E-commerce warehouse database

Group 7

Team members: Wan Luo, Zhiming Hu, and Weihe Pan

Executive Summary

This project presents the design and implementation of an integrated warehouse database system for an e-commerce platform, addressing the fundamental challenges of inventory management, supply-chain monitoring, and sales analysis. Modern e-commerce operations involve large-scale product catalogs, distributed warehouses, multiple suppliers, and millions of user interactions. Without a unified and well-structured data management system, issues such as overstocking, stockouts, inefficient allocation, and fragmented logistics information can severely disrupt operational efficiency.

To address these challenges, we developed a comprehensive data architecture combining relational database management using MySQL and scalable NoSQL storage using MongoDB. The MySQL database captures core transactional and warehouse data through a fully normalized relational schema, ensuring consistency, integrity, and efficient query processing for inventory, orders, logistics, and supplier relationships. In parallel, MongoDB supports flexible document-oriented storage for product information, supplier records, user profiles, and AI-generated inventory analyses, enabling rapid retrieval and horizontal scalability.

Additionally, an application layer built with Flask, Python, and a JavaScript-based web interface integrates advanced analytics powered by the Gemini-2.5-Flash large language model. The system periodically identifies the top unsold products, generates automated root-cause analysis, and stores these insights in MongoDB. Building upon this foundation, we implemented a machine learning demand forecasting system using Random Forest regression trained on over 2000 historical order records. The ML model achieves an R^2 score of 0.72, enabling accurate prediction of order quantities at different price points. This capability supports data-driven pricing optimization for slow-moving inventory, where the system analyzes multiple discount scenarios (10%, 20%, 30%, and 40% reductions) and predicts their impact on sales volume and revenue. By combining ML-powered demand forecasting with AI-generated strategic recommendations, warehouse managers can make informed decisions on optimal pricing strategies to maximize revenue while efficiently clearing excess inventory. Visualizations produced via Python and Matplotlib provide real-time data interpretation for end users. Overall, this project demonstrates a robust, extensible, and AI-enhanced warehouse data system capable of supporting data-driven decision-making in modern e-commerce environments.

I. Introduction

Currently, the e-commerce platforms manage a lot of goods and have a complex warehouse network. Without united warehouse data management, problems such as overstocking and stockouts can easily occur. Moreover, the supply chain consists of multiple links, so real-time monitoring and early warnings are necessary to ensure the transparency and traceability of the pre-sale and post-sale processes.

For the platform, it is also very important to cross-check the warehouse data with the sales data to ensure the accuracy of operations and risk control. Therefore, building a complete warehouse database is a key part of the e-commerce platform. It helps merchants manage operations, maintain a stable supply chain, improve logistics efficiency, and reduce business risks.

Theory for E-commerce platform warehouse database:

When a merchant registers on the e-commerce platform and uploads their products, the system records the warehouse details of their goods. It tracks the number of each product and updates it in real time, so that merchants can clearly see the warehouse situation. This helps them make better decisions for supply and sales planning. Additionally, we have incorporated AI inventory analysis reports that provide merchants with tailored promotional recommendations based on their inventory status, as well as machine learning-powered pricing optimization that predicts demand at different discount levels to maximize revenue while clearing slow-moving stock.

The database also tracks the location where the products are stored, whether in the platform's warehouse or the merchant's own facilities. By recording the storage address and the entry and exit of the goods, the system supports faster order fulfillment. It also has a supply chain early-warning function. If there is a delivery delay or stock shortage, the system will send alerts to the merchant and the platform, so that they can take action promptly. Additionally, it will remind of situations of insufficient or excessive warehouse, helping to prevent stock shortages or stock accumulation. This makes the operation more cost-effective.

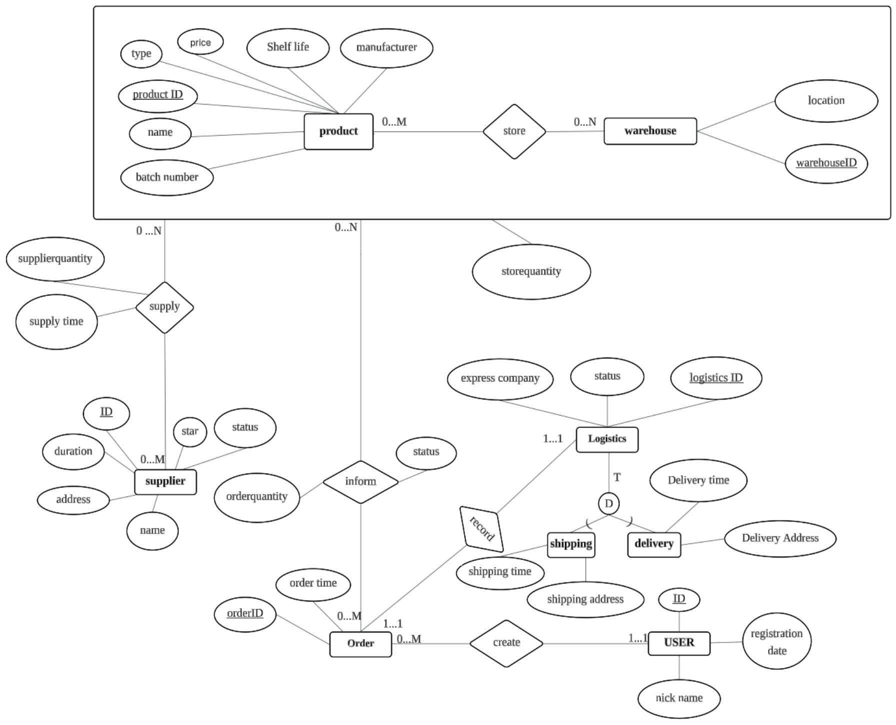
The database can also save detailed records of product returns and exchanges. For each update, such a return, the system records who initiated it, when it occurred, and the reason behind the transactions. This clarifies responsibilities and supports customer service. Finally, by comparing the warehouse flow data with the sales data, the platform can cross-check the information. This increases accuracy, ensures consistency of warehouse and orders, and provides better data integrity.

Other requirements:
1) Suppliers have their unique ID; warehouses have a unique warehouseID; different products have their unique productID; orders have a unique orderID; logistics have a unique logisticsID; and users have their unique ID.
2) An order includes one or more products that are stored in different warehouses, represented by the ‘store’ relationship
3) A supplier can supply zero to an infinite number of products to warehouses; a warehouse can be supplied by zero to an infinite number of merchants.
4) A user can create zero to an infinite number of orders; an order must be created by

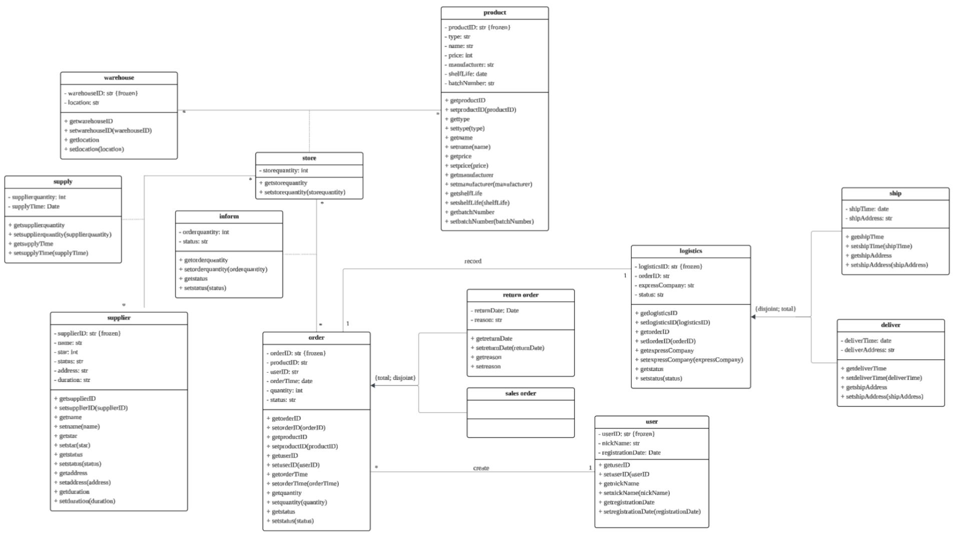
one and only one user.
5) The logistics record must belong to one and only one order; an order can have only one logistics record.
6) A warehouse can store zero to an infinite number of products; a product can be stored by zero to an infinite number of warehouses.
7) The logistics record has two subtypes through ISA relationship: shipping and delivery, where shipping and delivery share the same logistics record. Each logistics must be either delivery or shipping.

II. Conceptual Data Modeling

1. EER:



2. UML:



III. Mapping Conceptual Model to Relational Model

1. Relational Model

User table:

<u>UserID</u>	nick name	register time

UserID **PRIMARY KEY**

order table:

<u>OrderID</u>	*UserID*	order time

OrderID **PRIMARY KEY**
UserID *FOREIGN KEY* refers to UserID in User table, *NOT NULL*

Warehouse table:

<u>WarehouseID</u>	location

WarehouseID **PRIMARY KEY**

Product table:

<u>ProductID</u>	product name	type	price	manufacturer	shelf life	batch number

ProductID_**PRIMARY KEY**

Store record table:

<u>*WarehouseID*</u>	<u>*ProductID*</u>	storequantity

WarehouseID *FOREIGN KEY* refers to WarehouseID in Warehouse table, *NOT NULL*
ProductID_*FOREIGN KEY* refers to ProductID in Product table, *NOT NULL*
PRIMARY KEY (ProductID,_WarehouseID_)

inform table

<u>*OrderID*</u>	<u>*ProductID*</u>	<u>*WarehouseID*</u>	orderquanti ty	statu s

OrderID_*FOREIGN KEY* refers to OrderID in order table, *NOT NULL*
ProductID_*FOREIGN KEY* refers to ProductID in Store record table, *NOT NULL*
WarehouseID_*FOREIGN KEY* refers to WarehouseID_in Store record table, *NOT NULL*
PRIMARY KEY (OrderID, ProductID,_WarehouseID_)

supplier table:

<u>supplierID</u>	supplier name	address	star	duration	status

supplierID **PRIMARY KEY**

Good supply table:

<u>supplierID</u>	*<u>WarehouseID</u>*	*<u>ProductID</u>*	quantity	supply time

supplierID *FOREIGN KEY* refers to supplierID in supplier table, *NOT NULL*
ProductID *FOREIGN KEY* refers to ProductID in Store record table, *NOT NULL*
WarehouseID *FOREIGN KEY* refers to WarehouseID in Store record table, *NOT NULL*
PRIMARY KEY (supplierID, ProductID, WarehouseID)

Logistics table:

<u>LogisticsID</u>	*orderID*	express company	logistics status

LogisticsID **PRIMARY KEY**
OrderID *FOREIGN KEY* refers to OrderID in order table, *NOT NULL*

Shipping product table:

<u>LogisticsID</u>	shipping time	shipping address

LogisticsID **PRIMARY KEY**
LogisticsID *FOREIGN KEY* refers to LogisticsID in Logistics table, *NOT NULL*

Delivery product table:

<u>LogisticsID</u>	delivery time	delivery address

LogisticsID **PRIMARY KEY**
LogisticsID *FOREIGN KEY* refers to LogisticsID in Logistics table, *NOT NULL*

IV. Implementation of the Relation Model via MySQL and NoSQL

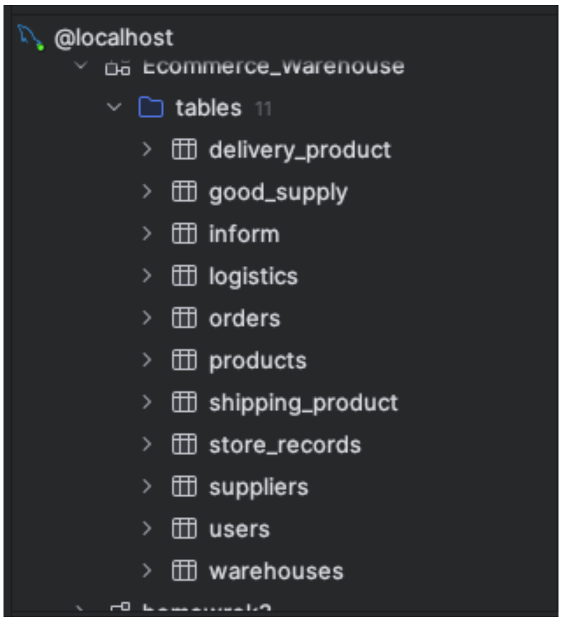
(1) MySQL

Create a database:

```
CREATE DATABASE IF NOT EXISTS Ecommerce_Warehouse;
USE Ecommerce_Warehouse;
```

