Use Case Study Report

Group #1

Lei Wu & Edward Brown

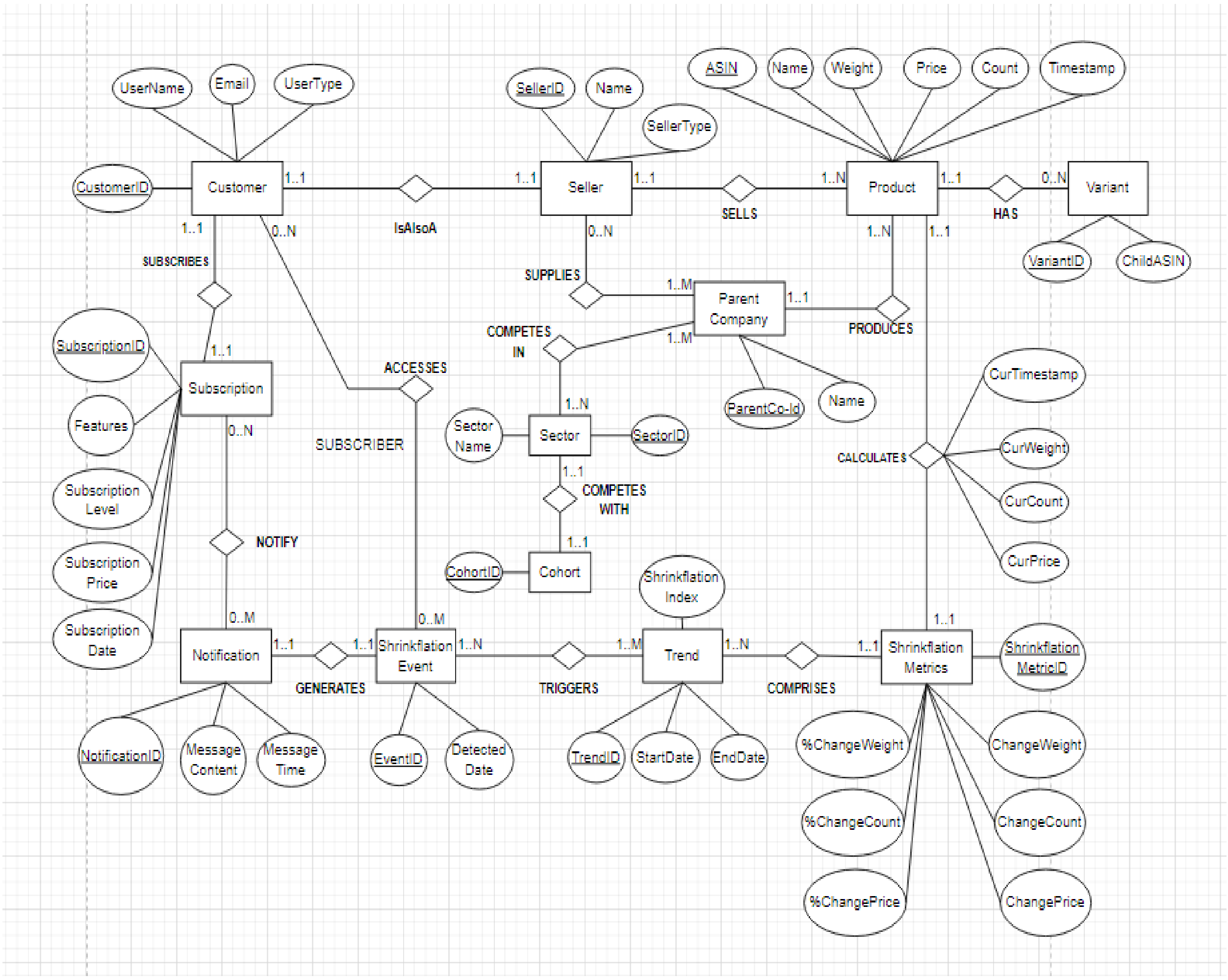
Project Overview

The recent studies conducted by Retail Insight highlighted an increase in price awareness among consumers, particularly for grocery items, since the onset of the pandemic(source: Price-conscious consumers hunger for lower food tab | Supermarket News). In response, numerous companies and stores have used a tactic known as "shrinkflation," subtly reducing a product size while maintaining price, to address the consumers' heightened price sensitivity. This approach helps many businesses manage escalating production costs and preserve competitive pricing and profit margins, often without clear disclosure to the consumers. Our initiative is creating a digital platform dedicated to tracking shrinkflation for those consumers who value transparency and informed grocery shopping. We primarily focus on the online shopping platform Amazon.

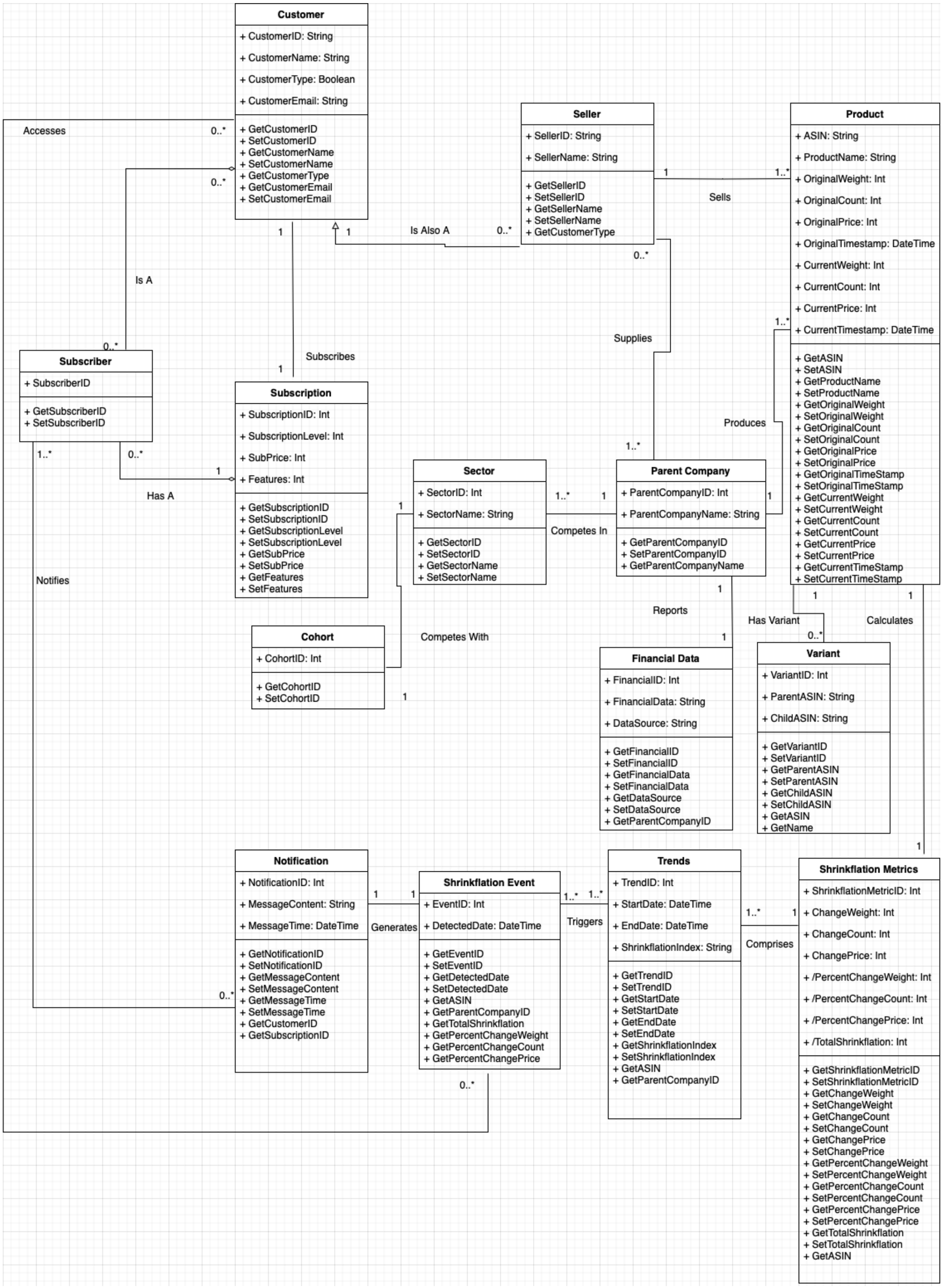
Our platform will also be capable of providing data to companies and third-party Amazon resellers. The metrics we provide will be available for companies to analyze and utilize for tactical and strategic purposes. Competitors can analyze peer firm's results and inform their decisions. Companies in the same sectors can evaluate their cohort and determine whether their products should be shrinkflated more or less based on their competitor's results. Economists can gather real-world data on consumer tolerance for shrinkflation. Amazon resellers can track which products and sectors have more resilient consumer demand and can withstand shrinkflation. This information could potentially increase profitability for companies and resellers.

