
             JOIN items ON carts.id = items.cart_id
             GROUP BY transactions.transaction_type;
       """

cur.execute(KPI8)
result8 = cur.fetchall()
df = pd.DataFrame(result8, columns=['Transaction Type', 'Avg Transaction Value'])
print("Average transaction value by transaction type :",)
print(df)
conn.commit()
```

```
Average transaction value by transaction type :
  Transaction Type Avg Transaction Value
0           Credit    53839.163793103448
1            Debit    64039.445490196078
```

8. <u>Number of loyalty program customers by loyalty level:</u> calculated as the count of customers for each loyalty level within the supermarket's customer loyalty program. This metric helps the supermarket understand effectiveness of the loyalty program and track changes over time. The most loyal customers are repeat customers and must be incentivized to continue to shop at our supermarket. Figure 8 below shows the total number of customers in each loyalty level.

Figure 8 – Number of Customers per Loyalty Level



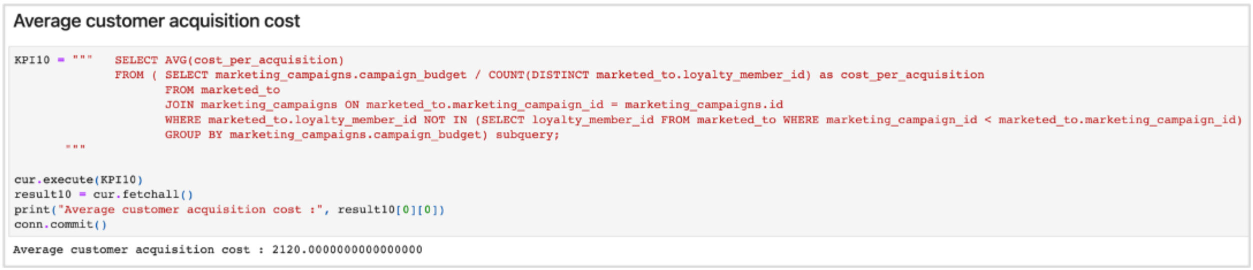
Number of loyalty members by level

```
KPI9 = """SELECT level, COUNT(*) AS num_members
          FROM loyalty_member
          GROUP BY level;
       """

cur.execute(KPI9)
result9 = cur.fetchall()
df = pd.DataFrame(result9, columns=['Level', 'Number of Members'])
print("Number of loyalty members by level:")
print(df)
conn.commit()
```