Create tables by SQL

```
-- User table
DROP TABLE IF EXISTS users;
CREATE TABLE users (
  user_id VARCHAR(32) PRIMARY KEY,
  nickname VARCHAR(64),
  register_time DATETIME
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- order table
DROP TABLE IF EXISTS orders;
CREATE TABLE orders (
  order_id VARCHAR(64) PRIMARY KEY,
  user_id VARCHAR(32),
  order_time DATETIME,
  FOREIGN KEY (user_id) REFERENCES users(user_id) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- warehouse table
DROP TABLE IF EXISTS warehouses;
CREATE TABLE warehouses (
  warehouse_id VARCHAR(32) PRIMARY KEY,
  location VARCHAR(128)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- product table
DROP TABLE IF EXISTS products;
CREATE TABLE products (
  product_id VARCHAR(32) PRIMARY KEY,
  product_name VARCHAR(128),
  type VARCHAR(64),
  price DECIMAL(10,2),
  manufacturer VARCHAR(128),
  shelf_life INT,
  batch_number VARCHAR(64)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- store table
DROP TABLE IF EXISTS store_records;
CREATE TABLE store_records (
  warehouse_id VARCHAR(32) NOT NULL,
  product_id VARCHAR(32) NOT NULL,
   storequantity INT,
   PRIMARY KEY (product_id, warehouse_id),
   FOREIGN KEY (warehouse_id) REFERENCES warehouses(warehouse_id) ON UPDATE
CASCADE,
   FOREIGN KEY (product_id) REFERENCES products(product_id) ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- information table
DROP TABLE IF EXISTS inform;
CREATE TABLE inform (
   order_id VARCHAR(64) NOT NULL,
   product_id VARCHAR(32) NOT NULL,
   warehouse_id VARCHAR(32) NOT NULL,
   orderquantity INT,
   status VARCHAR(32),
   PRIMARY KEY (order_id, product_id, warehouse_id),
   FOREIGN KEY (order_id) REFERENCES orders(order_id) ON UPDATE CASCADE,
   FOREIGN KEY (product_id, warehouse_id) REFERENCES store_records(product_id,
warehouse_id) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- supplier table
DROP TABLE IF EXISTS suppliers;
CREATE TABLE suppliers (
   supplier_id VARCHAR(32) PRIMARY KEY,
   supplier_name VARCHAR(128),
   address VARCHAR(256),
   star INT,
   duration INT,
   status VARCHAR(32)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- good supply table
DROP TABLE IF EXISTS good_supply;
CREATE TABLE good_supply (
   supplier_id VARCHAR(32) NOT NULL,
   product_id VARCHAR(32) NOT NULL,
   warehouse_id VARCHAR(32) NOT NULL,
   quantity INT,
   supply_time DATETIME,
   PRIMARY KEY (supplier_id, product_id, warehouse_id),
   FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id) ON UPDATE
CASCADE,
   FOREIGN KEY (product_id, warehouse_id) REFERENCES store_records(product_id,
warehouse_id) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- logistic table
DROP TABLE IF EXISTS logistics;
CREATE TABLE logistics (
   logistics_id VARCHAR(32) PRIMARY KEY,
   order_id VARCHAR(64) NOT NULL,
   express_company VARCHAR(64),
   logistics_status VARCHAR(32),
    FOREIGN KEY (order_id) REFERENCES orders(order_id) ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- shipping table
DROP TABLE IF EXISTS shipping_product;
CREATE TABLE shipping_product (
    logistics_id VARCHAR(32) PRIMARY KEY,
    shipping_time DATETIME,
    shipping_address VARCHAR(256),
    FOREIGN KEY (logistics_id) REFERENCES logistics(logistics_id) ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- delivery table
DROP TABLE IF EXISTS delivery_product;
CREATE TABLE delivery_product (
    logistics_id VARCHAR(32) PRIMARY KEY,
    delivery_time DATETIME,
    delivery_address VARCHAR(256),
    FOREIGN KEY (logistics_id) REFERENCES logistics(logistics_id) ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```



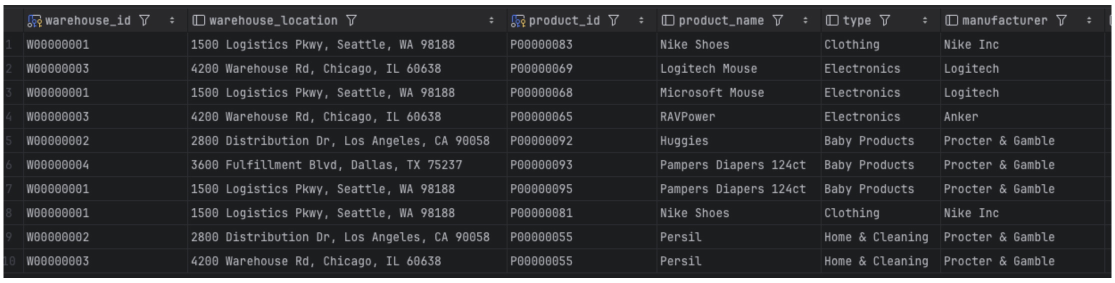
```
@localhost
  Ecommerce_Warehouse
    tables 11
      delivery_product
      good_supply
      inform
      logistics
      orders
      products
      shipping_product
      store_records
      suppliers
      users
      warehouses
```

SQL queries on the database

1. The timer will periodically retrieve items from the warehouse that require alerts.

Plan:
Get all products that quantity within 20

```
SELECT

sr.warehouse_id,

w.location AS warehouse_location,

sr.product_id,

p.product_name,

p.type,

p.manufacturer,

sr.storequantity

FROM store_records sr

JOIN products p ON sr.product_id = p.product_id

JOIN warehouses w ON sr.warehouse_id = w.warehouse_id

WHERE sr.storequantity < 100

ORDER BY sr.storequantity ASC;
```

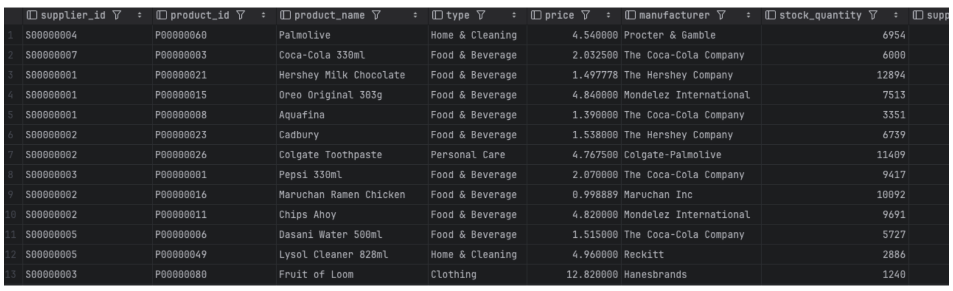


	warehouse_id	warehouse_location	product_id	product_name	type	manufacturer
1	W00000001	1500 Logistics Pkwy, Seattle, WA 98188	P00000083	Nike Shoes	Clothing	Nike Inc
2	W00000003	4200 Warehouse Rd, Chicago, IL 60638	P00000069	Logitech Mouse	Electronics	Logitech
3	W00000001	1500 Logistics Pkwy, Seattle, WA 98188	P00000068	Microsoft Mouse	Electronics	Logitech
4	W00000003	4200 Warehouse Rd, Chicago, IL 60638	P00000065	RAVPower	Electronics	Anker
5	W00000002	2800 Distribution Dr, Los Angeles, CA 90058	P00000092	Huggies	Baby Products	Procter & Gamble
6	W00000004	3600 Fulfillment Blvd, Dallas, TX 75237	P00000093	Pampers Diapers 124ct	Baby Products	Procter & Gamble
7	W00000001	1500 Logistics Pkwy, Seattle, WA 98188	P00000095	Pampers Diapers 124ct	Baby Products	Procter & Gamble
8	W00000001	1500 Logistics Pkwy, Seattle, WA 98188	P00000081	Nike Shoes	Clothing	Nike Inc
9	W00000002	2800 Distribution Dr, Los Angeles, CA 90058	P00000055	Persil	Home & Cleaning	Procter & Gamble
10	W00000003	4200 Warehouse Rd, Chicago, IL 60638	P00000055	Persil	Home & Cleaning	Procter & Gamble

2. The warehouse management system needs to identify slow-moving products and generate AI-powered promotional strategies, analyzing product performance across multiple warehouses and suppliers.

Plan:
Query the top 50 products' information with low sell-through rates (<40%) from the database

```
SELECT
    MIN(gs.supplier_id) as supplier_id,
    MIN(p.product_id) as product_id,
    p.product_name,
    MIN(p.type) as type,
    AVG(p.price) as price,
    MIN(p.manufacturer) as manufacturer,
    SUM(sr.storequantity) as stock_quantity,
    SUM(gs.quantity) as supply_quantity,
    MIN(sr.warehouse_id) as warehouse_id,
    MIN(gs.supply_time) as supply_time,
    MAX(DATEDIFF(CURDATE(), DATE(gs.supply_time))) as days_in_stock,
    AVG(ROUND((gs.quantity - sr.storequantity) / gs.quantity, 2)) as sell_through_rate
 FROM good_supply gs
 LEFT JOIN store_records sr
       ON gs.warehouse_id = sr.warehouse_id
       AND gs.product_id = sr.product_id
 LEFT JOIN products p
        ON gs.product_id = p.product_id
 WHERE DATEDIFF(CURDATE(), DATE(gs.supply_time)) >= 10
         AND (gs.quantity - sr.storequantity) / gs.quantity < 0.4

 GROUP BY p.product_name
 ORDER BY sell_through_rate ASC
 LIMIT50
```



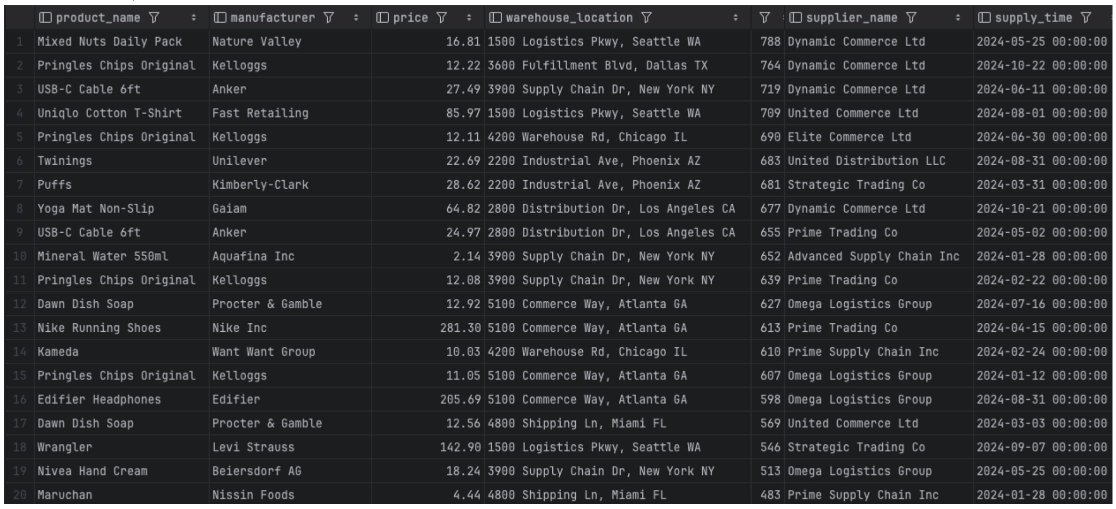
	supplier_id	product_id	product_name	type	price	manufacturer	stock_quantity	supp
1	S00000004	P00000060	Palmolive	Home & Cleaning	4.540000	Procter & Gamble	6954	
2	S00000007	P00000003	Coca-Cola 330ml	Food & Beverage	2.032500	The Coca-Cola Company	6000	
3	S00000001	P00000021	Hershey Milk Chocolate	Food & Beverage	1.497778	The Hershey Company	12894	
4	S00000001	P00000015	Oreo Original 303g	Food & Beverage	4.840000	Mondelez International	7513	
5	S00000001	P00000008	Aquafina	Food & Beverage	1.390000	The Coca-Cola Company	3351	
6	S00000002	P00000023	Cadbury	Food & Beverage	1.538000	The Hershey Company	6739	
7	S00000002	P00000026	Colgate Toothpaste	Personal Care	4.767500	Colgate-Palmolive	11409	
8	S00000003	P00000001	Pepsi 330ml	Food & Beverage	2.070000	The Coca-Cola Company	9417	
9	S00000002	P00000016	Maruchan Ramen Chicken	Food & Beverage	0.998889	Maruchan Inc	10092	
10	S00000002	P00000011	Chips Ahoy	Food & Beverage	4.820000	Mondelez International	9691	
11	S00000005	P00000006	Dasani Water 500ml	Food & Beverage	1.515000	The Coca-Cola Company	5727	
12	S00000005	P00000049	Lysol Cleaner 828ml	Home & Cleaning	4.960000	Reckitt	2886	
13	S00000003	P00000080	Fruit of Loom	Clothing	12.820000	Hanesbrands	1240	

3. Inventory managers need to view which products have sufficient stock (>100 units) while understanding supplier information and recent restocking times for these products to make inventory allocation decisions.

Plan:
Find products with their warehouse stock levels and supplier information