Conceptual Data Modeling

EER Diagram



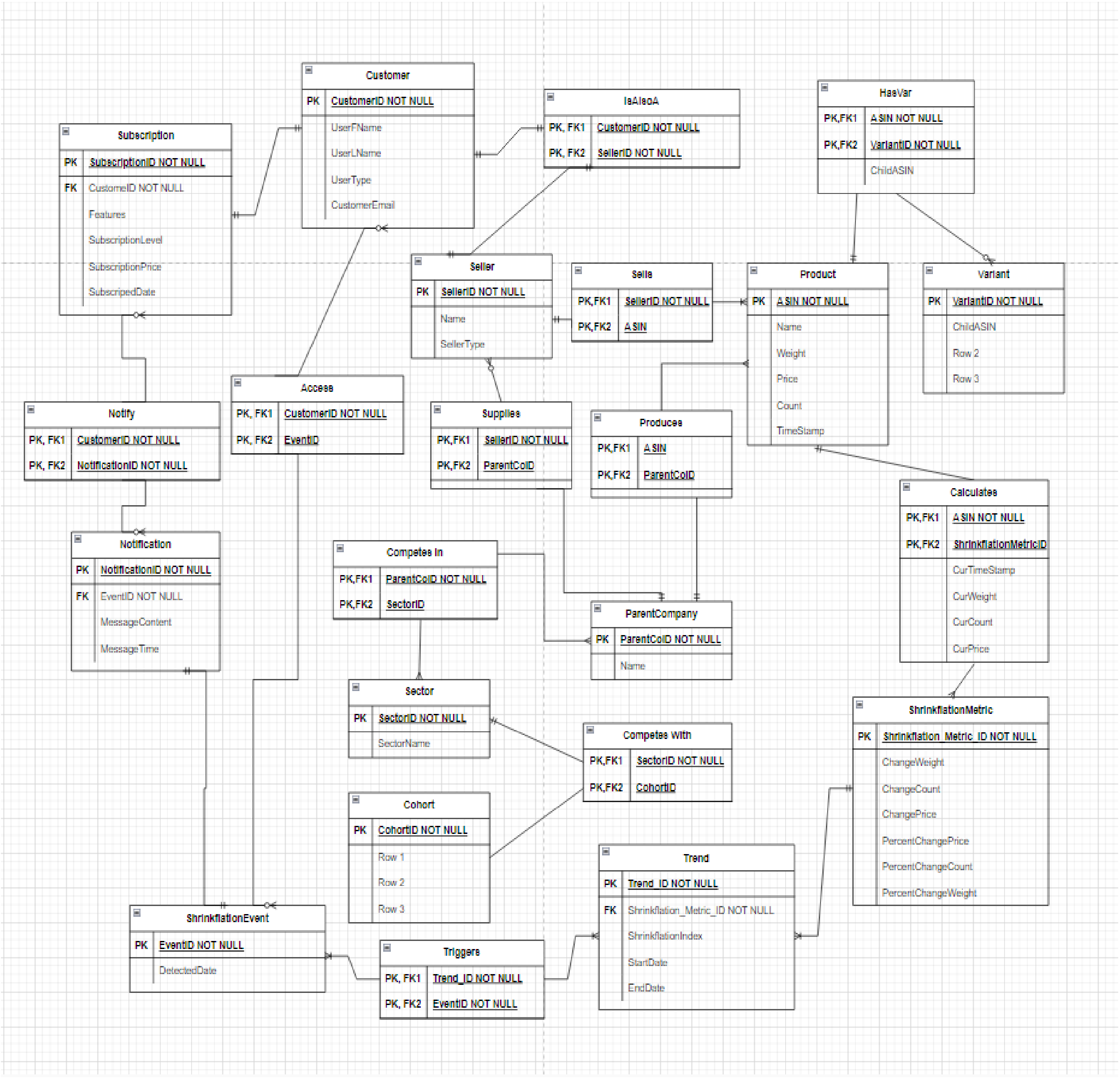
UML Diagram


Customer
+ CustomerID: String
+ CustomerName: String
+ CustomerType: Boolean
+ CustomerEmail: String
+ GetCustomerID
+ SetCustomerID
+ GetCustomerName
+ SetCustomerName
+ GetCustomerType
+ GetCustomerEmail
+ SetCustomerEmail
Seller
+ SellerID: String
+ SellerName: String
+ GetSellerID
+ SetSellerID
+ GetSellerName
+ SetSellerName
+ GetCustomerType
Product
+ ASIN: String
+ ProductName: String
+ OriginalWeight: Int
+ OriginalCount: Int
+ OriginalPrice: Int
+ OriginalTimestamp: DateTime
+ CurrentWeight: Int
+ CurrentCount: Int
+ CurrentPrice: Int
+ CurrentTimestamp: DateTime
+ GetASIN
+ SetASIN
+ GetProductName
+ SetProductName
+ GetOriginalWeight
+ SetOriginalWeight
+ GetOriginalCount
+ SetOriginalCount
+ GetOriginalPrice
+ SetOriginalPrice
+ GetOriginalTimeStamp
+ SetOriginalTimeStamp
+ GetCurrentWeight
+ SetCurrentWeight
+ GetCurrentCount
+ SetCurrentCount
+ GetCurrentPrice
+ SetCurrentPrice
+ GetCurrentTimeStamp
+ SetCurrentTimeStamp
Accesses
0..*
0..*
1
Is Also A
0..*
1
1..*
Sells
0..*
Is A
Supplies
0..*
1
Subscribes
1
1..*
Produces
Subscriber
+ SubscriberID
+ GetSubscriberID
+ SetSubscriberID
Subscription
+ SubscriptionID: Int
+ SubscriptionLevel: Int
+ SubPrice: Int
+ Features: Int
+ GetSubscriptionID
+ SetSubscriptionID
+ GetSubscriptionLevel
+ SetSubscriptionLevel
+ GetSubPrice
+ SetSubPrice
+ GetFeatures
+ SetFeatures
1..*
0..*
1
Has A
Sector
+ SectorID: Int
+ SectorName: String
+ GetSectorID
+ SetSectorID
+ GetSectorName
+ SetSectorName
1..*
1
Competes In
1
Parent Company
+ ParentCompanyID: Int
+ ParentCompanyName: String
+ GetParentCompanyID
+ SetParentCompanyID
+ GetParentCompanyName
1..*
1
Notifies
1
Reports
1
1
Has Variant
Calculates
0..*
Cohort
+ CohortID: Int
+ GetCohortID
+ SetCohortID
Competes With
1
Financial Data
+ FinancialID: Int
+ FinancialData: String
+ DataSource: String
+ GetFinancialID
+ SetFinancialID
+ GetFinancialData
+ SetFinancialData
+ GetDataSource
+ SetDataSource
+ GetParentCompanyID
1
Variant
+ VariantID: Int
+ ParentASIN: String
+ ChildASIN: String
+ GetVariantID
+ SetVariantID
+ GetParentASIN
+ SetParentASIN
+ GetChildASIN
+ SetChildASIN
+ GetASIN
+ GetName
1
Notification
+ NotificationID: Int
+ MessageContent: String
+ MessageTime: DateTime
+ GetNotificationID
+ SetNotificationID
+ GetMessageContent
+ SetMessageContent
+ GetMessageTime
+ SetMessageTime
+ GetCustomerID
+ GetSubscriptionID
0..*
1
1
Generates
Shrinkflation Event
+ EventID: Int
+ DetectedDate: DateTime
+ GetEventID
+ SetEventID
+ GetDetectedDate
+ SetDetectedDate
+ GetASIN
+ GetParentCompanyID
+ GetTotalShrinkflation
+ GetPercentChangeWeight
+ GetPercentChangeCount
+ GetPercentChangePrice
0..*
1..*
1..*
Triggers
Trends
+ TrendID: Int
+ StartDate: DateTime
+ EndDate: DateTime
+ ShrinkflationIndex: String
+ GetTrendID
+ SetTrendID
+ GetStartDate
+ SetStartDate
+ GetEndDate
+ SetEndDate
+ GetShrinkflationIndex
+ SetShrinkflationIndex
+ GetASIN
+ GetParentCompanyID
1..*
1
Comprises
Shrinkflation Metrics
+ ShrinkflationMetricID: Int
+ ChangeWeight: Int
+ ChangeCount: Int
+ ChangePrice: Int
+ /PercentChangeWeight: Int
+ /PercentChangeCount: Int
+ /PercentChangePrice: Int
+ /TotalShrinkflation: Int
+ GetShrinkflationMetricID
+ SetShrinkflationMetricID
+ GetChangeWeight
+ SetChangeWeight
+ GetChangeCount
+ SetChangeCount
+ GetChangePrice
+ SetChangePrice
+ GetPercentChangeWeight
+ SetPercentChangeWeight
+ GetPercentChangeCount
+ SetPercentChangeCount
+ GetPercentChangePrice
+ SetPercentChangePrice
+ GetTotalShrinkflation
+ SetTotalShrinkflation
+ GetASIN

Mapping Conceptual Model to Relational Model

1. Customer: CustomerID (PK NOT NULL), UserFName, UserLName, UserType, CustomerEmail.
2. IsAlsoA: CustomerID (PK, FK NOT NULL), SellerID (PK, FK NOT NULL).
3. Seller: SellerID (PK NOT NULL), SellerFName, SellerLName, SellerType.
4. Sells: SellerID (PK, FK NOT NULL), ASIN (PK, FK NOT NULL).
5. Product: ASIN (PK NOT NULL), ProductName, Weight, Price, Count, TimeStamp.
6. HasVar: ASIN (PK, FK NOT NULL), VariantID (PK, FK NOT NULL).
7. Variant: VariantID (PK NOT NULL), ChildASIN.
8. Produces: ASIN (PK, FK NOT NULL), ParentCoID (PK, FK NOT NULL).
9. Supplies: SellerID (PK, FK NOT NULL), ParentCoID (PK, FK NOT NULL).
10. ParentCompany: ParentCoID (PK NOT NULL), CompanyName.
11. CompetesIn: ParentCoID (PK, FK NOT NULL), SectorID (PK, FK NOT NULL).
12. Sector: SectorID (PK NOT NULL).
13. CompetesWith: SectorID (PK, FK NOT NULL), CohortID (PK, FK NOT NULL).
14. Cohort: CohortID (PK NOT NULL).
15. Calculates: ASIN (PK, FK NOT NULL), ShrinkflationMetricID (PK, FK NOT NULL), CurTimeStamp, CurWeight, CurCount, CurPrice.
16. ShrinkflationMetric: ShrinkflationMetricID (PK NOT NULL), ChangeWeight, ChangeCount, ChangePrice, PercentChangePrice, PercentChangeCount, PercentChangeWeight.
17. Trend: TrendID (PK NOT NULL), ShrinkflationMetricID (FK NOT NULL), ShrinkflationIndex, StartDate, EndDate.
18. Triggers: TrendID (PK, FK NOT NULL), EventID (PK, FK NOT NULL).
19. ShrinkflationEvent: EventID (PK NOT NULL), DetectedDate.
20. Notification: NotificationID (PK NOT NULL), EventID (FK NOT NULL), MessageContent, MessageTime.
21. Notify: CutomerID (PK, FK NOT NULL), NotificationID (PK, FK NOT NULL).
22. Subscription: SubscriptionID (PK NOT NULL), CustomerID (FK NOT NULL), Features, SubscriptionLevel, SubscriptionPrice, SubscriptionDate.
23. Access: CustomerID (PK, FK NOT NULL), EventID (PK, FK NOT NULL).

Relational Model Diagram



Implementation of Relation Model via MySQL and NoSQL

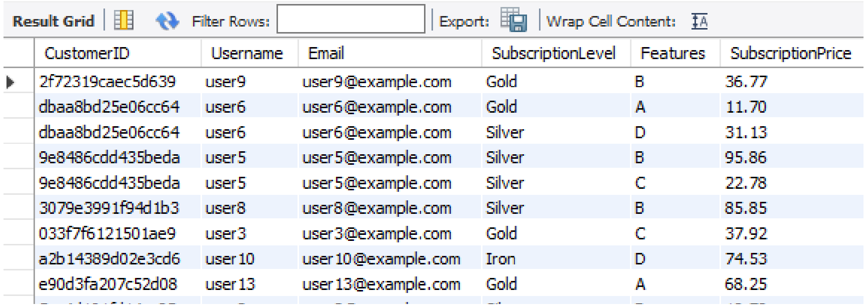
MySQL:

We implemented our database in MySQL workbench in the class VM and then performed the following queries:

SQL Queries:

1. Join customer and subscription tables to find customer details along with their subscription types

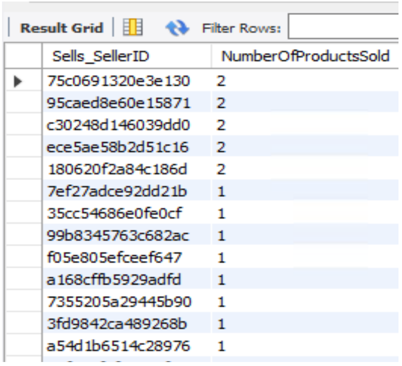
```
SELECT c.CustomerID, c.Username, c.Email, s.SubscriptionLevel, s.Features,
s.SubscriptionPrice
FROM customer c
JOIN subscription s ON c.CustomerID = s.CustomerID;
```