```
Number of loyalty members by level:
      Level  Number of Members
0    Bronze                 23
1    Silver                 23
2      Gold                 34
3  Platinum                 29
```

9. <u>Customer Acquisition Cost:</u> calculated as the money spent in marketing campaigns per customer that becomes a loyalty member. This metric helps the supermarket understand the effectiveness of its marketing campaigns in creating loyal customers as measured by the number of customers that join the supermarket's loyalty program, which can result in additional revenue over time. Figure 9 below shows the customer acquisition cost.

Figure 9 – Customer Acquisition Cost



Average customer acquisition cost

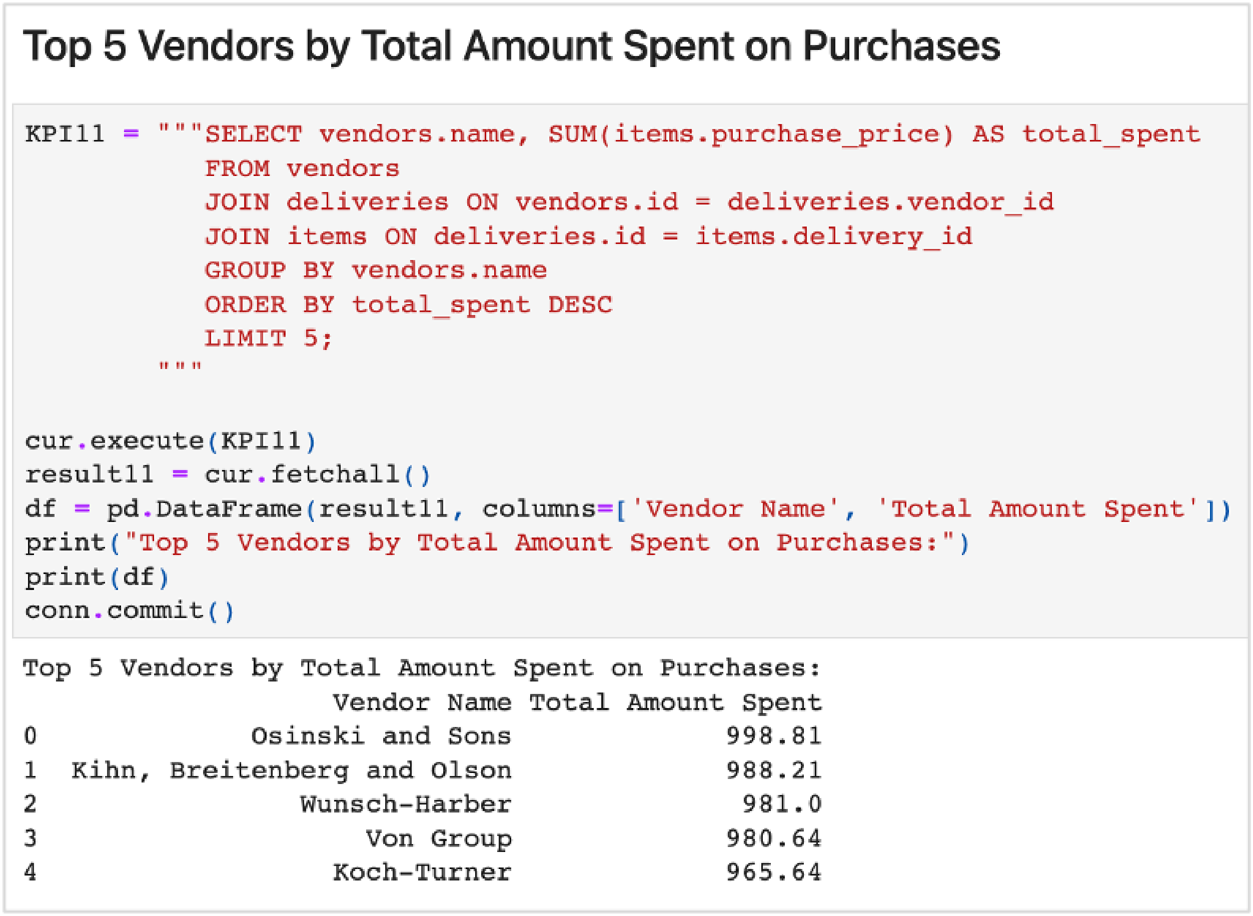
```
KPI10 = """   SELECT AVG(cost_per_acquisition)
              FROM ( SELECT marketing_campaigns.campaign_budget / COUNT(DISTINCT marketed_to.loyalty_member_id) as cost_per_acquisition
                     FROM marketed_to
                     JOIN marketing_campaigns ON marketed_to.marketing_campaign_id = marketing_campaigns.id
                     WHERE marketed_to.loyalty_member_id NOT IN (SELECT loyalty_member_id FROM marketed_to WHERE marketing_campaign_id < marketed_to.marketing_campaign_id)
                     GROUP BY marketing_campaigns.campaign_budget) subquery;
        """

cur.execute(KPI10)
result10 = cur.fetchall()
print("Average customer acquisition cost :", result10[0][0])
conn.commit()
```

```
Average customer acquisition cost : 2120.0000000000000000
```

10. <u>Top Vendors:</u> calculated as the total revenue generated by the supermarket from a specific product vendor. This metric helps the supermarket identify and track its top vendors for relationship management with those vendors, but also

advertisement strategy development. Figure 10 below shows the top 5 vendors for the supermarket.

Figure 10 – Top Five Vendors



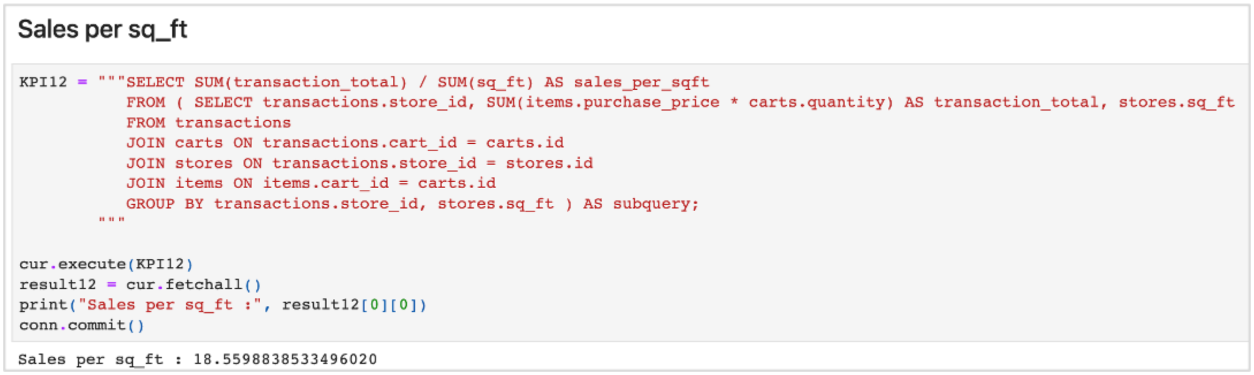
Top 5 Vendors by Total Amount Spent on Purchases