```
SELECT
   p.product_name,
   p.manufacturer,
   p.price,
   w.location AS warehouse_location,
   sr.storequantity,
   s.supplier_name,
   gs.supply_time
FROM products p
INNER JOIN store_records sr ON p.product_id = sr.product_id
INNER JOIN warehouses w ON sr.warehouse_id = w.warehouse_id
INNER JOIN good_supply gs ON p.product_id = gs.product_id AND
sr.warehouse_id = gs.warehouse_id
INNER JOIN suppliers s ON gs.supplier_id = s.supplier_id
WHERE sr.storequantity > 100
```

```
ORDER BY sr.storequantity DESC
LIMIT 20;
```

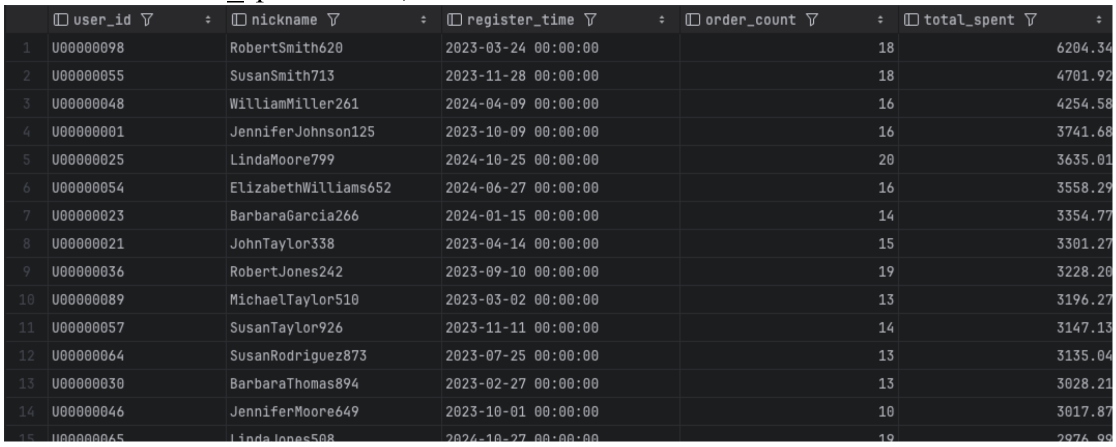


	product_name	manufacturer	price	warehouse_location		supplier_name	supply_time
1	Mixed Nuts Daily Pack	Nature Valley	16.81	1500 Logistics Pkwy, Seattle WA	788	Dynamic Commerce Ltd	2024-05-25 00:00:00
2	Pringles Chips Original	Kelloggs	12.22	3600 Fulfillment Blvd, Dallas TX	764	Dynamic Commerce Ltd	2024-10-22 00:00:00
3	USB-C Cable 6ft	Anker	27.49	3900 Supply Chain Dr, New York NY	719	Dynamic Commerce Ltd	2024-06-11 00:00:00
4	Uniqlo Cotton T-Shirt	Fast Retailing	85.97	1500 Logistics Pkwy, Seattle WA	709	United Commerce Ltd	2024-08-01 00:00:00
5	Pringles Chips Original	Kelloggs	12.11	4200 Warehouse Rd, Chicago IL	690	Elite Commerce Ltd	2024-06-30 00:00:00
6	Twinings	Unilever	22.69	2200 Industrial Ave, Phoenix AZ	683	United Distribution LLC	2024-08-31 00:00:00
7	Puffs	Kimberly-Clark	28.62	2200 Industrial Ave, Phoenix AZ	681	Strategic Trading Co	2024-03-31 00:00:00
8	Yoga Mat Non-Slip	Gaiam	64.82	2800 Distribution Dr, Los Angeles CA	677	Dynamic Commerce Ltd	2024-10-21 00:00:00
9	USB-C Cable 6ft	Anker	24.97	2800 Distribution Dr, Los Angeles CA	655	Prime Trading Co	2024-05-02 00:00:00
10	Mineral Water 550ml	Aquafina Inc	2.14	3900 Supply Chain Dr, New York NY	652	Advanced Supply Chain Inc	2024-01-28 00:00:00
11	Pringles Chips Original	Kelloggs	12.08	3900 Supply Chain Dr, New York NY	639	Prime Trading Co	2024-02-22 00:00:00
12	Dawn Dish Soap	Procter & Gamble	12.92	5100 Commerce Way, Atlanta GA	627	Omega Logistics Group	2024-07-16 00:00:00
13	Nike Running Shoes	Nike Inc	281.30	5100 Commerce Way, Atlanta GA	613	Prime Trading Co	2024-04-15 00:00:00
14	Kameda	Want Want Group	10.03	4200 Warehouse Rd, Chicago IL	610	Prime Supply Chain Inc	2024-02-24 00:00:00
15	Pringles Chips Original	Kelloggs	11.05	5100 Commerce Way, Atlanta GA	607	Omega Logistics Group	2024-01-12 00:00:00
16	Edifier Headphones	Edifier	205.69	5100 Commerce Way, Atlanta GA	598	Omega Logistics Group	2024-08-31 00:00:00
17	Dawn Dish Soap	Procter & Gamble	12.56	4800 Shipping Ln, Miami FL	569	United Commerce Ltd	2024-03-03 00:00:00
18	Wrangler	Levi Strauss	142.90	1500 Logistics Pkwy, Seattle WA	546	Strategic Trading Co	2024-09-07 00:00:00
19	Nivea Hand Cream	Beiersdorf AG	18.24	3900 Supply Chain Dr, New York NY	513	Omega Logistics Group	2024-05-25 00:00:00
20	Maruchan	Nissin Foods	4.44	4800 Shipping Ln, Miami FL	483	Prime Supply Chain Inc	2024-01-28 00:00:00

4. The marketing department needs to analyze user activity, specifically to identify "dormant users" who registered but never placed orders for marketing activation campaigns, while also viewing active users' spending behavior.

Plan:
List all users and their order count (including users with no orders)

```
SELECT
    u.user_id,
    u.nickname,
    u.register_time,
    COUNT(o.order_id) AS order_count,
    COALESCE(SUM(i.orderquantity * p.price), 0) AS total_spent
FROM users u
LEFT JOIN orders o ON u.user_id = o.user_id
LEFT JOIN inform i ON o.order_id = i.order_id
LEFT JOIN products p ON i.product_id = p.product_id
GROUP BY u.user_id, u.nickname, u.register_time
ORDER BY total_spent DESC;
```

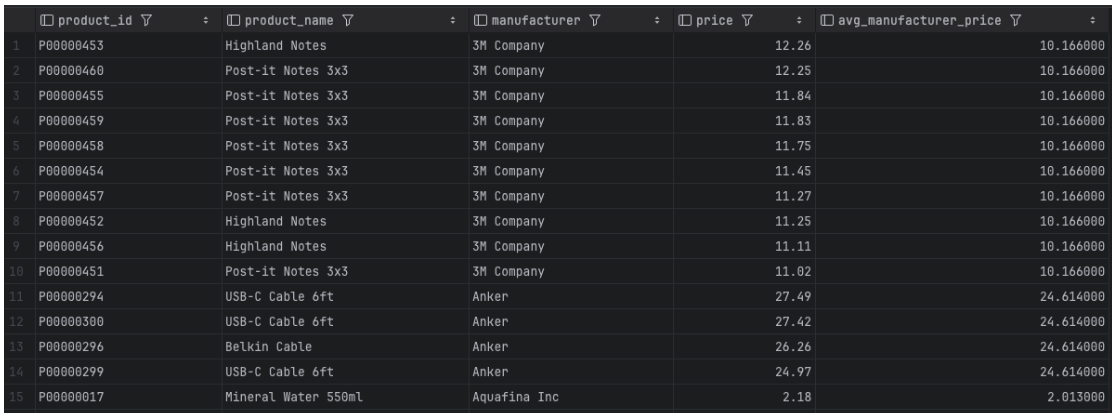


	user_id	nickname	register_time	order_count	total_spent
1	U00000098	RobertSmith620	2023-03-24 00:00:00	18	6204.34
2	U00000055	SusanSmith713	2023-11-28 00:00:00	18	4701.92
3	U00000048	WilliamMiller261	2024-04-09 00:00:00	16	4254.58
4	U00000001	JenniferJohnson125	2023-10-09 00:00:00	16	3741.68
5	U00000025	LindaMoore799	2024-10-25 00:00:00	20	3635.01
6	U00000054	ElizabethWilliams652	2024-06-27 00:00:00	16	3558.29
7	U00000023	BarbaraGarcia266	2024-01-15 00:00:00	14	3354.77
8	U00000021	JohnTaylor338	2023-04-14 00:00:00	15	3301.27
9	U00000036	RobertJones242	2023-09-10 00:00:00	19	3228.20
10	U00000089	MichaelTaylor510	2023-03-02 00:00:00	13	3196.27
11	U00000057	SusanTaylor926	2023-11-11 00:00:00	14	3147.13
12	U00000064	SusanRodriguez873	2023-07-25 00:00:00	13	3135.04
13	U00000030	BarbaraThomas894	2023-02-27 00:00:00	13	3028.21
14	U00000046	JenniferMoore649	2023-10-01 00:00:00	10	3017.87
15	U00000065	LindaJones508	2024-10-27 00:00:00	19	2976.99

5. Product managers want to identify "premium products" within each brand (priced above that brand's average) to develop differentiated pricing strategies or promotional plans.

Plan:
Find products that are more expensive than the average price in their category

```
SELECT
    p1.product_id,
    p1.product_name,
    p1.manufacturer,
    p1.price,
    (SELECT AVG(p2.price)
     FROM products p2
     WHERE p2.manufacturer = p1.manufacturer) AS avg_manufacturer_price
FROM products p1
WHERE p1.price > (
    SELECT AVG(p2.price)
    FROM products p2
    WHERE p2.manufacturer = p1.manufacturer
)
ORDER BY p1.manufacturer, p1.price DESC;
```

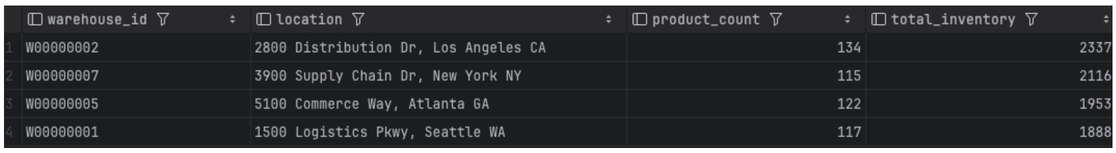


product_id	product_name	manufacturer	price	avg_manufacturer_price
P00000453	Highland Notes	3M Company	12.26	10.166000
P00000460	Post-it Notes 3x3	3M Company	12.25	10.166000
P00000455	Post-it Notes 3x3	3M Company	11.84	10.166000
P00000459	Post-it Notes 3x3	3M Company	11.83	10.166000
P00000458	Post-it Notes 3x3	3M Company	11.75	10.166000
P00000454	Post-it Notes 3x3	3M Company	11.45	10.166000
P00000457	Post-it Notes 3x3	3M Company	11.27	10.166000
P00000452	Highland Notes	3M Company	11.25	10.166000
P00000456	Highland Notes	3M Company	11.11	10.166000
P00000451	Post-it Notes 3x3	3M Company	11.02	10.166000
P00000294	USB-C Cable 6ft	Anker	27.49	24.614000
P00000300	USB-C Cable 6ft	Anker	27.42	24.614000
P00000296	Belkin Cable	Anker	26.26	24.614000
P00000299	USB-C Cable 6ft	Anker	24.97	24.614000
P00000017	Mineral Water 550ml	Aquafina Inc	2.18	2.013000

6. Operations directors need to identify warehouses with "excess inventory" for inventory redistribution or clearance promotions to avoid capital tie-up.

Plan:
Find warehouses with above-average inventory levels