CustomerID	Username	Email	SubscriptionLevel	Features	SubscriptionPrice
2f72319caec5d639	user9	user9@example.com	Gold	B	36.77
dbaa8bd25e06cc64	user6	user6@example.com	Gold	A	11.70
dbaa8bd25e06cc64	user6	user6@example.com	Silver	D	31.13
9e8486cdd435beda	user5	user5@example.com	Silver	B	95.86
9e8486cdd435beda	user5	user5@example.com	Silver	C	22.78
3079e3991f94d1b3	user8	user8@example.com	Silver	B	85.85
033f7f6121501ae9	user3	user3@example.com	Gold	C	37.92
a2b14389d02e3cd6	user10	user10@example.com	Iron	D	74.53
e90d3fa207c52d08	user13	user13@example.com	Gold	A	68.25

2. Aggregate to count the number of products sold by each seller

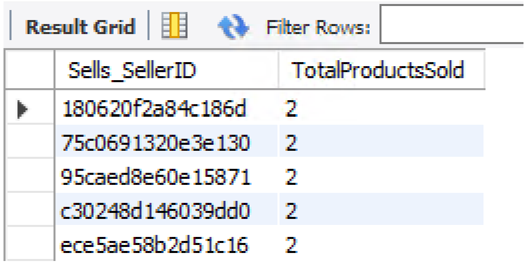
```
SELECT s.Sells_SellerID, COUNT(p.ASIN) AS NumberOfProductsSold
FROM sells s
JOIN product p ON s.Sells_ASIN = p.ASIN
GROUP BY s.Sells_SellerID;
```



	Sells_SellerID	NumberOfProductsSold
▶	75c0691320e3e130	2
	95caed8e60e15871	2
	c30248d146039dd0	2
	ece5ae58b2d51c16	2
	180620f2a84c186d	2
	7ef27adce92dd21b	1
	35cc54686e0fe0cf	1
	99b8345763c682ac	1
	f05e805efceef647	1
	a168cffb5929adfd	1
	7355205a29445b90	1
	3fd9842ca489268b	1
	a54d1b6514c28976	1

3. Nested query to find sellers who have sold more than an average number of products

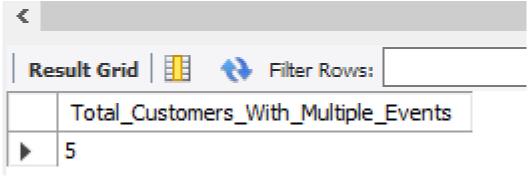
```
SELECT s.Sells_SellerID, COUNT(s.Sells_ASIN) AS TotalProductsSold
FROM sells s
GROUP BY s.Sells_SellerID
HAVING TotalProductsSold > (
        SELECT AVG(ProductsSold) FROM (
        SELECT COUNT(*) AS ProductsSold
        FROM sells
        GROUP BY Sells_SellerID
        ) AS AvgProductsSold
);
```



	Sells_SellerID	TotalProductsSold
▶	180620f2a84c186d	2
	75c0691320e3e130	2
	95caed8e60e15871	2
	c30248d146039dd0	2
	ece5ae58b2d51c16	2

4. Aggregate to find the total number of customers who have accessed more than one event

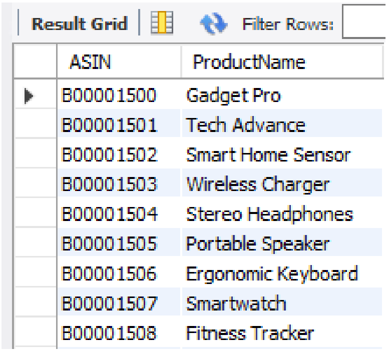
```
SELECT COUNT(*) AS Total_Customers_With_Multiple_Events
FROM (
   SELECT CustomerID
   FROM access
   GROUP BY CustomerID
   HAVING COUNT(EventID) > 1
) AS SubQuery;
```



Total_Customers_With_Multiple_Events
5

5. Inner join to determine which products have a variant

```
SELECT p.ASIN, p.ProductName
FROM product p
INNER JOIN has_variant hv ON p.ASIN = hv.ASIN
INNER JOIN variant v ON hv.VariantID = v.VariantID;
```

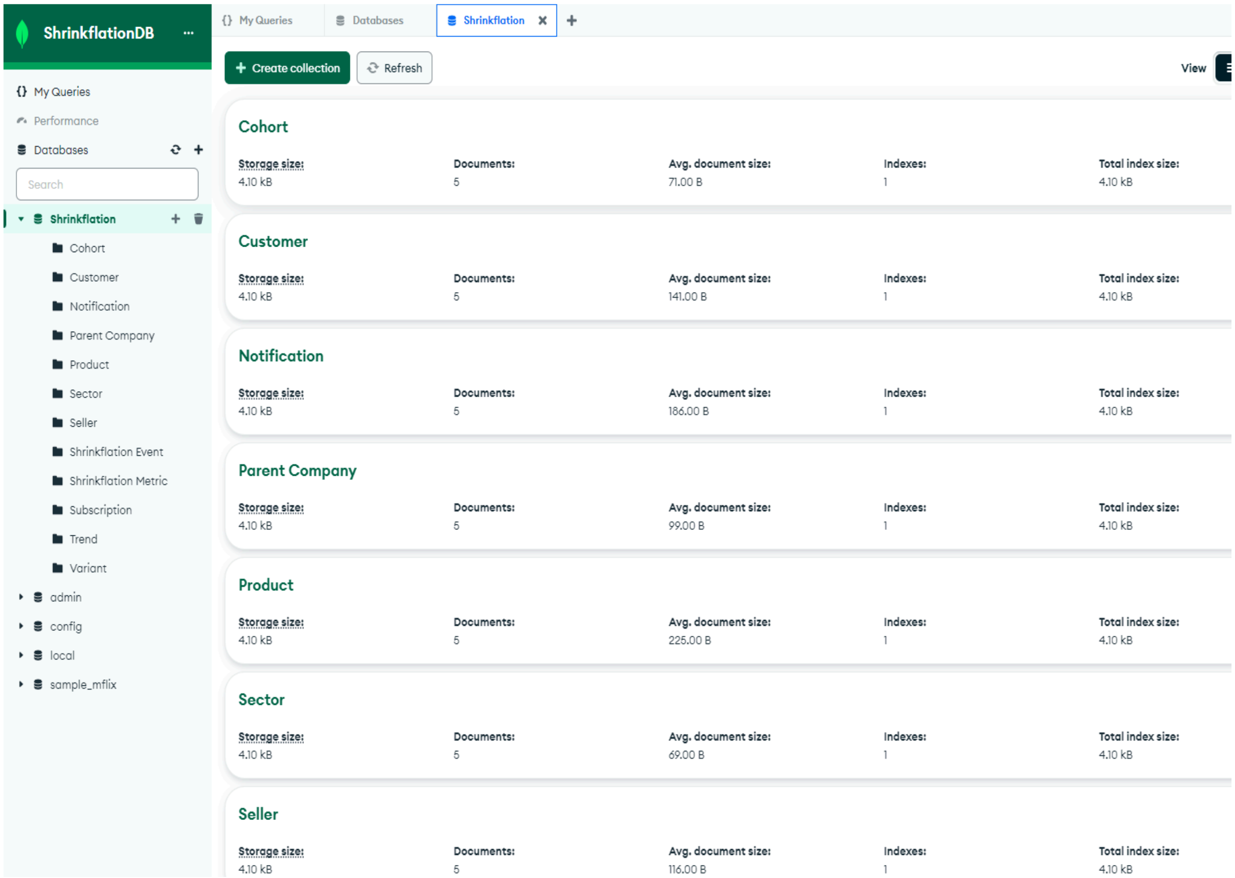


ASIN	ProductName
B00001500	Gadget Pro
B00001501	Tech Advance
B00001502	Smart Home Sensor
B00001503	Wireless Charger
B00001504	Stereo Headphones
B00001505	Portable Speaker
B00001506	Ergonomic Keyboard
B00001507	Smartwatch
B00001508	Fitness Tracker

NoSQL:

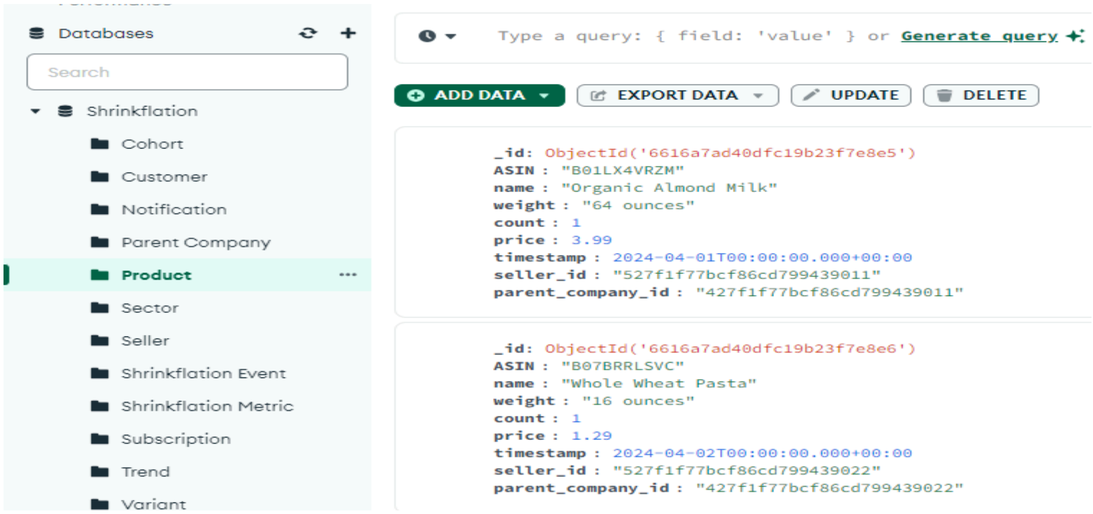
We chose to implement our database in MongoDB and used MongoDB Compass to perform the following queries:

Overview of database in MongoDB Compass:



	Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
Cohort	4.10 kB	5	71.00 B	1	4.10 kB
Customer	4.10 kB	5	141.00 B	1	4.10 kB
Notification	4.10 kB	5	186.00 B	1	4.10 kB
Parent Company	4.10 kB	5	99.00 B	1	4.10 kB
Product	4.10 kB	5	225.00 B	1	4.10 kB
Sector	4.10 kB	5	69.00 B	1	4.10 kB
Seller	4.10 kB	5	116.00 B	1	4.10 kB