```
KPI11 = """SELECT vendors.name, SUM(items.purchase_price) AS total_spent
           FROM vendors
           JOIN deliveries ON vendors.id = deliveries.vendor_id
           JOIN items ON deliveries.id = items.delivery_id
           GROUP BY vendors.name
           ORDER BY total_spent DESC
           LIMIT 5;
        """

cur.execute(KPI11)
result11 = cur.fetchall()
df = pd.DataFrame(result11, columns=['Vendor Name', 'Total Amount Spent'])
print("Top 5 Vendors by Total Amount Spent on Purchases:")
print(df)
conn.commit()
```

```
Top 5 Vendors by Total Amount Spent on Purchases:
                   Vendor Name Total Amount Spent
0              Osinski and Sons             998.81
1  Kihn, Breitenberg and Olson             988.21
2               Wunsch-Harber              981.0
3                   Von Group             980.64
4                 Koch-Turner             965.64
```

11. Sales per square foot: calculated as the total revenue per store area. This metric helps the supermarket keep track of the differences between its locations, establish comparisons and take actions that leverage the best lessons learned across their stores. Figure 11 below shows the total sales per square foot at one of the stores.

Figure 11 – Total Sales per Square Foot


Sales per sq_ft

```
KPI12 = """SELECT SUM(transaction_total) / SUM(sq_ft) AS sales_per_sqft
           FROM ( SELECT transactions.store_id, SUM(items.purchase_price * carts.quantity) AS transaction_total, stores.sq_ft
           FROM transactions
           JOIN carts ON transactions.cart_id = carts.id
           JOIN stores ON transactions.store_id = stores.id
           JOIN items ON items.cart_id = carts.id
           GROUP BY transactions.store_id, stores.sq_ft ) AS subquery;
        """

cur.execute(KPI12)
result12 = cur.fetchall()
print("Sales per sq_ft :", result12[0][0])
conn.commit()
```