```
SELECT
    w.warehouse_id,
    w.location,
    (SELECT COUNT(*)
     FROM store_records sr1
     WHERE sr1.warehouse_id = w.warehouse_id) AS product_count,
    (SELECT SUM(sr2.storequantity)
     FROM store_records sr2
     WHERE sr2.warehouse_id = w.warehouse_id) AS total_inventory
FROM warehouses w
WHERE (
    SELECT SUM(sr3.storequantity)
    FROM store_records sr3
    WHERE sr3.warehouse_id = w.warehouse_id
) > (
    SELECT AVG(warehouse_total)
    FROM (
       SELECT SUM(sr4.storequantity) AS warehouse_total
       FROM store_records sr4
       GROUP BY sr4.warehouse_id
    ) AS avg_calc
)
ORDER BY total_inventory DESC;
```

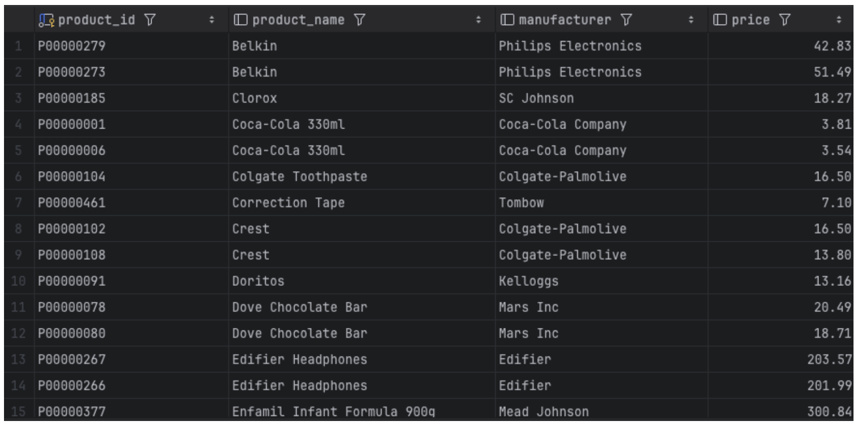


	warehouse_id	location	product_count	total_inventory
1	W00000002	2800 Distribution Dr, Los Angeles CA	134	2337
2	W00000007	3900 Supply Chain Dr, New York NY	115	2116
3	W00000005	5100 Commerce Way, Atlanta GA	122	1953
4	W00000001	1500 Logistics Pkwy, Seattle WA	117	1888

7. Product managers discover some products have never been purchased and need to identify these "slow-moving items" to decide whether to delist them or run special promotions.

Plan:
Find products that have NEVER been ordered

```
SELECT
   p.product_id,
   p.product_name,
   p.manufacturer,
   p.price
FROM products p
WHERE NOT EXISTS (
   SELECT 1
   FROM inform i
   WHERE i.product_id = p.product_id
)
ORDER BY p.product_name;
```

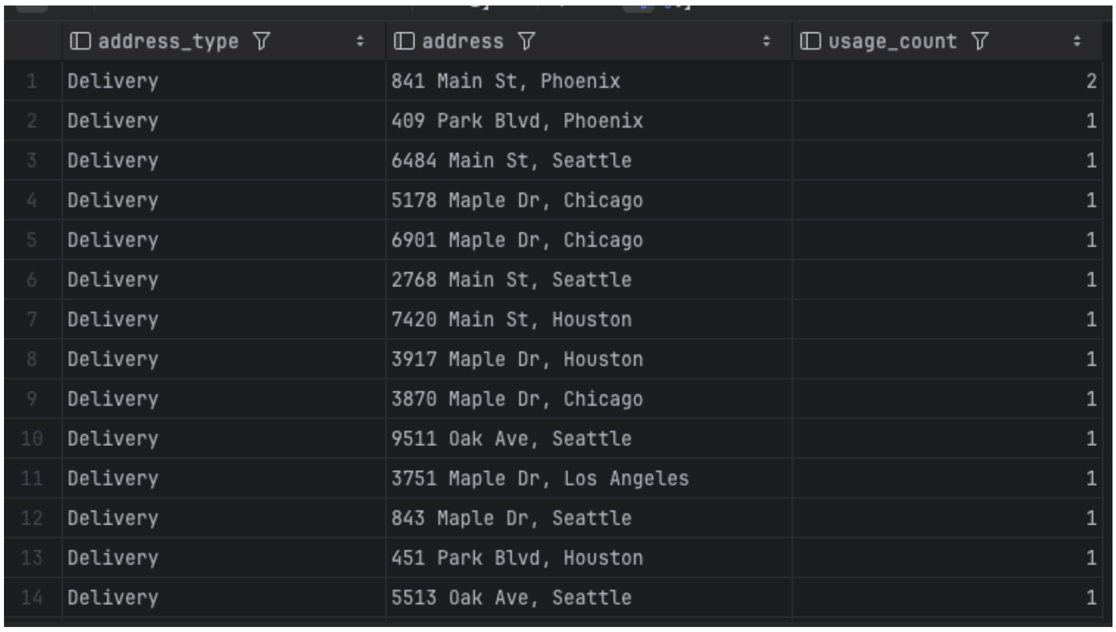


	product_id	product_name	manufacturer	price
1	P00000279	Belkin	Philips Electronics	42.83
2	P00000273	Belkin	Philips Electronics	51.49
3	P00000185	Clorox	SC Johnson	18.27
4	P00000001	Coca-Cola 330ml	Coca-Cola Company	3.81
5	P00000006	Coca-Cola 330ml	Coca-Cola Company	3.54
6	P00000104	Colgate Toothpaste	Colgate-Palmolive	16.50
7	P00000461	Correction Tape	Tombow	7.10
8	P00000102	Crest	Colgate-Palmolive	16.50
9	P00000108	Crest	Colgate-Palmolive	13.80
10	P00000091	Doritos	Kelloggs	13.16
11	P00000078	Dove Chocolate Bar	Mars Inc	20.49
12	P00000080	Dove Chocolate Bar	Mars Inc	18.71
13	P00000267	Edifier Headphones	Edifier	203.57
14	P00000266	Edifier Headphones	Edifier	201.99
15	P00000377	Enfamil Infant Formula 900g	Mead Johnson	300.84

8. The logistics department needs to generate a "nationwide service network map," requiring integration of all types of address information in the system (delivery addresses, shipping addresses, warehouse locations, supplier addresses).

Plan:
Get all addresses used in the system (delivery, shipping, warehouse, supplier)

```
SELECT 'Delivery' AS address_type, delivery_address AS address, COUNT(*) AS
usage_count
FROM delivery_product
GROUP BY delivery_address

UNION

SELECT 'Shipping' AS address_type, shipping_address AS address, COUNT(*) AS
usage_count
FROM shipping_product
GROUP BY shipping_address

UNION

SELECT 'Warehouse' AS address_type, location AS address, 1 AS usage_count
FROM warehouses

UNION

SELECT 'Supplier' AS address_type, address AS address, 1 AS usage_count
FROM suppliers

ORDER BY address_type, usage_count DESC;
```



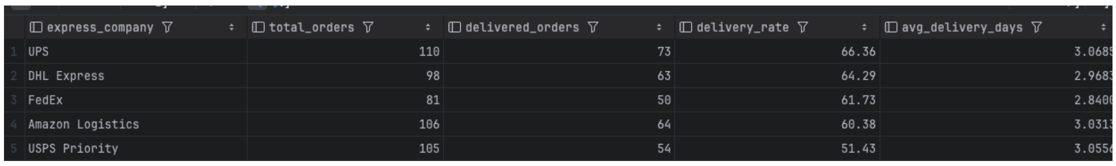
	address_type	address	usage_count
1	Delivery	841 Main St, Phoenix	2
2	Delivery	409 Park Blvd, Phoenix	1
3	Delivery	6484 Main St, Seattle	1
4	Delivery	5178 Maple Dr, Chicago	1
5	Delivery	6901 Maple Dr, Chicago	1
6	Delivery	2768 Main St, Seattle	1
7	Delivery	7420 Main St, Houston	1
8	Delivery	3917 Maple Dr, Houston	1
9	Delivery	3870 Maple Dr, Chicago	1
10	Delivery	9511 Oak Ave, Seattle	1
11	Delivery	3751 Maple Dr, Los Angeles	1
12	Delivery	843 Maple Dr, Seattle	1
13	Delivery	451 Park Blvd, Houston	1
14	Delivery	5513 Oak Ave, Seattle	1

9. Logistics directors need to evaluate courier companies' service quality (delivery success rate, average delivery days) to decide partner selection and quota allocation for the next quarter.

Plan:
Analyze order fulfillment performance by the logistics company

```
SELECT
   express_company,
   total_orders,
   delivered_orders,
   ROUND(delivered_orders * 100.0 / total_orders, 2) AS delivery_rate,
   avg_delivery_days
FROM (
   SELECT
      l.express_company,
      COUNT(*) AS total_orders,
      SUM(CASE WHEN l.logistics_status = 'delivered' THEN 1 ELSE 0 END) AS
delivered_orders,
      AVG(
         CASE
            WHEN l.logistics_status = 'delivered'
            THEN DATEDIFF(dp.delivery_time, sp.shipping_time)
            ELSE NULL
         END
      ) AS avg_delivery_days
   FROM logistics l
   LEFT JOIN shipping_product sp ON l.logistics_id = sp.logistics_id
   LEFT JOIN delivery_product dp ON l.logistics_id = dp.logistics_id
   GROUP BY l.express_company
) AS logistics_stats
```

```
ORDER BY delivery_rate DESC, avg_delivery_days ASC;
```

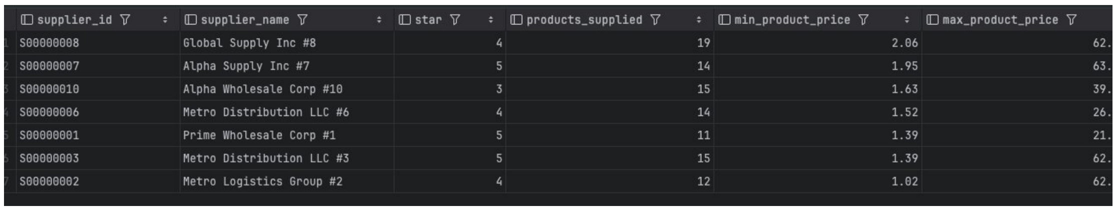


	express_company	total_orders	delivered_orders	delivery_rate	avg_delivery_days
1	UPS	110	73	66.36	3.0685
2	DHL Express	98	63	64.29	2.9683
3	FedEx	81	50	61.73	2.8400
4	Amazon Logistics	106	64	60.38	3.0313
5	USPS Priority	105	54	51.43	3.0556

10. **The procurement department wants to find suppliers who "only supply premium products" (all products priced ≥$1) to establish exclusive supply chain partnerships for premium product lines.**

Plan:
Find suppliers who supply products that are ALL above $1