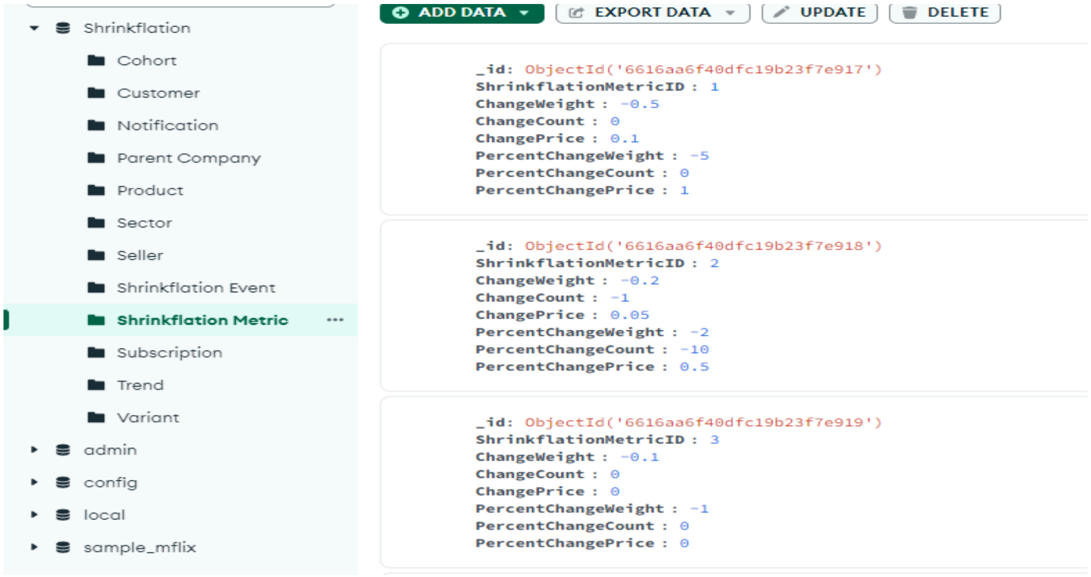
Product Collection:



```
_id: ObjectId('6616a7ad40dfc19b23f7e8e5')
ASIN : "B01LX4VRZM"
name : "Organic Almond Milk"
weight : "64 ounces"
count : 1
price : 3.99
timestamp : 2024-04-01T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439011"
parent_company_id : "427f1f77bcf86cd799439011"
```

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e6')
ASIN : "B07BRRLSVC"
name : "Whole Wheat Pasta"
weight : "16 ounces"
count : 1
price : 1.29
timestamp : 2024-04-02T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439022"
parent_company_id : "427f1f77bcf86cd799439022"
```

Shrinkflation Metric Collection:

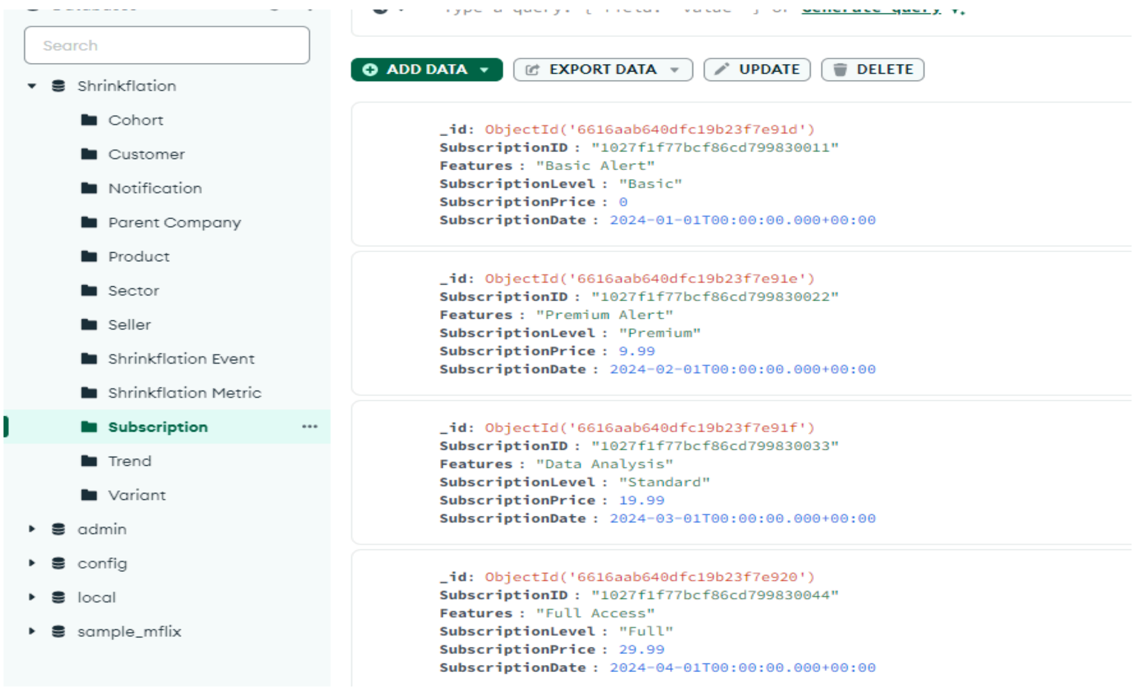


```
_id: ObjectId('6616aa6f40dfc19b23f7e917')
ShrinkflationMetricID : 1
ChangeWeight : -0.5
ChangeCount : 0
ChangePrice : 0.1
PercentChangeWeight : -5
PercentChangeCount : 0
PercentChangePrice : 1
```

```
_id: ObjectId('6616aa6f40dfc19b23f7e918')
ShrinkflationMetricID : 2
ChangeWeight : -0.2
ChangeCount : -1
ChangePrice : 0.05
PercentChangeWeight : -2
PercentChangeCount : -10
PercentChangePrice : 0.5
```

```
_id: ObjectId('6616aa6f40dfc19b23f7e919')
ShrinkflationMetricID : 3
ChangeWeight : -0.1
ChangeCount : 0
ChangePrice : 0
PercentChangeWeight : -1
PercentChangeCount : 0
PercentChangePrice : 0
```

Subscription Collection:



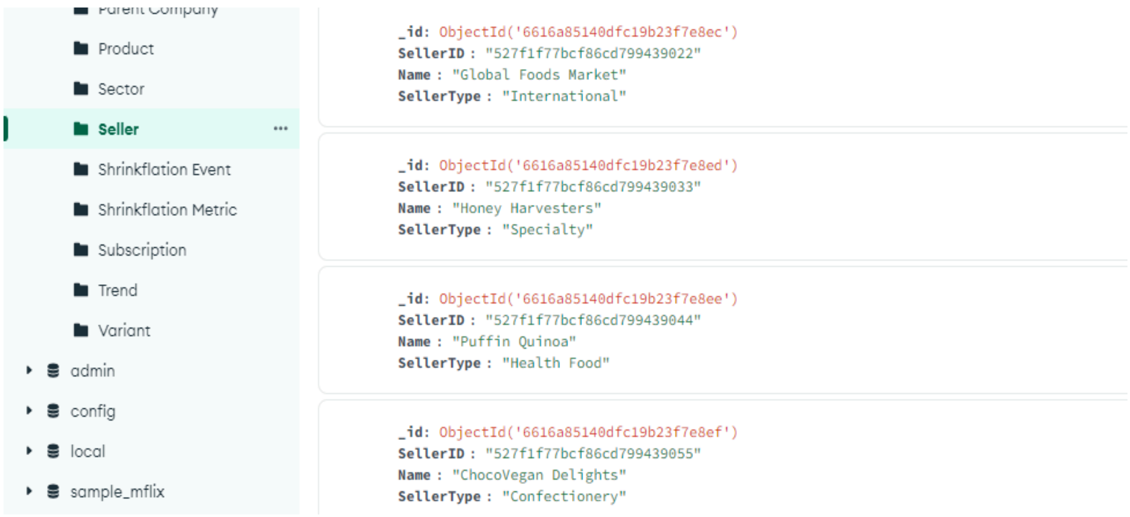
```
_id: ObjectId('6616aab640dfc19b23f7e91d')
SubscriptionID : "1027f1f77bcf86cd799830011"
Features : "Basic Alert"
SubscriptionLevel : "Basic"
SubscriptionPrice : 0
SubscriptionDate : 2024-01-01T00:00:00.000+00:00

_id: ObjectId('6616aab640dfc19b23f7e91e')
SubscriptionID : "1027f1f77bcf86cd799830022"
Features : "Premium Alert"
SubscriptionLevel : "Premium"
SubscriptionPrice : 9.99
SubscriptionDate : 2024-02-01T00:00:00.000+00:00

_id: ObjectId('6616aab640dfc19b23f7e91f')
SubscriptionID : "1027f1f77bcf86cd799830033"
Features : "Data Analysis"
SubscriptionLevel : "Standard"
SubscriptionPrice : 19.99
SubscriptionDate : 2024-03-01T00:00:00.000+00:00

_id: ObjectId('6616aab640dfc19b23f7e920')
SubscriptionID : "1027f1f77bcf86cd799830044"
Features : "Full Access"
SubscriptionLevel : "Full"
SubscriptionPrice : 29.99
SubscriptionDate : 2024-04-01T00:00:00.000+00:00
```

Seller Collection:



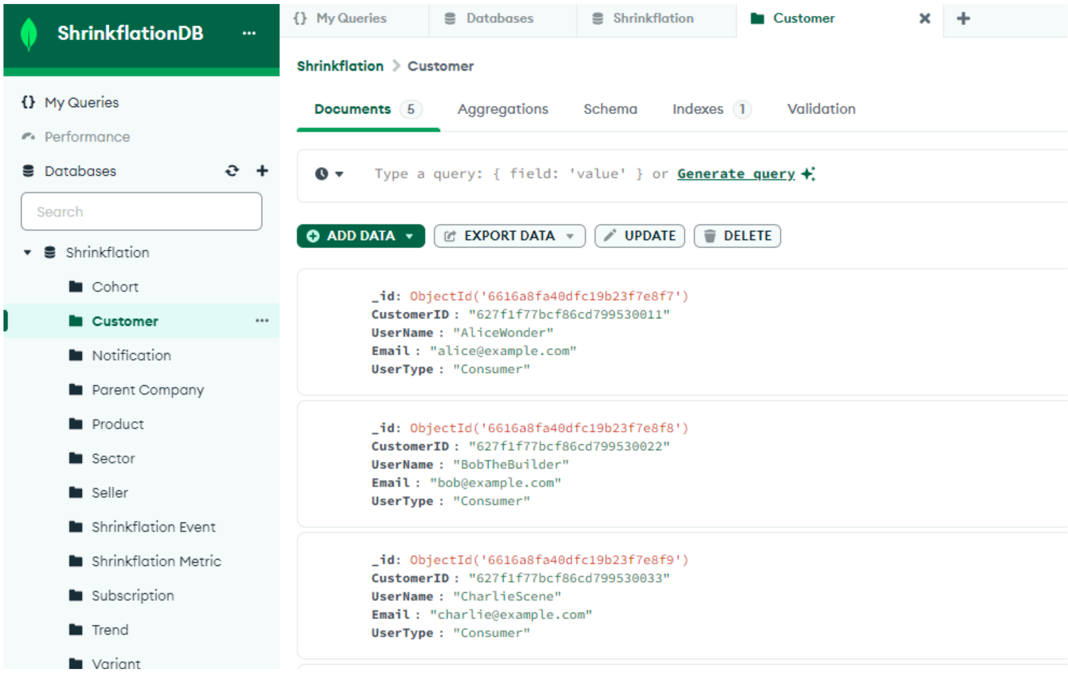
```
_id: ObjectId('6616a85140dfc19b23f7e8ec')
SellerID : "527f1f77bcf86cd799439022"
Name : "Global Foods Market"
SellerType : "International"

_id: ObjectId('6616a85140dfc19b23f7e8ed')
SellerID : "527f1f77bcf86cd799439033"
Name : "Honey Harvesters"
SellerType : "Specialty"

_id: ObjectId('6616a85140dfc19b23f7e8ee')
SellerID : "527f1f77bcf86cd799439044"
Name : "Puffin Quinoa"
SellerType : "Health Food"