```
Sales per sq_ft : 18.5598838533496020
```

How Analysts and Management will Interact with the Database

For our analysts, we propose two ways in which they can access the database. First, they can directly access the PostgreSQL database using pgAdmin4 or Jupyter Notebook with the psycopg2 package. This method allows them to see the entire set of tables, join tables through complex queries, and perform any necessary aggregations to extract insights.

Alternatively, analysts can write queries using Metabase, a more user-friendly approach to obtaining analytical insights. While this approach does have some limitations compared to using raw SQL input into a GUID, the analysts can ask Metabase questions and get valuable insights without having to write complex SQL queries.

Lastly, we expect management will want a suite of metrics they can pore over without having to write any code. We therefore propose they leverage the dashboard we built, which contains the core KPIs upon which they measure the success of ABC Foodmart. Overall, our goal is to provide our analysts and C-level officers with easy-to-use analytical tools that are fit for purpose. Analysts' purposes may require more complex query formation, while C-level executives require an easy way in which to consume insights.

Enabling Redundancy and Performance

Hosting storage on-premises means the database infrastructure is installed locally and onsite. As a result, the company can control, administer, and maintain the lifecycle of servers and data shared through the local network. This helps to better ensure infrastructure security, as there is no vendor risk (Diamond, 2020). By contrast, cloud storage is hosted by third-party providers. These providers install and maintain all hardware and software in their data centers. As a result, cloud storage provides easier scalability (pay for what you need and no more and cost savings. Additionally, thes providers backup data regularly and offer features and applications that may be needed, as we covered in class as part of the overview of the Google suite of cloud products. (Morefield, 2022). Below are additional considerations that informed our ultimate recommendation to ABC Foodmart on whether the database should be hosted on premise or in the cloud.

Costs and Maintenance

On-premises storage requires a sizable upfront investment in hardware, software licensing fees, data backups, electricity costs, and IT experts for ongoing support, maintenance, and security. Also, this may require additional investment for ongoing expenses to replace or update the software. For smaller businesses, the investments may prove prohibitively large. By contrast, cloud storage has no upfront investment, as the company only needs to pay monthly subscription fees for the resources it uses. Also, cloud service providers can deliver services offsite such as software maintenance, up-to-date software, and data security. Cloud storage really is storage as a service, which is an efficient use of capital given you only pay for what you need.

Security

As hosting on-premises keeps all the physical assets (hardware and software systems) onsite and under direct company control, the company assumes no vendor risk. That said, the company hosting on premises needs to ensure security best practices were included as part of the implementation and there is the in-house expertise required to monitor and maintain the assets. Alternatively, cloud service providers offer comprehensive security features such as access control systems, network protection and security, disaster recovery, data redundancy, and suspicious login and activity monitoring. It is worth noting that security is among the top priorities for cloud storage providers, as they have many tenants using their systems. A security breach that threatens their tenants' data would result in devastating reputational and regulatory damage. (Cleo, n.d.).