```
SELECT DISTINCT
    s.supplier_id,
    s.supplier_name,
    s.star,
    COUNT(DISTINCT gs.product_id) AS products_supplied,
    MIN(p.price) AS min_product_price,
    MAX(p.price) AS max_product_price
FROM suppliers s
JOIN good_supply gs ON s.supplier_id = gs.supplier_id
JOIN products p ON gs.product_id = p.product_id
WHERE s.supplier_id NOT IN (
    -- Exclude suppliers who have ANY product below $1
    SELECT DISTINCT gs2.supplier_id
    FROM good_supply gs2
    JOIN products p2 ON gs2.product_id = p2.product_id
    WHERE p2.price < 1
)
GROUP BY s.supplier_id, s.supplier_name, s.star
HAVING COUNT(DISTINCT gs.product_id) > 0
ORDER BY min_product_price DESC;
```

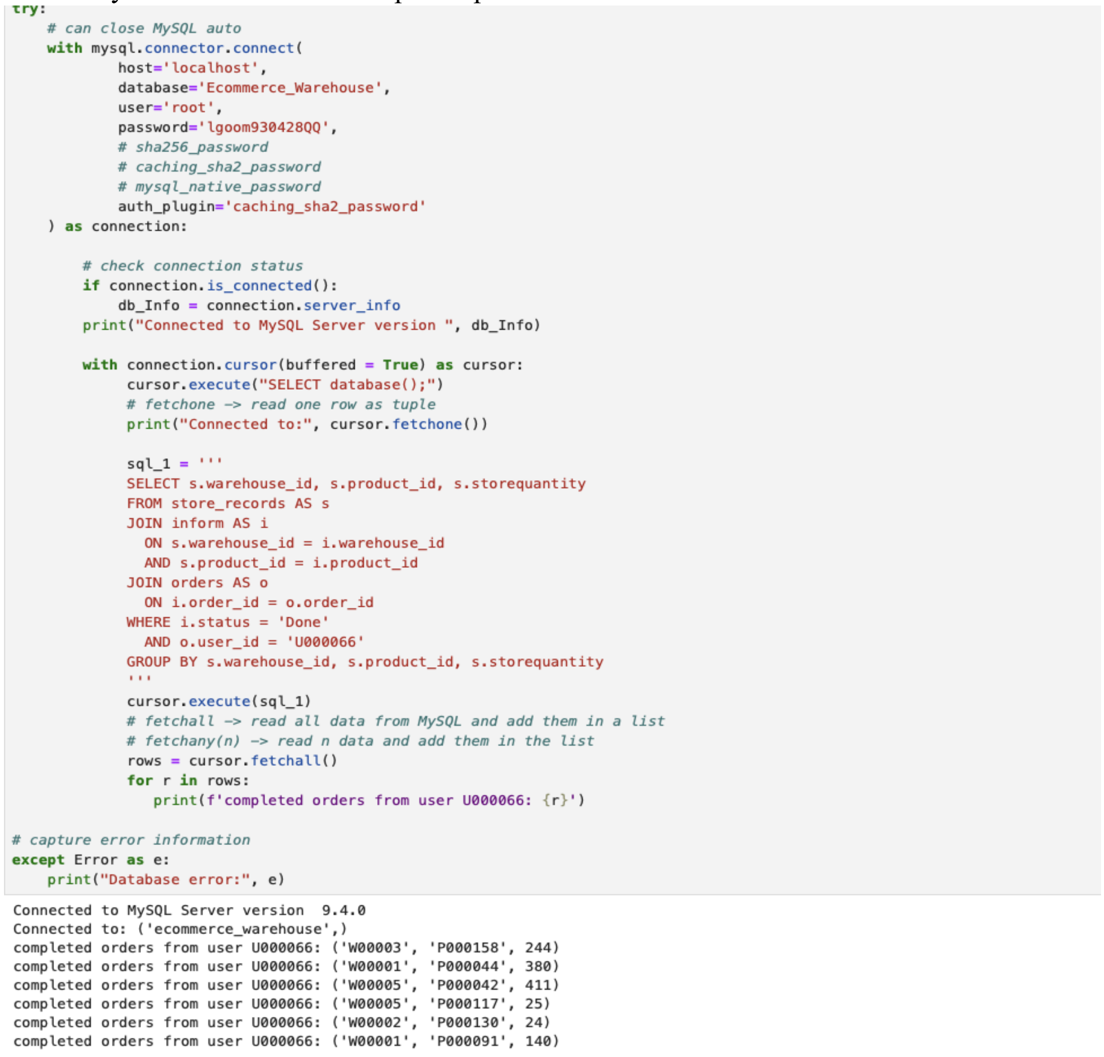


	supplier_id	supplier_name	star	products_supplied	min_product_price	max_product_price
1	S00000008	Global Supply Inc #8	4	19	2.06	62.
2	S00000007	Alpha Supply Inc #7	5	14	1.95	63.
3	S00000010	Alpha Wholesale Corp #10	3	15	1.63	39.
4	S00000006	Metro Distribution LLC #6	4	14	1.52	26.
5	S00000001	Prime Wholesale Corp #1	5	11	1.39	21.
6	S00000003	Metro Distribution LLC #3	5	15	1.39	62.
7	S00000002	Metro Logistics Group #2	4	12	1.02	62.

(2) MySQL in Python Code:

1. This query retrieves all completed orders from user U000066 and returns the related product_id, warehouse_id, and orderquantity. It helps identify what products the user

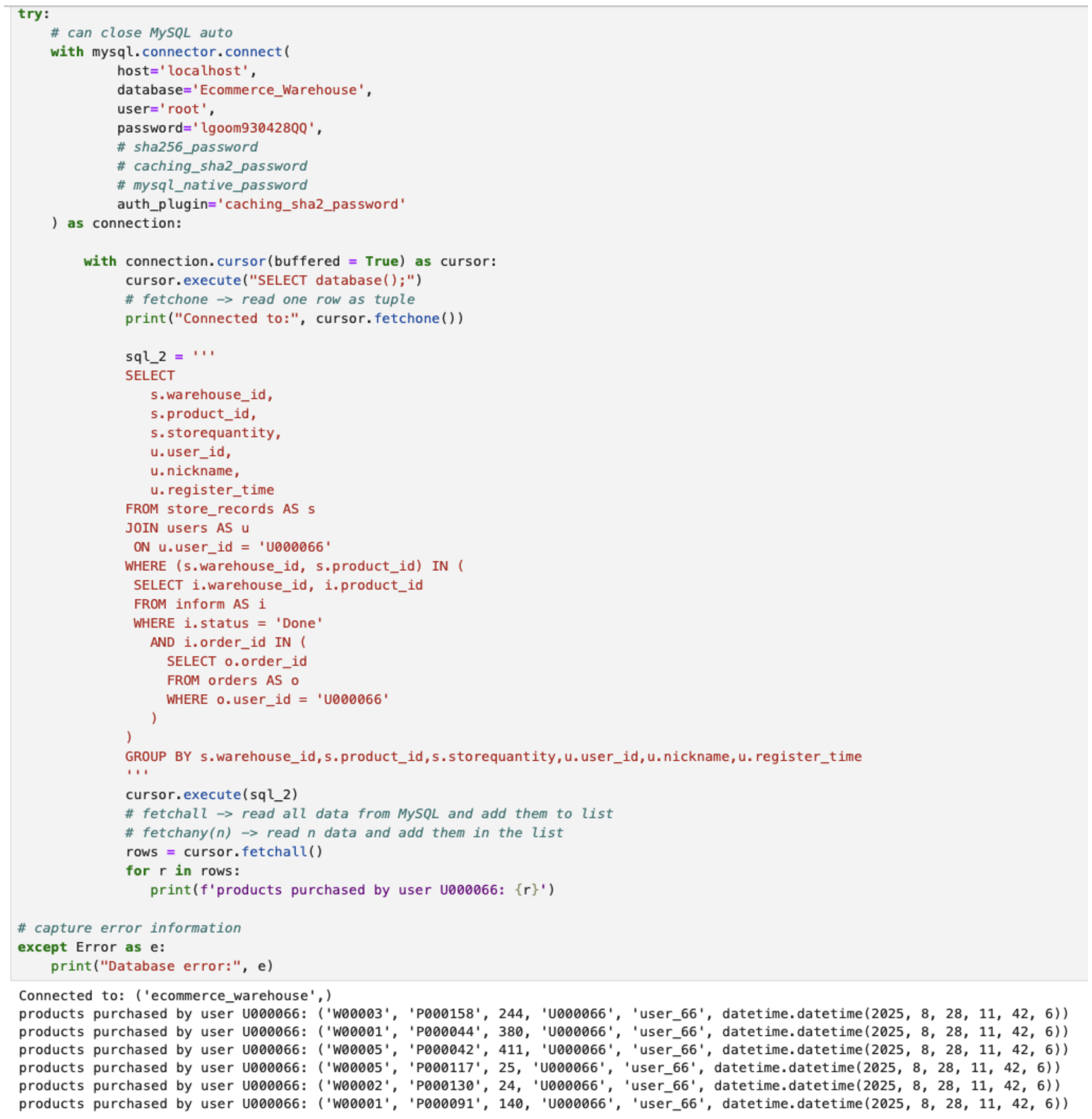
purchased, from which warehouse, and in what quantities. This information is useful for understanding user buying behavior, analyzing fulfillment flows, checking inventory movements, and preparing features for customer analytics or recommendation models. It essentially reveals the user's completed purchase details and their warehouse associations.



```
try:
    # can close MySQL auto
    with mysql.connector.connect(
            host='localhost',
            database='Ecommerce_Warehouse',
            user='root',
            password='lgoom930428QQ',
            # sha256_password
            # caching_sha2_password
            # mysql_native_password
            auth_plugin='caching_sha2_password'
    ) as connection:

        # check connection status
        if connection.is_connected():
            db_Info = connection.server_info
        print("Connected to MySQL Server version ", db_Info)

        with connection.cursor(buffered = True) as cursor:
             cursor.execute("SELECT database();")
             # fetchone -> read one row as tuple
             print("Connected to:", cursor.fetchone())

             sql_1 = '''
             SELECT s.warehouse_id, s.product_id, s.storequantity
             FROM store_records AS s
             JOIN inform AS i
               ON s.warehouse_id = i.warehouse_id
               AND s.product_id = i.product_id
             JOIN orders AS o
               ON i.order_id = o.order_id
             WHERE i.status = 'Done'
               AND o.user_id = 'U000066'
             GROUP BY s.warehouse_id, s.product_id, s.storequantity
             '''
             cursor.execute(sql_1)
             # fetchall -> read all data from MySQL and add them in a list
             # fetchany(n) -> read n data and add them in the list
             rows = cursor.fetchall()
             for r in rows:
                print(f'completed orders from user U000066: {r}')