_id: ObjectId('6616a85140dfc19b23f7e8ef')
SellerID : "527f1f77bcf86cd799439055"
Name : "ChocoVegan Delights"
SellerType : "Confectionery"
```

Customer Collection:



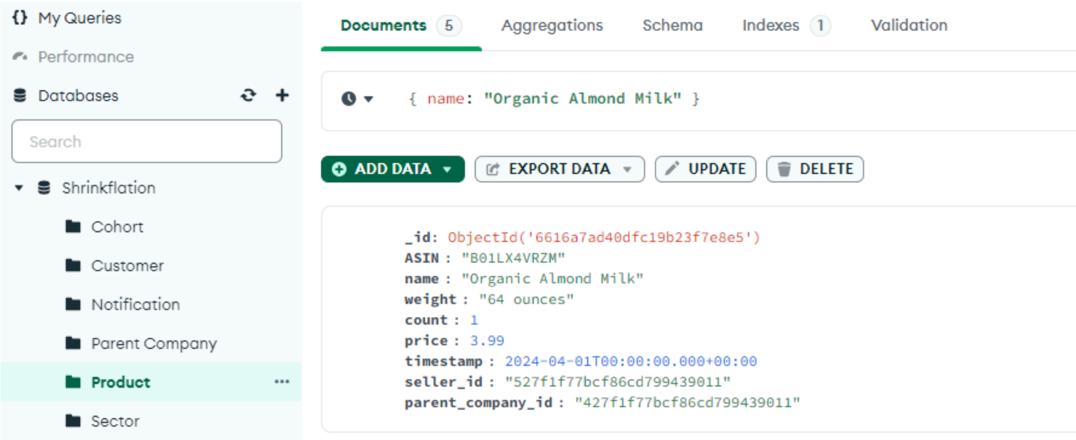
```
_id: ObjectId('6616a8fa40dfc19b23f7e8f7')
CustomerID : "627f1f77bcf86cd799530011"
UserName : "AliceWonder"
Email : "alice@example.com"
UserType : "Consumer"
```

```
_id: ObjectId('6616a8fa40dfc19b23f7e8f8')
CustomerID : "627f1f77bcf86cd799530022"
UserName : "BobTheBuilder"
Email : "bob@example.com"
UserType : "Consumer"
```

```
_id: ObjectId('6616a8fa40dfc19b23f7e8f9')
CustomerID : "627f1f77bcf86cd799530033"
UserName : "CharlieScene"
Email : "charlie@example.com"
UserType : "Consumer"
```

MongoDB Queries

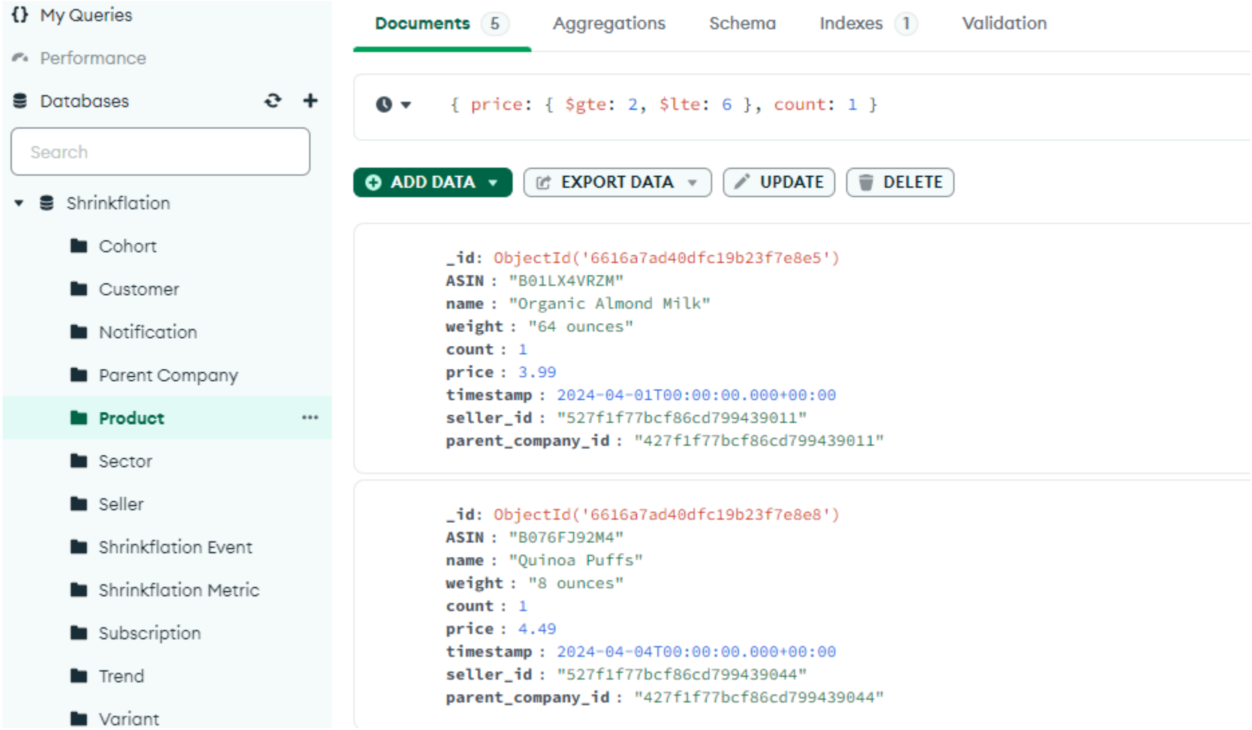
Simple Query: Find a product named “Organic Almond Milk”



```
{ name: "Organic Almond Milk" }
```

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e5')
ASIN : "B01LX4VRZM"
name : "Organic Almond Milk"
weight : "64 ounces"
count : 1
price : 3.99
timestamp : 2024-04-01T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439011"
parent_company_id : "427f1f77bcf86cd799439011"
```

Complex Query: 2 conditions price between 2 & 6 and a item count of 1

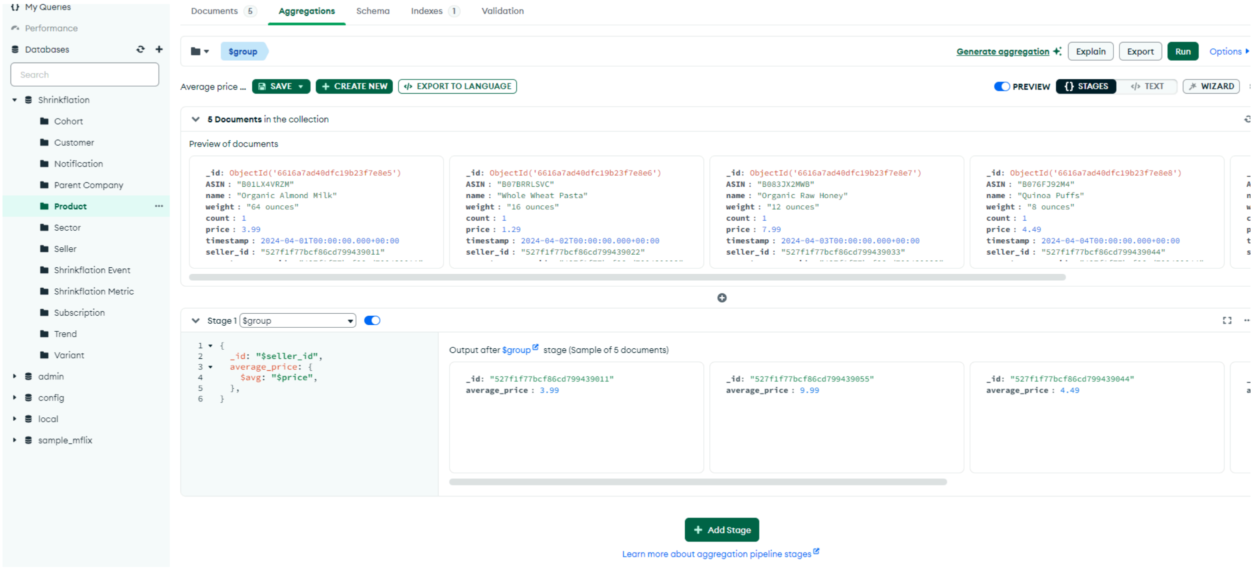


```
{ price: { $gte: 2, $lte: 6 }, count: 1 }
```

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e5')
ASIN : "B01LX4VRZM"
name : "Organic Almond Milk"
weight : "64 ounces"
count : 1
price : 3.99
timestamp : 2024-04-01T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439011"
parent_company_id : "427f1f77bcf86cd799439011"
```

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e8')
ASIN : "B076FJ92M4"
name : "Quinoa Puffs"
weight : "8 ounces"
count : 1
price : 4.49
timestamp : 2024-04-04T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439044"
parent_company_id : "427f1f77bcf86cd799439044"
```

Aggregation: average price of products sold by a seller



5 Documents in the collection

Preview of documents

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e5')
ASIN : "B01LX4VRZM"
name : "Organic Almond Milk"
weight : "64 ounces"
count : 1
price : 3.99
timestamp : 2024-04-01T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439011"
```

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e6')
ASIN : "B07BRRLSVC"
name : "Whole Wheat Pasta"
weight : "16 ounces"
count : 1
price : 1.29
timestamp : 2024-04-02T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439022"
```

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e7')
ASIN : "B083JX2MWB"
name : "Organic Raw Honey"
weight : "12 ounces"
count : 1
price : 7.99
timestamp : 2024-04-03T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439033"
```

```
_id: ObjectId('6616a7ad40dfc19b23f7e8e8')
ASIN : "B076FJ92M4"
name : "Quinoa Puffs"
weight : "8 ounces"
count : 1
price : 4.49
timestamp : 2024-04-04T00:00:00.000+00:00
seller_id : "527f1f77bcf86cd799439044"
```

Stage 1 $group