Compliance

It is important to ensure that the company complies with regulations when hosting on-premises, such as configuring and maintaining the system and having the right resources and employees who have specialized knowledge and are familiar with compliance rules. Cloud providers have teams of experts with compliance certifications in different industries. As the company is responsible for the consequences of non-compliance, the company must perform due diligence on cloud providers and ensure the security of sensitive customer data (Cleo, n.d.).

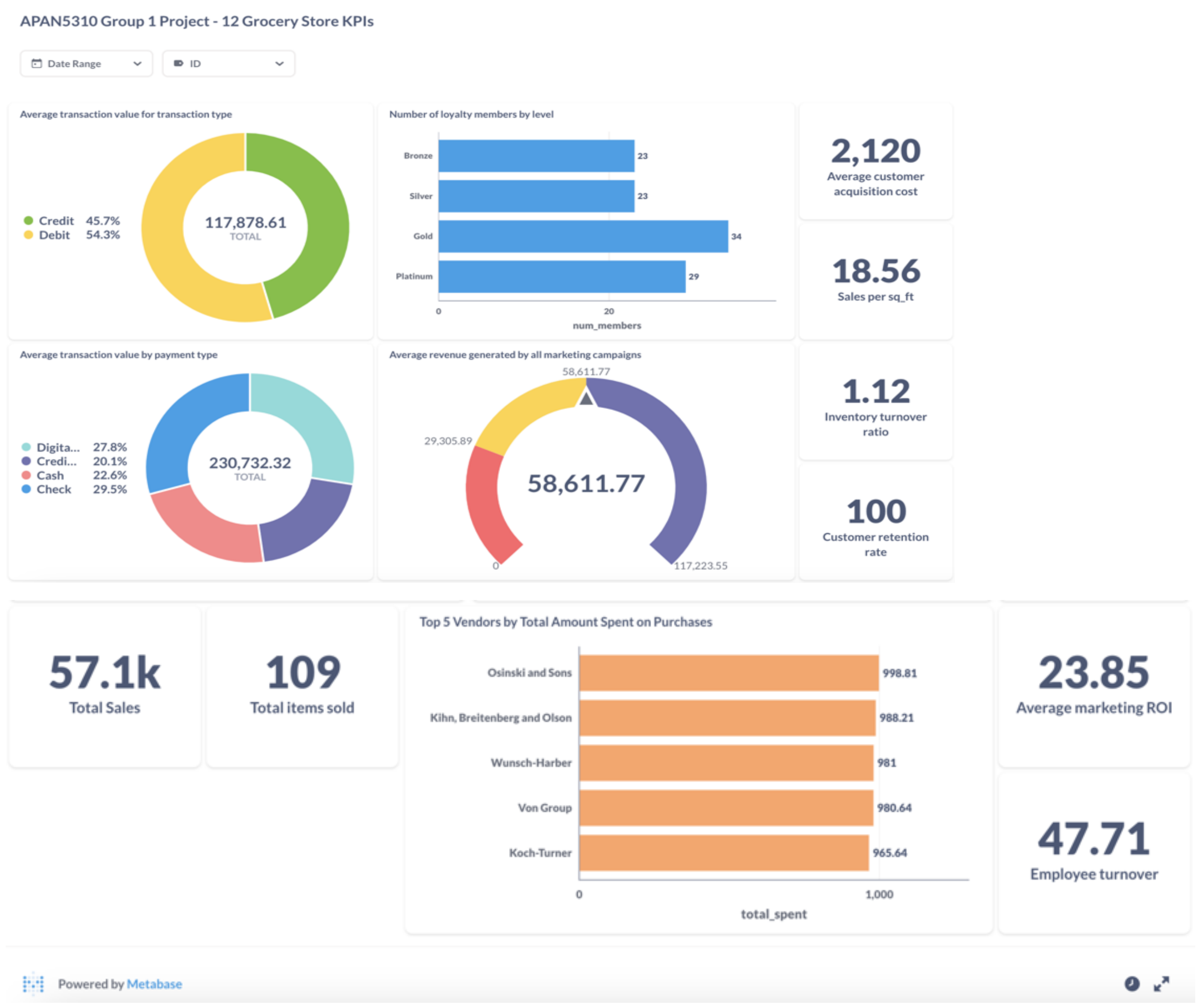
Scalability

When there is a workload problem and poor performance, the on-premises storage will need to scale up through the addition of server capacity. Additionally, the company can invest in more powerful tools that increase memory and computing power, though there are limits to this vertical scaling approach (Diamond, 2020) In comparison, cloud storage offers the flexibility to scale up when required. While scalability isn't free, cloud storage providers offer guidance on best practices that help improve performance, provide additional storage when needed, and, as previously mentioned, additional features that may prove helpful (machine learning applications, integration services, etc.)

Our Recommendation

To ensure redundancy and the ability to scale, we recommend that ABC Foodmart uses cloud storage. By partnering with a cloud storage provider, the company can scale its database horizontally and on demand, should it be necessary. With this cloud storage provider, we will pursue the best practices of data replication and sharding to mitigate the risk of data loss and increase the efficiency with which the data are accessed. We believe this is superior to the alternative of scaling vertically.

Data Visualization



We leveraged Metabase to provide analytical outputs for the C-level officers at our grocery store chain. Metabase facilitates the visual representation of the data resident within our PostgreSQL database. Above is an example dashboard that enables management to extract insights that can help in guiding the company's strategic direction. The dashboard components are built based on key performance indicators (KPIs) that we identified during the project's initial stages. These include employee turnover, total sales, and inventory turnover.

You can also access the dashboard through the below public link: http://localhost:3000/public/dashboard/86a389ba-2aa3-4424-a62d-c421c1ea81ac

Conclusion

Our goal in engaging with ABC Foodmart was to modernize their organization in an effort to enable them to scale to three additional locations in the Brooklyn area. In achieving this goal, we designed their database, imported their data, recommended cloud infrastructure, developed and scripted key performance indicators, and produced management-level dashboards. Our client should feel secure in its ability to realize a successful expansion and recognize its ability to further expand given the sustainable database infrastructure. With the ability to monitor performance and respond to metrics made visible through the dashboards, those at ABC Foodmart are now on the trajectory required for a successful future.

Bibliography:

Diamond, P. (2020, September 25). *Cloud storage vs. on-premises servers: 9 things to keep in mind*. Microsoft. Retrieved April 29, 2023, from https://www.microsoft.com/en-us/microsoft-365/business-insights-ideas/resources/cloud-storage-vs-on-premises-servers

Morefield. (2022, May 27). On-Premises vs. Cloud. Morefield. Retrieved April 29, 2023, from https://morefield.com/blog/on-premises-vs-cloud/

Cleo. (n.d.). Blog: On Premise vs. Cloud: Key Differences, Benefits and Risks | Cleo. Retrieved April 29, 2023, from https://www.cleo.com/blog/knowledge-base-on-premise-vs-cloud