# capture error information
except Error as e:
    print("Database error:", e)
```

```
Connected to MySQL Server version  9.4.0
Connected to: ('ecommerce_warehouse',)
completed orders from user U000066: ('W00003', 'P000158', 244)
completed orders from user U000066: ('W00001', 'P000044', 380)
completed orders from user U000066: ('W00005', 'P000042', 411)
completed orders from user U000066: ('W00005', 'P000117', 25)
completed orders from user U000066: ('W00002', 'P000130', 24)
completed orders from user U000066: ('W00001', 'P000091', 140)
```

2. This query lists the inventory levels for all products purchased by user U000066. It joins store_records, inform, orders, and users to connect warehouse inventory with the user's completed orders. The output includes the warehouse, product, current store quantity, and the user's nickname and register_time. This helps understand what the user bought and how much inventory remains across warehouses, while also referencing the user's profile information.

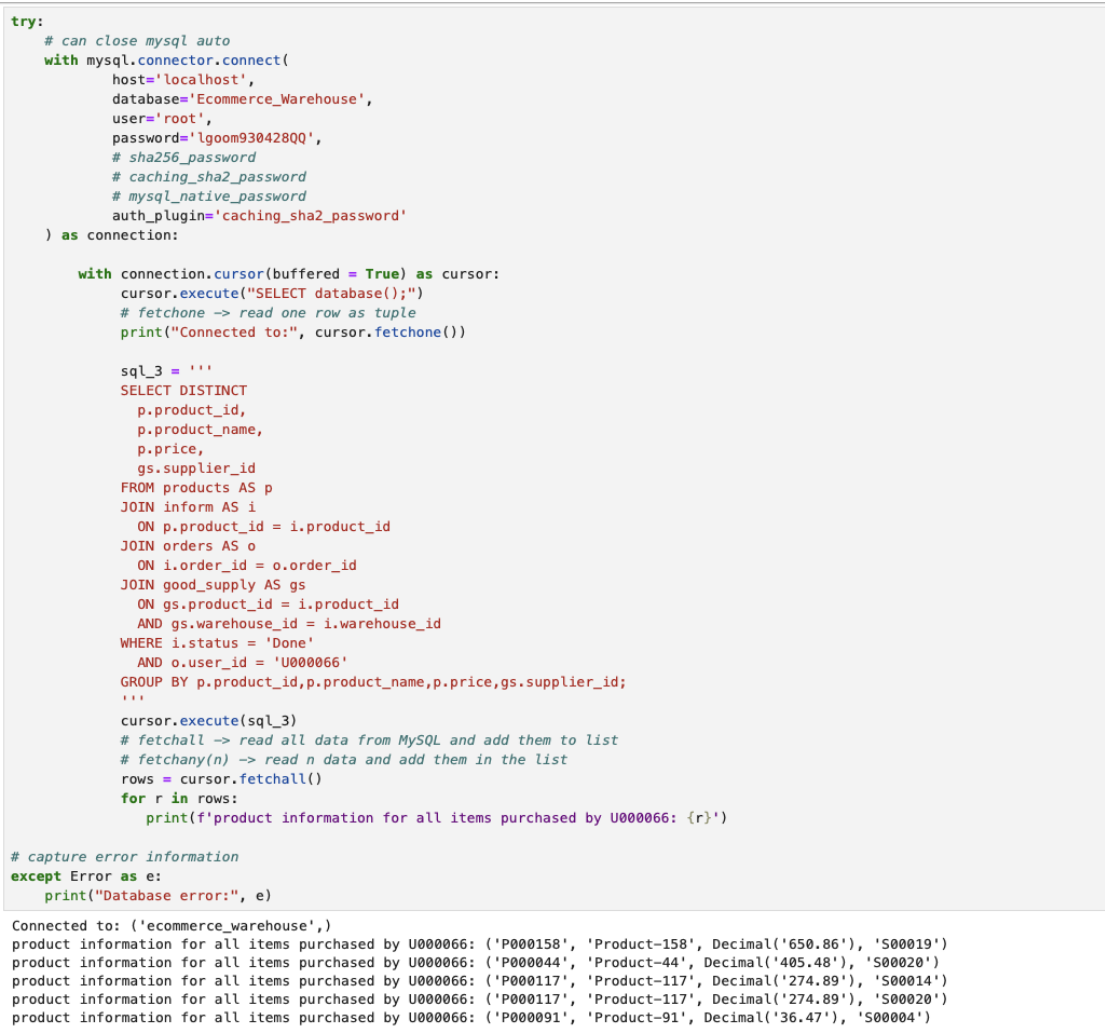


```python
try:
    # can close MySQL auto
    with mysql.connector.connect(
            host='localhost',
            database='Ecommerce_Warehouse',
            user='root',
            password='lgoom930428QQ',
            # sha256_password
            # caching_sha2_password
            # mysql_native_password
            auth_plugin='caching_sha2_password'
    ) as connection:

        with connection.cursor(buffered = True) as cursor:
             cursor.execute("SELECT database();")
             # fetchone -> read one row as tuple
             print("Connected to:", cursor.fetchone())

             sql_2 = '''
             SELECT
                s.warehouse_id,
                s.product_id,
                s.storequantity,
                u.user_id,
                u.nickname,
                u.register_time
             FROM store_records AS s
             JOIN users AS u
              ON u.user_id = 'U000066'
             WHERE (s.warehouse_id, s.product_id) IN (
              SELECT i.warehouse_id, i.product_id
              FROM inform AS i
              WHERE i.status = 'Done'
                AND i.order_id IN (
                  SELECT o.order_id
                  FROM orders AS o
                  WHERE o.user_id = 'U000066'
                )
             )
             GROUP BY s.warehouse_id,s.product_id,s.storequantity,u.user_id,u.nickname,u.register_time
             '''
             cursor.execute(sql_2)
             # fetchall -> read all data from MySQL and add them to list
             # fetchany(n) -> read n data and add them in the list
             rows = cursor.fetchall()
             for r in rows:
                print(f'products purchased by user U000066: {r}')

# capture error information
except Error as e:
    print("Database error:", e)
```

```
Connected to: ('ecommerce_warehouse',)
products purchased by user U000066: ('W00003', 'P000158', 244, 'U000066', 'user_66', datetime.datetime(2025, 8, 28, 11, 42, 6))
products purchased by user U000066: ('W00001', 'P000044', 380, 'U000066', 'user_66', datetime.datetime(2025, 8, 28, 11, 42, 6))
products purchased by user U000066: ('W00005', 'P000042', 411, 'U000066', 'user_66', datetime.datetime(2025, 8, 28, 11, 42, 6))
products purchased by user U000066: ('W00005', 'P000117', 25, 'U000066', 'user_66', datetime.datetime(2025, 8, 28, 11, 42, 6))
products purchased by user U000066: ('W00002', 'P000130', 24, 'U000066', 'user_66', datetime.datetime(2025, 8, 28, 11, 42, 6))
products purchased by user U000066: ('W00001', 'P000091', 140, 'U000066', 'user_66', datetime.datetime(2025, 8, 28, 11, 42, 6))
```

3. This query helps analyze all products that user U000066 has purchased by linking order data with product details and supplier information. It shows which products the user buys, how much those products cost, and which suppliers provide them. With this information, we can study the user's purchasing preferences, evaluate product assortment quality, understand supplier performance, and identify which suppliers contribute most to this user's completed orders. It

also supports further analysis, such as pricing patterns, supplier reliability, product popularity, and building user-level product profiles for recommendation, segmentation, or supply-chain planning.

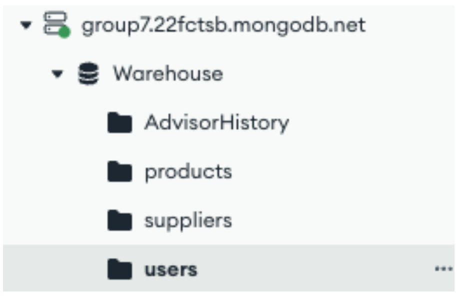


```python
try:
    # can close mysql auto
    with mysql.connector.connect(
            host='localhost',
            database='Ecommerce_Warehouse',
            user='root',
            password='lgoom930428QQ',
            # sha256_password
            # caching_sha2_password
            # mysql_native_password
            auth_plugin='caching_sha2_password'
    ) as connection:

        with connection.cursor(buffered = True) as cursor:
            cursor.execute("SELECT database();")
            # fetchone -> read one row as tuple
            print("Connected to:", cursor.fetchone())

            sql_3 = '''
            SELECT DISTINCT
              p.product_id,
              p.product_name,
              p.price,
              gs.supplier_id
            FROM products AS p
            JOIN inform AS i
              ON p.product_id = i.product_id
            JOIN orders AS o
              ON i.order_id = o.order_id
            JOIN good_supply AS gs
              ON gs.product_id = i.product_id
              AND gs.warehouse_id = i.warehouse_id
            WHERE i.status = 'Done'
              AND o.user_id = 'U000066'
            GROUP BY p.product_id,p.product_name,p.price,gs.supplier_id;
            '''
            cursor.execute(sql_3)
            # fetchall -> read all data from MySQL and add them to list
            # fetchany(n) -> read n data and add them in the list
            rows = cursor.fetchall()
            for r in rows:
               print(f'product information for all items purchased by U000066: {r}')

# capture error information
except Error as e:
    print("Database error:", e)
```

```
Connected to: ('ecommerce_warehouse',)
product information for all items purchased by U000066: ('P000158', 'Product-158', Decimal('650.86'), 'S00019')
product information for all items purchased by U000066: ('P000044', 'Product-44', Decimal('405.48'), 'S00020')
product information for all items purchased by U000066: ('P000117', 'Product-117', Decimal('274.89'), 'S00014')
product information for all items purchased by U000066: ('P000117', 'Product-117', Decimal('274.89'), 'S00020')
product information for all items purchased by U000066: ('P000091', 'Product-91', Decimal('36.47'), 'S00004')
```

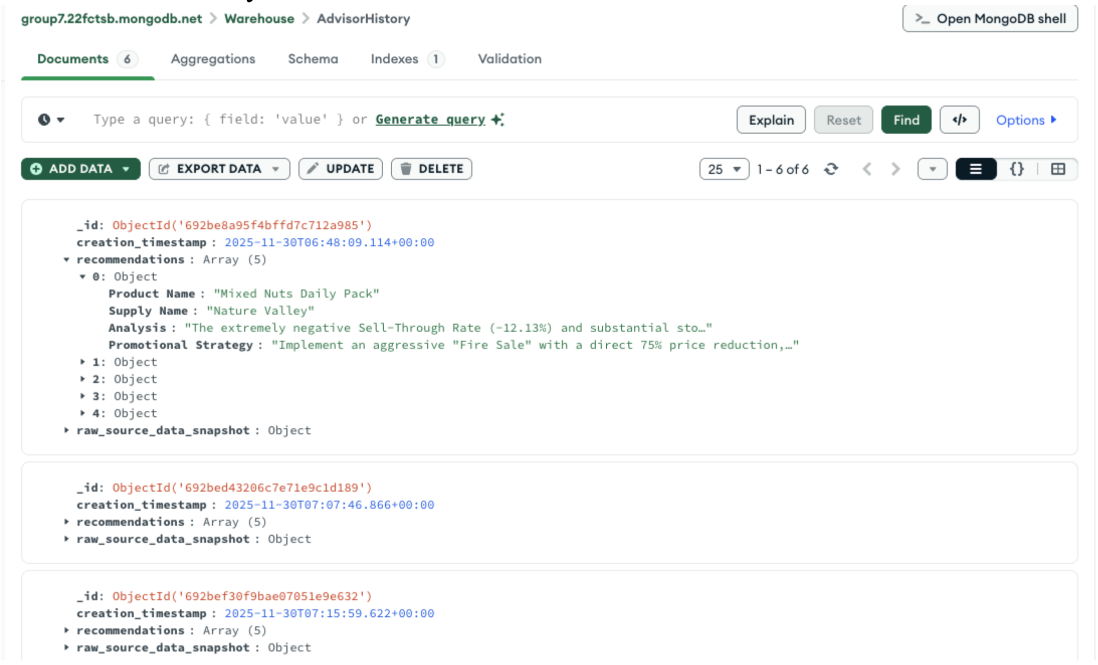
(4) NoSQL:

We create four collections in our MongoDB: AdvisorHistory, which stores AI sales advice; products, which stores product information; suppliers, which stores supplier information; and users, which stores user information. Because the system needs both the strong consistency and relational integrity of SQL and the high-performance, highly scalable read capabilities of NoSQL, the same entities (products, suppliers, users) exist in both databases—SQL as the source of truth and NoSQL as the fast-access layer.



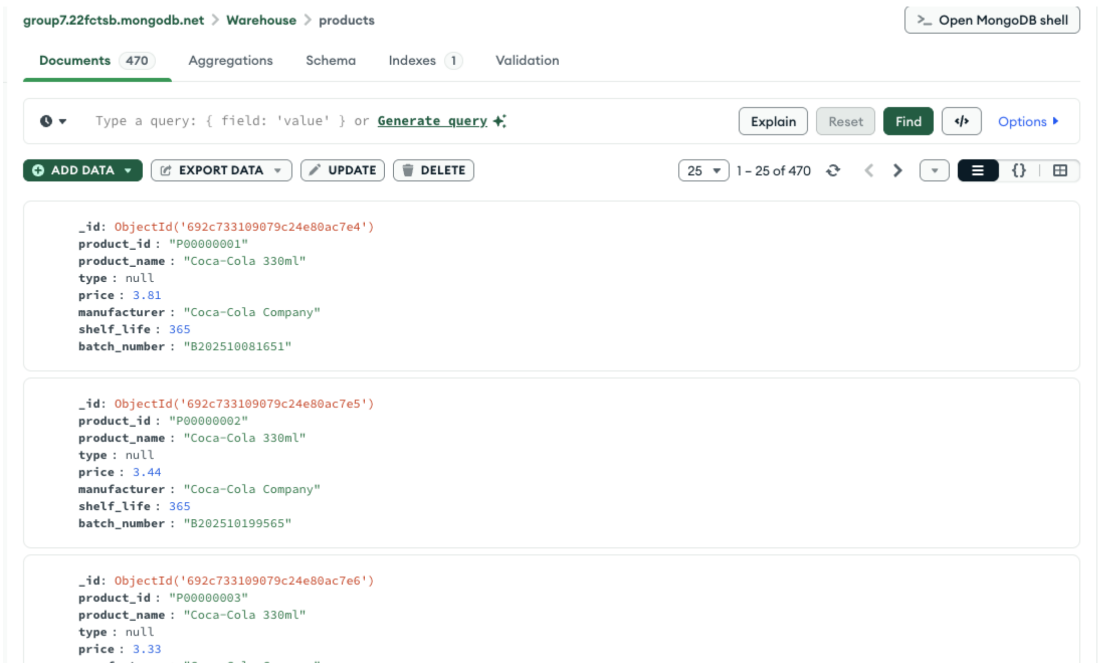
The record in collections:

1. AdvisorHistory



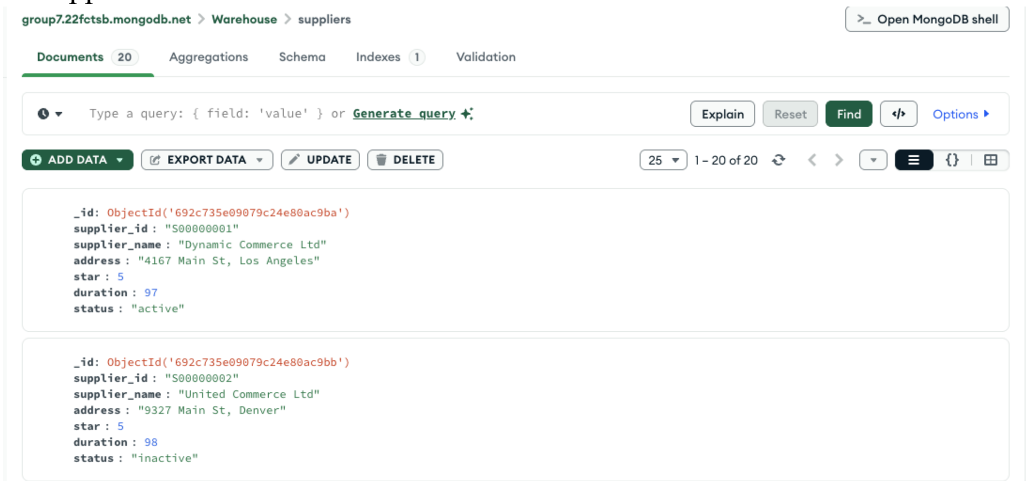
```
_id: ObjectId('692be8a95f4bffd7c712a985')
creation_timestamp : 2025-11-30T06:48:09.114+00:00
▾ recommendations : Array (5)
  ▾ 0: Object
      Product Name : "Mixed Nuts Daily Pack"
      Supply Name : "Nature Valley"
      Analysis : "The extremely negative Sell-Through Rate (-12.13%) and substantial sto…"
      Promotional Strategy : "Implement an aggressive "Fire Sale" with a direct 75% price reduction,…"
  ▸ 1: Object
  ▸ 2: Object
  ▸ 3: Object
  ▸ 4: Object
▸ raw_source_data_snapshot : Object