```
{
  _id: "$seller_id",
  average_price: {
    $avg: "$price",
  },
}
```

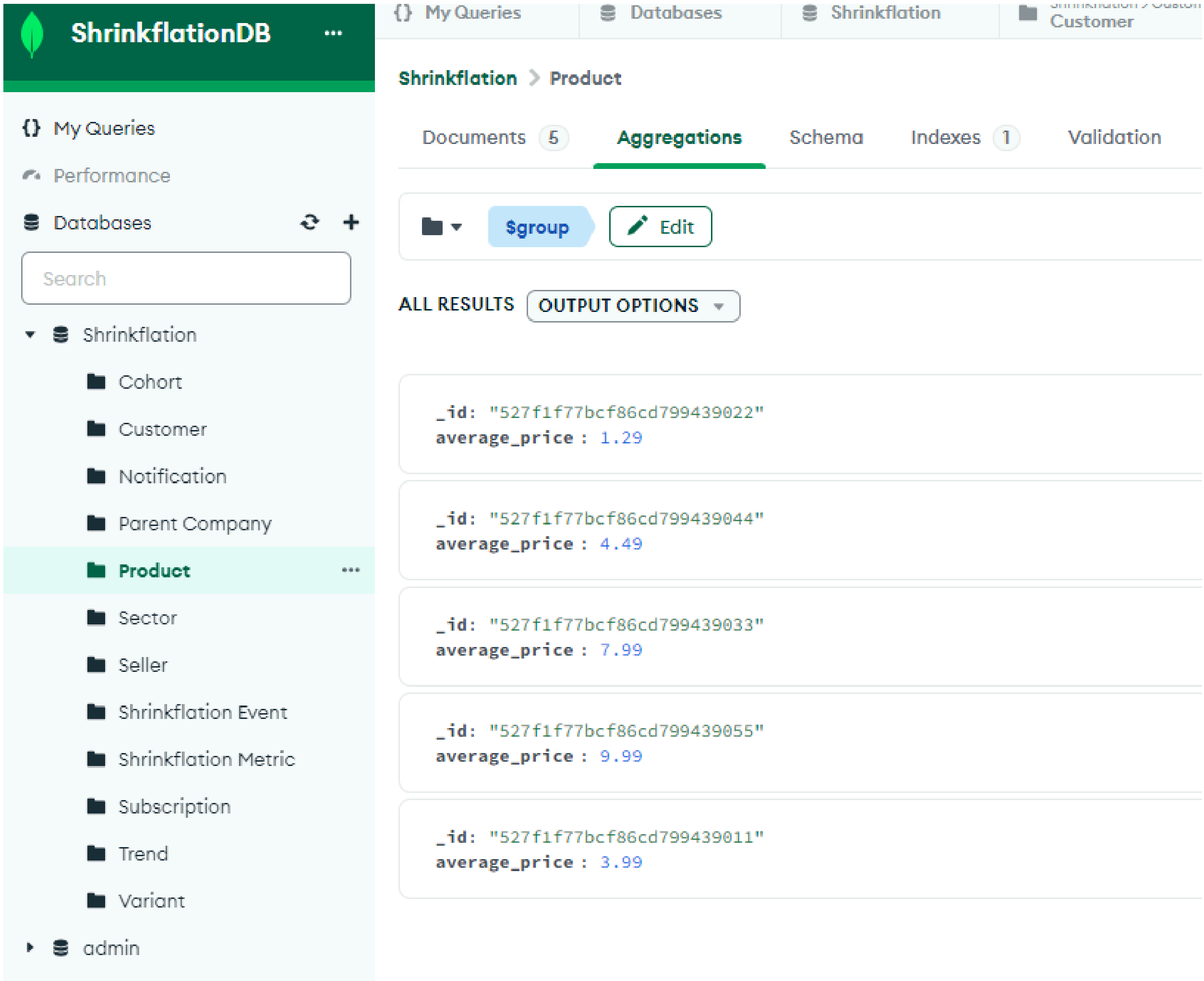
Output after $group stage (Sample of 5 documents)

```
_id: "527f1f77bcf86cd799439011"
average_price : 3.99
```

```
_id: "527f1f77bcf86cd799439055"
average_price : 9.99
```

```
_id: "527f1f77bcf86cd799439044"
average_price : 4.49
```

Aggregation Results:



```
_id: "527f1f77bcf86cd799439022"
average_price : 1.29

_id: "527f1f77bcf86cd799439044"
average_price : 4.49

_id: "527f1f77bcf86cd799439033"
average_price : 7.99

_id: "527f1f77bcf86cd799439055"
average_price : 9.99

_id: "527f1f77bcf86cd799439011"
average_price : 3.99
```

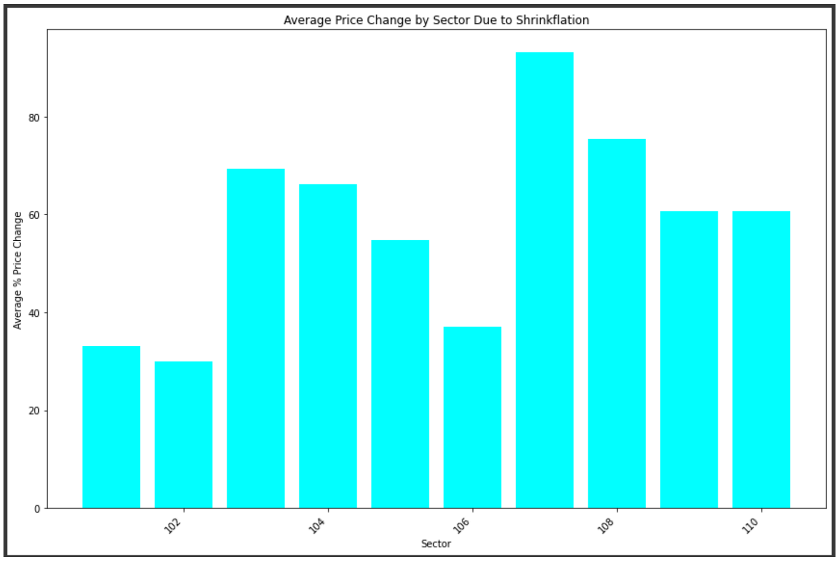
Accessing the Database with a Python Application

1) Establishing a connection to the Shrinkflation database

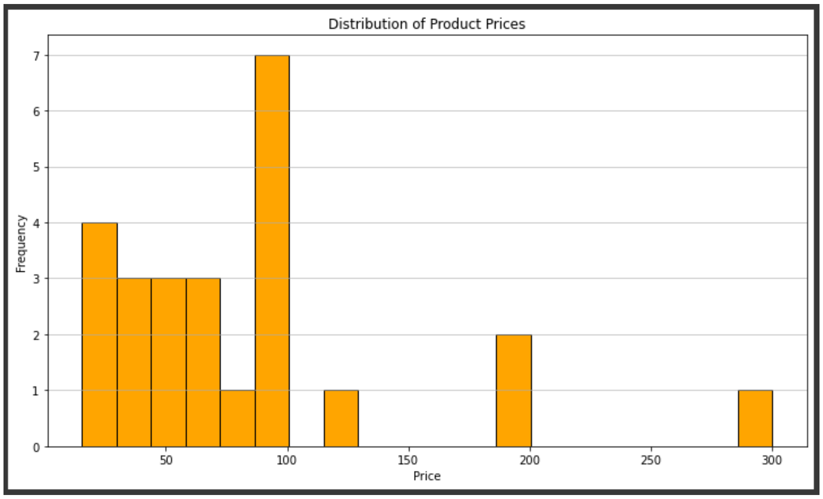
We utilized Python to access our Shrinkflation database and established a connection between the notebook and MySql local server through mysql.connector. This is followed by executing and fetching data from SQL query using cursor.execute. Then we convert the result into a Panda dataframe. Finally, matplotlib is employed to generate graphs for our analytics. See Appendix for detailed codes.

Application Queries and Visualizations

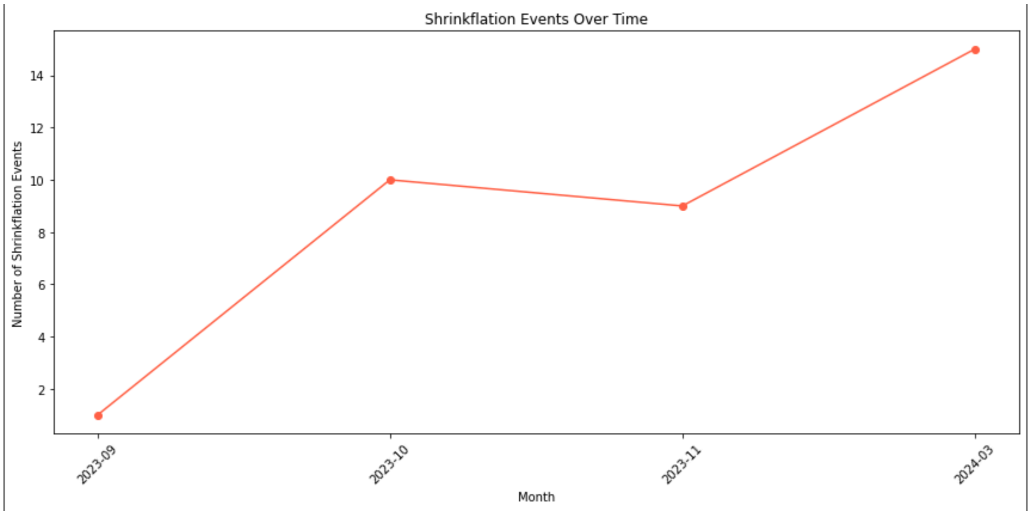
2) Bar Graph: Average price change by sector due to shrinkflation



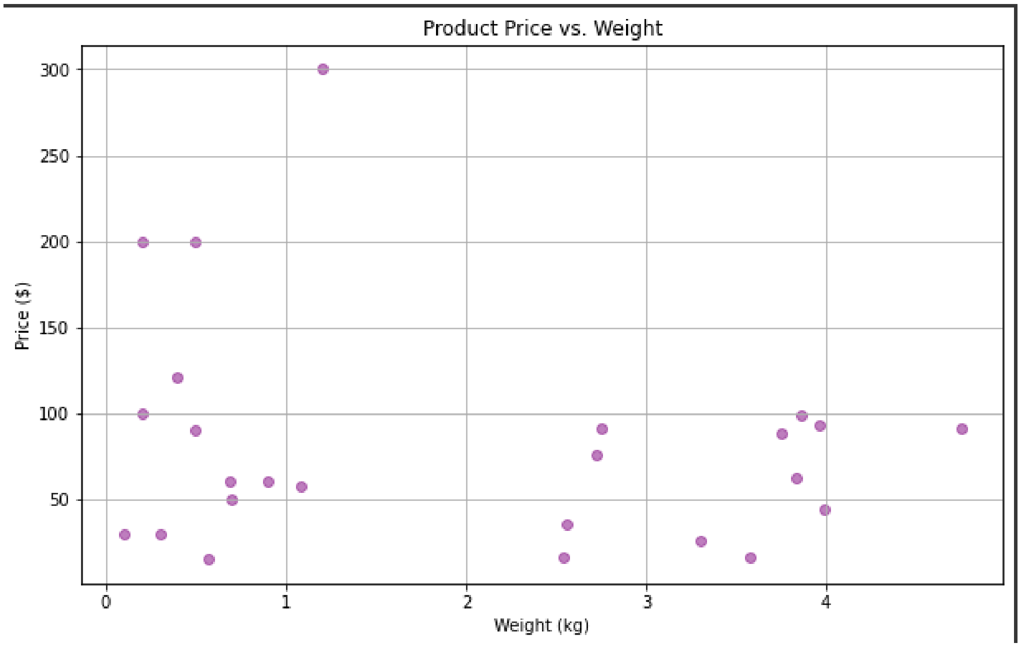
3) Histogram: Distribution of Product Prices



4) Chart: Shrinkflation Events Over Time



5) Scatter Plot: Product Price vs. Weight



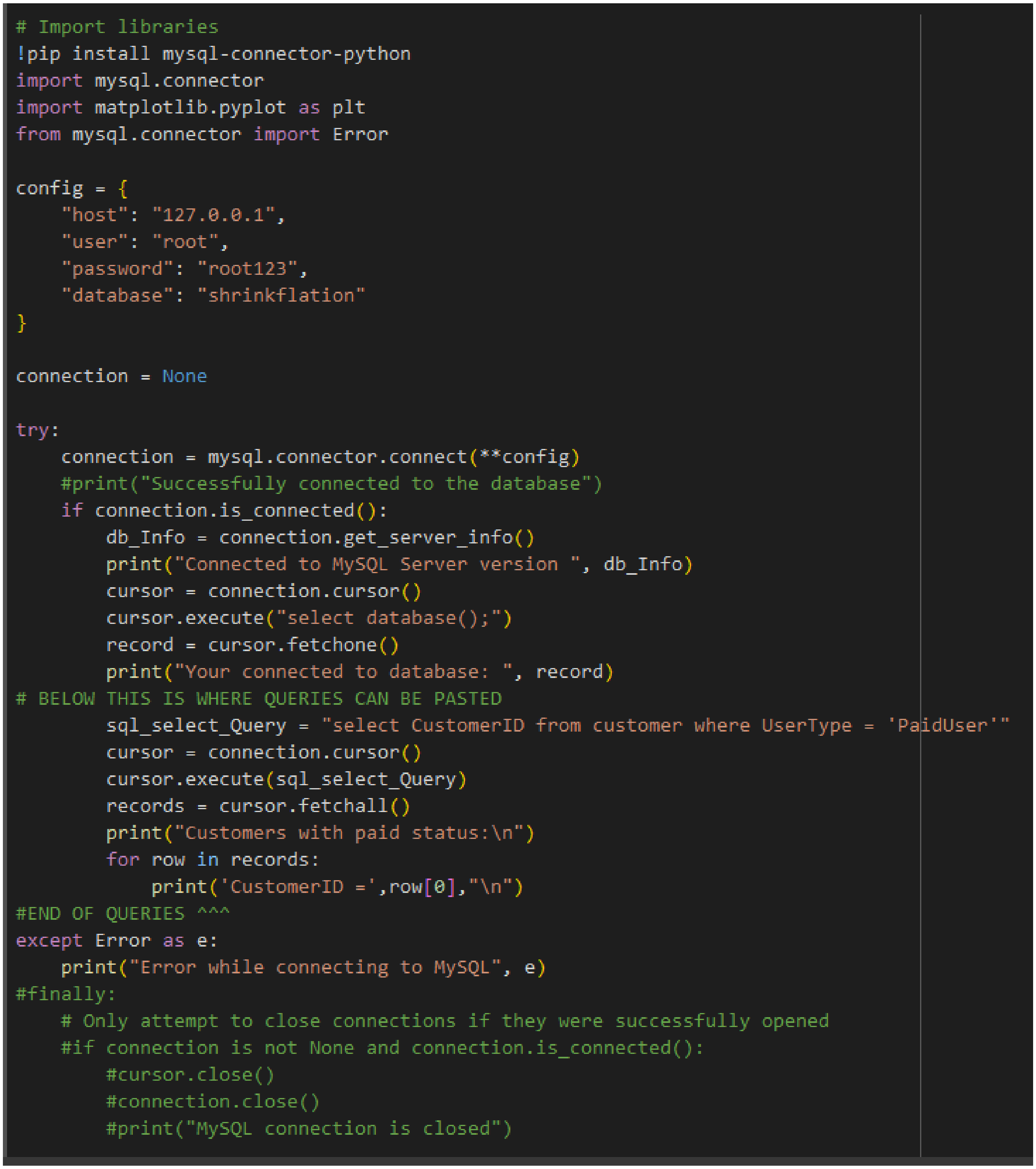
Summary & Conclusion:

Our database architecture is crafted to fulfill two core functions for our conceptual digital platform: it not only empowers consumers to make well-informed purchasing decisions but also provides businesses and third-party resellers with essential metrics. By recording variations in product dimensions and pricing, our platform unveils ongoing trends and behaviors associated with shrinkflation across different sectors and product lines. This repository of information is vital for raising consumer consciousness and assisting companies in strategizing their pricing and sizing strategies to stay competitive while retaining customer loyalty.

To achieve this, our platform is designed to deliver efficient and prompt alerting services, necessitating a backend architecture proficient in managing streaming data and conducting real-time data analysis. This setup is essential for seamless integration of live data into our database, ensuring that dashboards and alerts are continuously updated without any delays.

Further investigation and experimentation are imperative to determine the most effective methods by which our DBMS is capable of archiving historical data. This capability will enable us to monitor and analyze changes over varying timeframes with high efficiency. We also advocate for the adoption of advanced streaming processing technologies such as Google Pub for managing live data streams, and we suggest considering the integration of Memcached DBMS to enhance the speed of data retrieval.

Appendix:

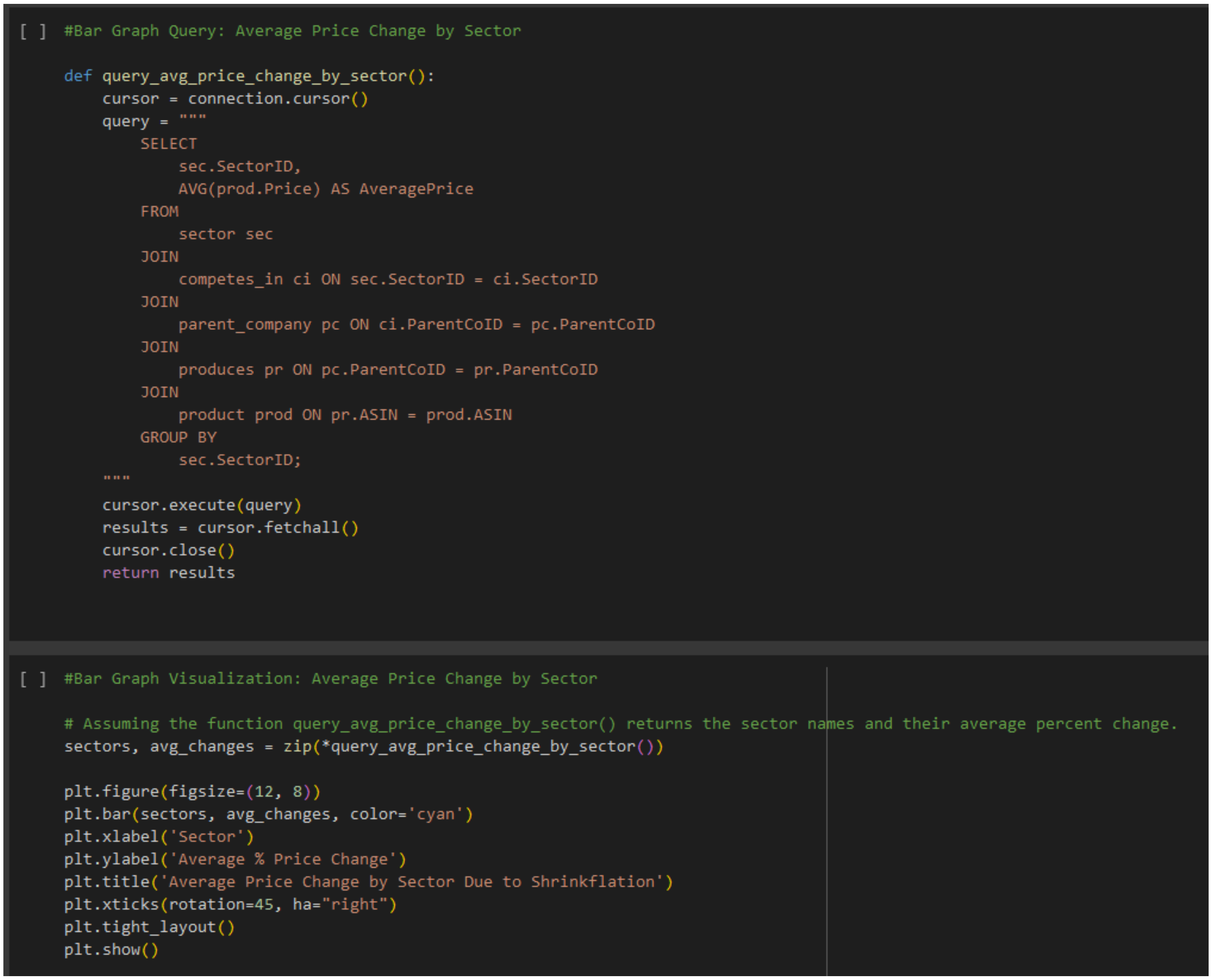