_id: ObjectId('692bed43206c7e71e9c1d189')
creation_timestamp : 2025-11-30T07:07:46.866+00:00
▸ recommendations : Array (5)
▸ raw_source_data_snapshot : Object

_id: ObjectId('692bef30f9bae07051e9e632')
creation_timestamp : 2025-11-30T07:15:59.622+00:00
▸ recommendations : Array (5)
▸ raw_source_data_snapshot : Object
```

2. products



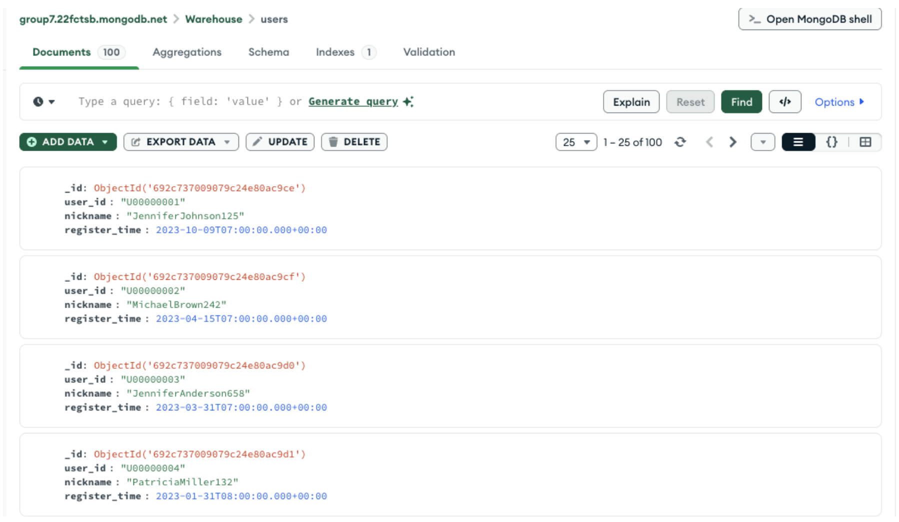
```
_id: ObjectId('692c733109079c24e80ac7e4')
product_id : "P00000001"
product_name : "Coca-Cola 330ml"
type : null
price : 3.81
manufacturer : "Coca-Cola Company"
shelf_life : 365
batch_number : "B202510081651"

_id: ObjectId('692c733109079c24e80ac7e5')
product_id : "P00000002"
product_name : "Coca-Cola 330ml"
type : null
price : 3.44
manufacturer : "Coca-Cola Company"
shelf_life : 365
batch_number : "B202510199565"

_id: ObjectId('692c733109079c24e80ac7e6')
product_id : "P00000003"
product_name : "Coca-Cola 330ml"
type : null
price : 3.33
```

3. suppliers



```
_id: ObjectId('692c735e09079c24e80ac9ba')
supplier_id : "S00000001"
supplier_name : "Dynamic Commerce Ltd"
address : "4167 Main St, Los Angeles"
star : 5
duration : 97
status : "active"

_id: ObjectId('692c735e09079c24e80ac9bb')
supplier_id : "S00000002"
supplier_name : "United Commerce Ltd"
address : "9327 Main St, Denver"
star : 5
duration : 98
status : "inactive"
```

4. users

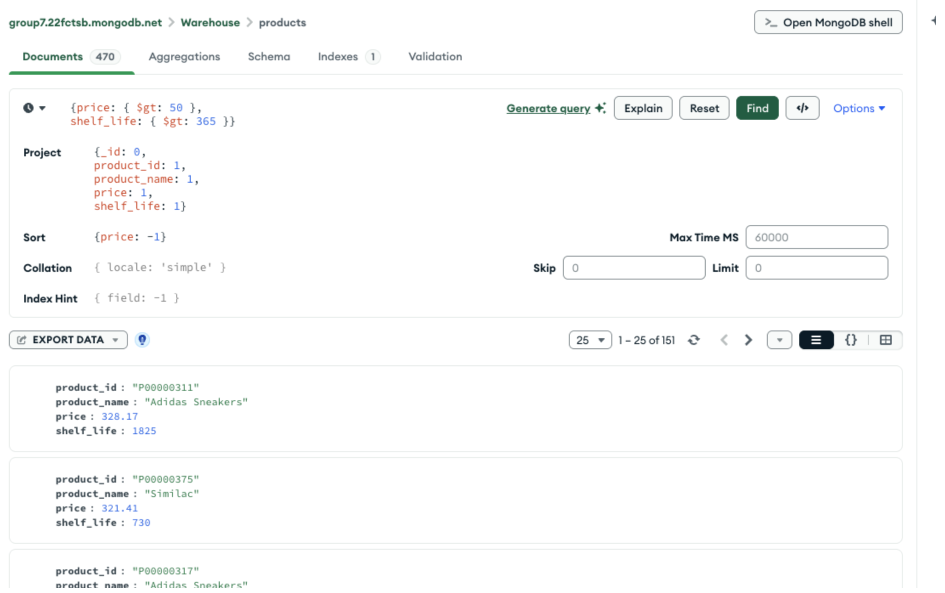


```
_id: ObjectId('692c737009079c24e80ac9ce')
user_id : "U00000001"
nickname : "JenniferJohnson125"
register_time : 2023-10-09T07:00:00.000+00:00

_id: ObjectId('692c737009079c24e80ac9cf')
user_id : "U00000002"
nickname : "MichaelBrown242"
register_time : 2023-04-15T07:00:00.000+00:00

_id: ObjectId('692c737009079c24e80ac9d0')
user_id : "U00000003"
nickname : "JenniferAnderson658"
register_time : 2023-03-31T07:00:00.000+00:00

_id: ObjectId('692c737009079c24e80ac9d1')
user_id : "U00000004"
nickname : "PatriciaMiller132"
register_time : 2023-01-31T08:00:00.000+00:00
```

1. In a warehouse management system, high-value products with long shelf life should be

prioritized during procurement to reduce expiration risks and maintain inventory quality.

Specifics:
In the `products` collection, query products with a unit price greater than 50 and a shelf life greater than 365 days.
Return only the `product_id`, `product_name`, `price`, `shelf_life`, and `shelf_life` fields, sorted by price from highest to lowest.

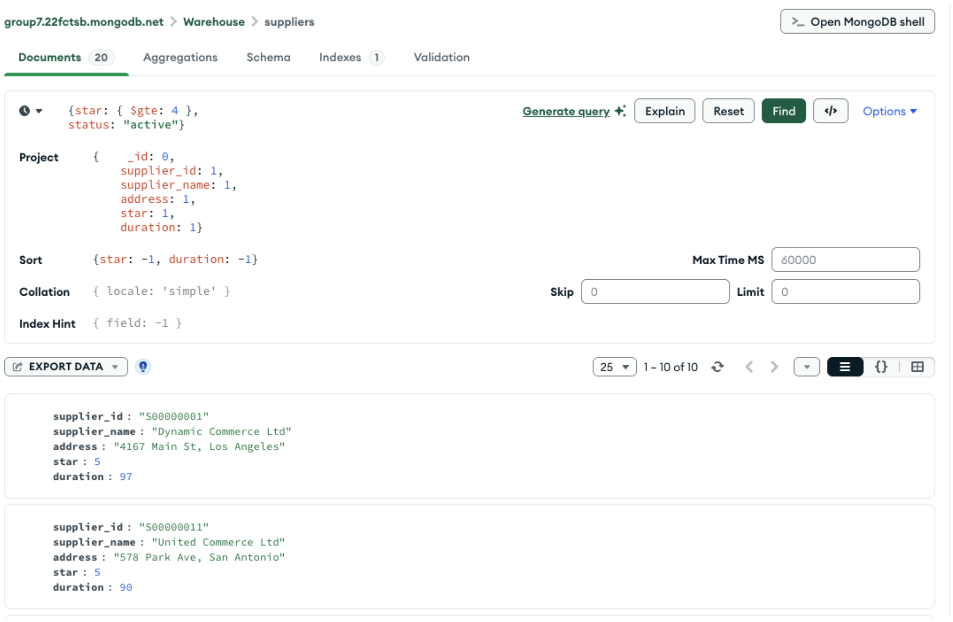


```
{price: { $gt: 50 },
shelf_life: { $gt: 365 }}

Project    {_id: 0,
            product_id: 1,
            product_name: 1,
            price: 1,
            shelf_life: 1}

Sort       {price: -1}                     Max Time MS  60000

Collation  { locale: 'simple' }     Skip  0      Limit  0

Index Hint { field: -1 }
```

```
product_id : "P00000311"
product_name : "Adidas Sneakers"
price : 328.17
shelf_life : 1825

product_id : "P00000375"
product_name : "Similac"
price : 321.41
shelf_life : 730

product_id : "P00000317"
product_name : "Adidas Sneakers"
```

2. The warehouse supply chain depends on reliable suppliers. Suppliers with high ratings who are currently active are typically more stable and suitable for long-term cooperation.

Specifics:
In the suppliers collection, find suppliers with a star rating of 4 or higher and a status of "active". Return the `supplier_id`, `supplier_name`, `address`, `star`, and `duration` fields, sorting by star rating from highest to lowest, and by duration from highest to lowest if star ratings are the same.