```
# Import libraries
!pip install mysql-connector-python
import mysql.connector
import matplotlib.pyplot as plt
from mysql.connector import Error

config = {
    "host": "127.0.0.1",
    "user": "root",
    "password": "root123",
    "database": "shrinkflation"
}

connection = None

try:
    connection = mysql.connector.connect(**config)
    #print("Successfully connected to the database")
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("Your connected to database: ", record)
# BELOW THIS IS WHERE QUERIES CAN BE PASTED
        sql_select_Query = "select CustomerID from customer where UserType = 'PaidUser'"
        cursor = connection.cursor()
        cursor.execute(sql_select_Query)
        records = cursor.fetchall()
        print("Customers with paid status:\n")
        for row in records:
            print('CustomerID =',row[0],"\n")
#END OF QUERIES ^^^
except Error as e:
    print("Error while connecting to MySQL", e)
#finally:
    # Only attempt to close connections if they were successfully opened
    #if connection is not None and connection.is_connected():
        #cursor.close()
        #connection.close()
        #print("MySQL connection is closed")
```



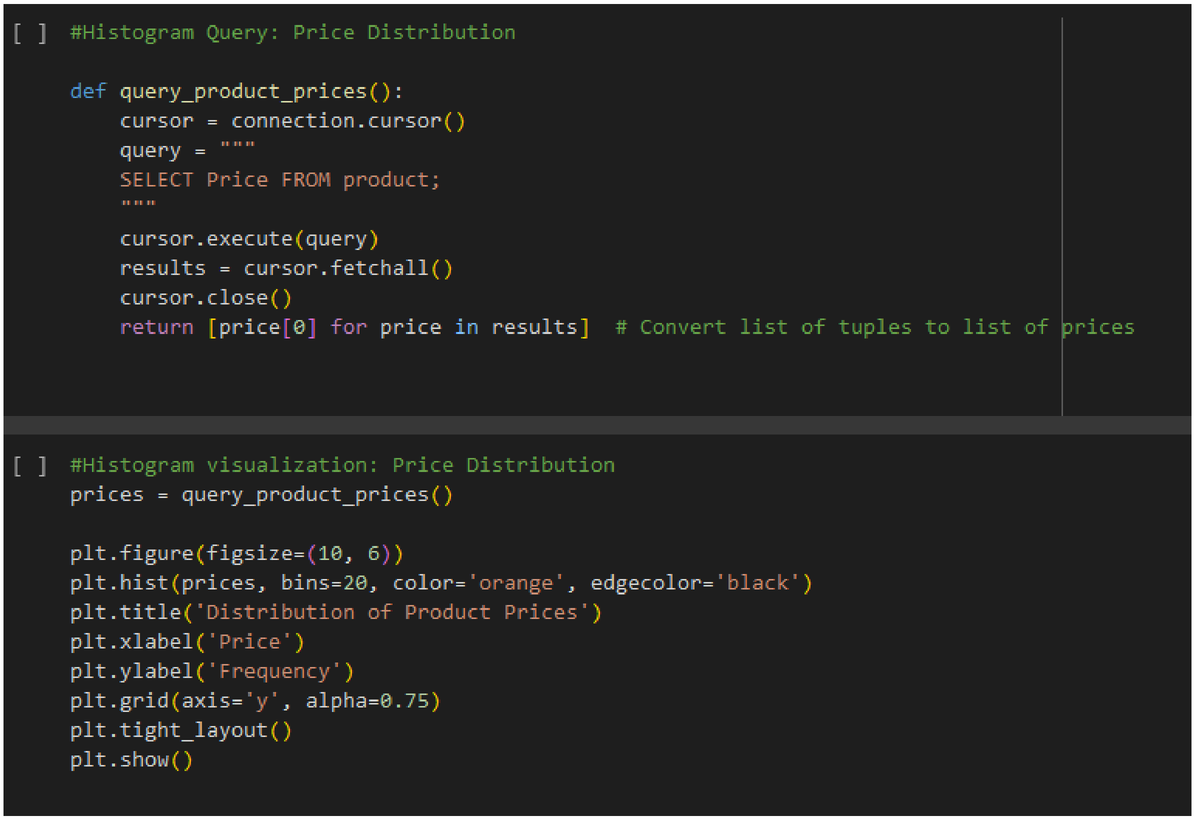
```python
#Bar Graph Query: Average Price Change by Sector

def query_avg_price_change_by_sector():
    cursor = connection.cursor()
    query = """
        SELECT
            sec.SectorID,
            AVG(prod.Price) AS AveragePrice
        FROM
            sector sec
        JOIN
            competes_in ci ON sec.SectorID = ci.SectorID
        JOIN
            parent_company pc ON ci.ParentCoID = pc.ParentCoID
        JOIN
            produces pr ON pc.ParentCoID = pr.ParentCoID
        JOIN
            product prod ON pr.ASIN = prod.ASIN
        GROUP BY
            sec.SectorID;
    """
    cursor.execute(query)
    results = cursor.fetchall()
    cursor.close()
    return results
```

```python
#Bar Graph Visualization: Average Price Change by Sector

# Assuming the function query_avg_price_change_by_sector() returns the sector names and their average percent change.
sectors, avg_changes = zip(*query_avg_price_change_by_sector())

plt.figure(figsize=(12, 8))
plt.bar(sectors, avg_changes, color='cyan')
plt.xlabel('Sector')
plt.ylabel('Average % Price Change')
plt.title('Average Price Change by Sector Due to Shrinkflation')
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
```

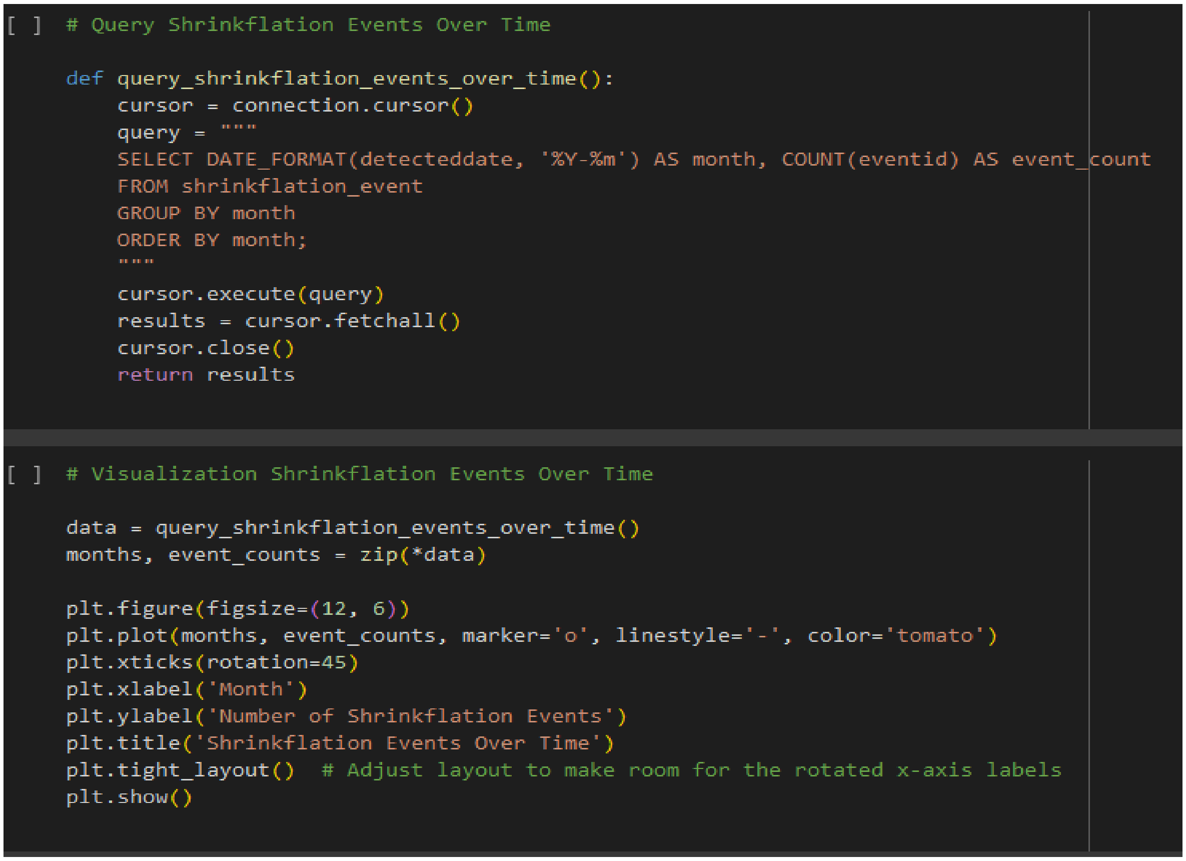


```python
#Histogram Query: Price Distribution

def query_product_prices():
    cursor = connection.cursor()
    query = """
    SELECT Price FROM product;
    """
    cursor.execute(query)
    results = cursor.fetchall()
    cursor.close()
    return [price[0] for price in results]  # Convert list of tuples to list of prices
```

```python
#Histogram visualization: Price Distribution
prices = query_product_prices()

plt.figure(figsize=(10, 6))
plt.hist(prices, bins=20, color='orange', edgecolor='black')
plt.title('Distribution of Product Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.grid(axis='y', alpha=0.75)
plt.tight_layout()
plt.show()
```



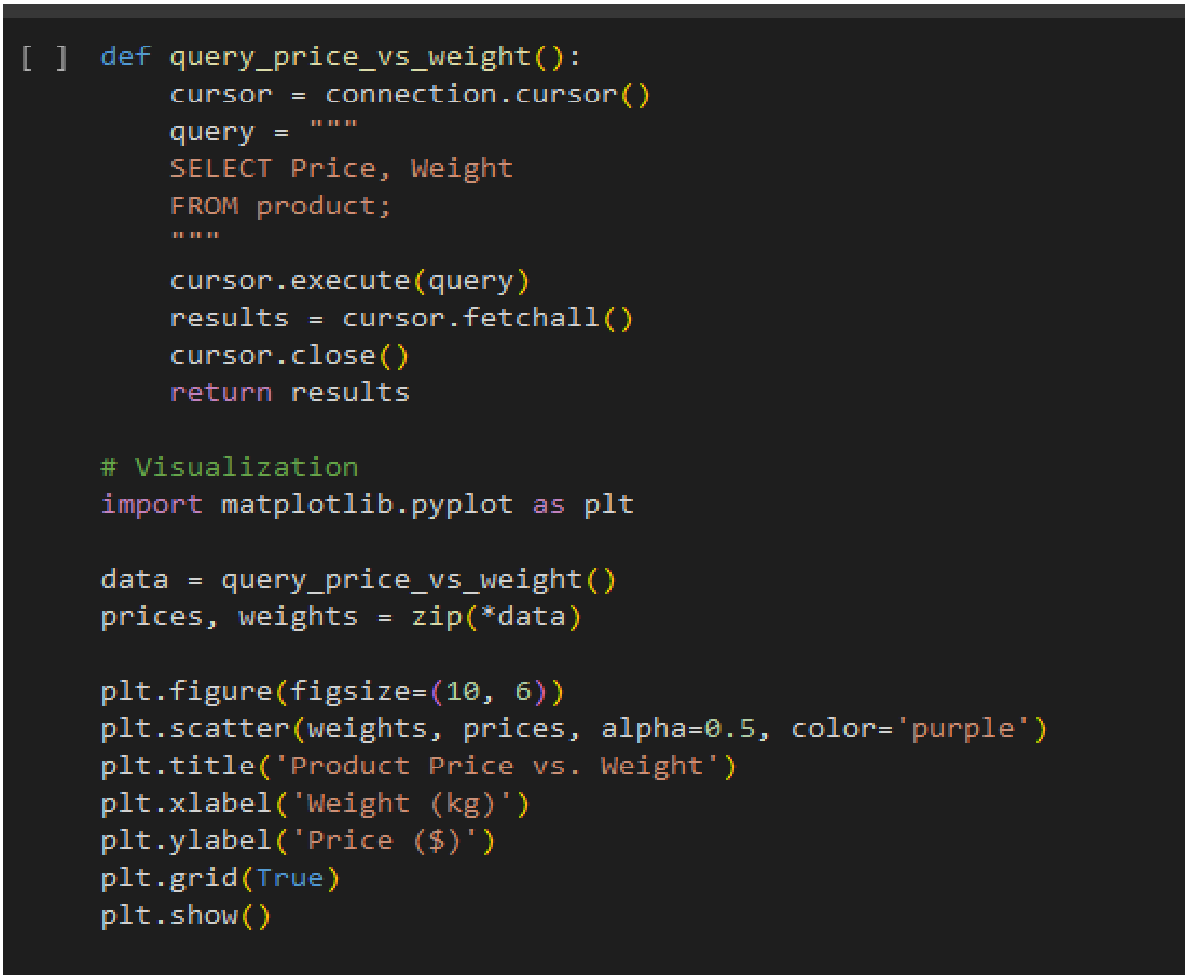
```python
# Query Shrinkflation Events Over Time

def query_shrinkflation_events_over_time():
    cursor = connection.cursor()
    query = """
    SELECT DATE_FORMAT(detecteddate, '%Y-%m') AS month, COUNT(eventid) AS event_count
    FROM shrinkflation_event
    GROUP BY month
    ORDER BY month;
    """
    cursor.execute(query)
    results = cursor.fetchall()
    cursor.close()
    return results
```

```python
# Visualization Shrinkflation Events Over Time

data = query_shrinkflation_events_over_time()
months, event_counts = zip(*data)

plt.figure(figsize=(12, 6))
plt.plot(months, event_counts, marker='o', linestyle='-', color='tomato')
plt.xticks(rotation=45)
plt.xlabel('Month')
plt.ylabel('Number of Shrinkflation Events')
plt.title('Shrinkflation Events Over Time')
plt.tight_layout()  # Adjust layout to make room for the rotated x-axis labels
plt.show()
```



```
def query_price_vs_weight():
    cursor = connection.cursor()
    query = """
    SELECT Price, Weight
    FROM product;
    """
    cursor.execute(query)
    results = cursor.fetchall()
    cursor.close()
    return results

# Visualization
import matplotlib.pyplot as plt

data = query_price_vs_weight()
prices, weights = zip(*data)

plt.figure(figsize=(10, 6))
plt.scatter(weights, prices, alpha=0.5, color='purple')
plt.title('Product Price vs. Weight')
plt.xlabel('Weight (kg)')
plt.ylabel('Price ($)')
plt.grid(True)
plt.show()
```