```
{star: { $gte: 4 },
 status: "active"}
```

Project
```
{   _id: 0,
    supplier_id: 1,
    supplier_name: 1,
    address: 1,
    star: 1,
    duration: 1}
```

Sort `{star: -1, duration: -1}`

Max Time MS 60000

Collation `{ locale: 'simple' }`

Skip 0 Limit 0

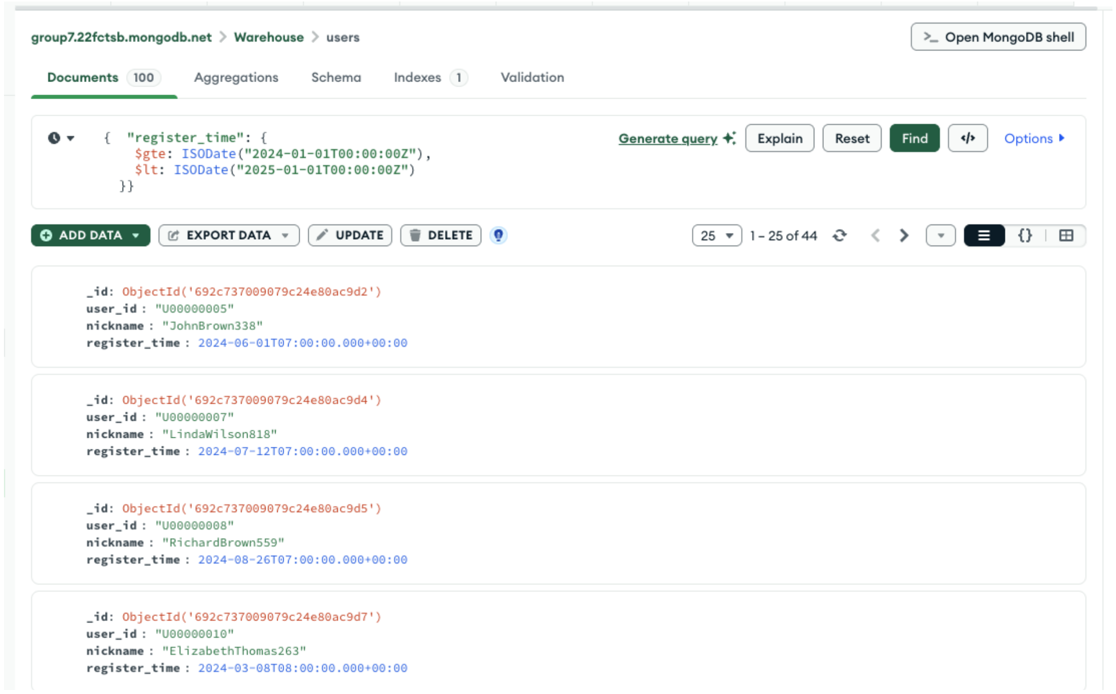
Index Hint `{ field: -1 }`

```
supplier_id : "S00000001"
supplier_name : "Dynamic Commerce Ltd"
address : "4167 Main St, Los Angeles"
star : 5
duration : 97
```

```
supplier_id : "S00000011"
supplier_name : "United Commerce Ltd"
address : "578 Park Ave, San Antonio"
star : 5
duration : 90
```

3. The platform's operations team needs to track the number of users registered each year to support growth evaluation, marketing analysis, and annual operational reporting.

Specifics:
Count the total number of users who registered in the `users` collection within 2024.
Assume the `register_time` field is of type Date.



```
{  "register_time": {
     $gte: ISODate("2024-01-01T00:00:00Z"),
     $lt: ISODate("2025-01-01T00:00:00Z")
  }}
```

```
_id: ObjectId('692c737009079c24e80ac9d2')
user_id : "U00000005"
nickname : "JohnBrown338"
register_time : 2024-06-01T07:00:00.000+00:00
```

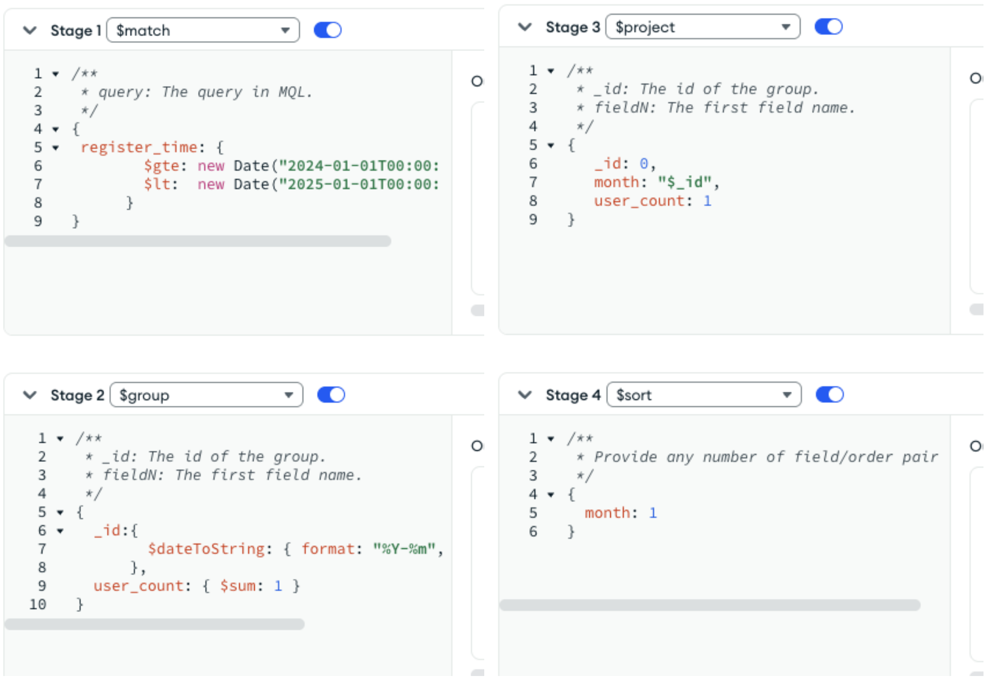
```
_id: ObjectId('692c737009079c24e80ac9d4')
user_id : "U00000007"
nickname : "LindaWilson818"
register_time : 2024-07-12T07:00:00.000+00:00
```

```
_id: ObjectId('692c737009079c24e80ac9d5')
user_id : "U00000008"
nickname : "RichardBrown559"
register_time : 2024-08-26T07:00:00.000+00:00
```

```
_id: ObjectId('692c737009079c24e80ac9d7')
user_id : "U00000010"
nickname : "ElizabethThomas263"
register_time : 2024-03-08T08:00:00.000+00:00
```

4. To analyze user growth trends, the platform needs monthly registration statistics to identify seasonal patterns, evaluate campaign effectiveness, and observe overall user growth dynamics.

Specifics:
For users who registered in 2024, calculate the number of registered users each month in the `users` collection. Output the fields: `month` (format "YYYY-MM") and `user_count`, sorted in ascending order by `month`.

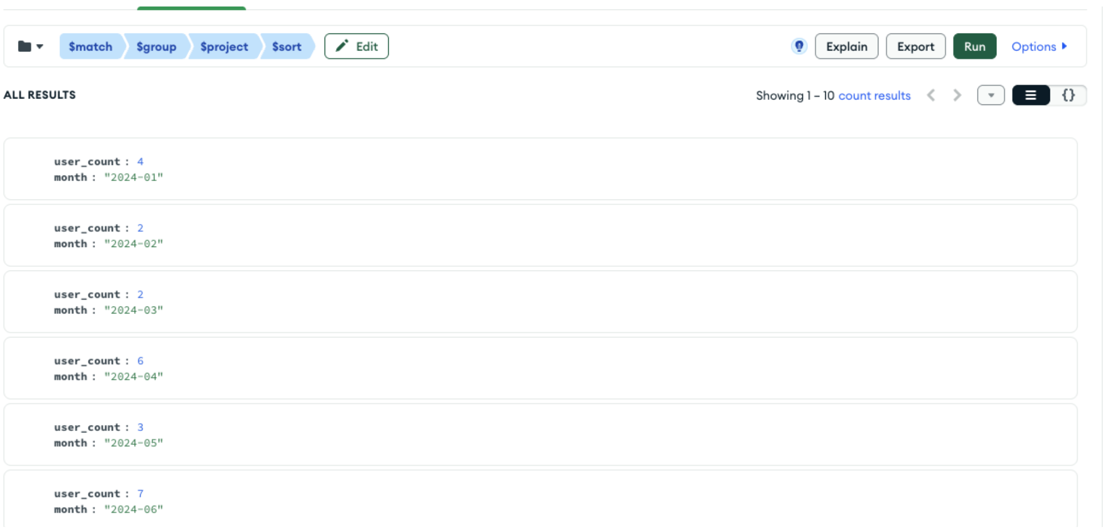


```
/**
 * query: The query in MQL.
 */
{
  register_time: {
         $gte: new Date("2024-01-01T00:00:
         $lt:  new Date("2025-01-01T00:00:
      }
  }
```

```
/**
 * _id: The id of the group.
 * fieldN: The first field name.
 */
{
   _id: 0,
   month: "$_id",
   user_count: 1
}
```

```
/**
 * _id: The id of the group.
 * fieldN: The first field name.
 */
{
  _id:{
        $dateToString: { format: "%Y-%m",
      },
  user_count: { $sum: 1 }
}
```

```
/**
 * Provide any number of field/order pair
 */
{
  month: 1
}
```



ALL RESULTS

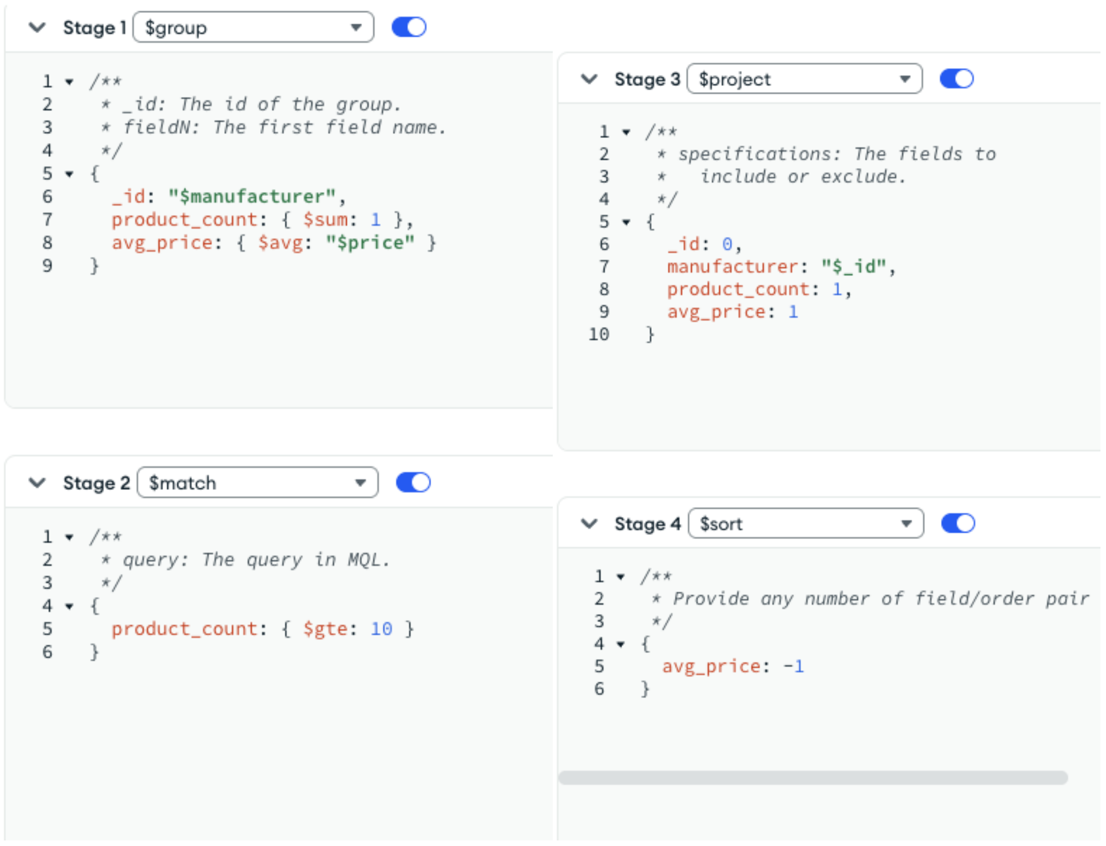
Showing 1 – 10 count results

```
user_count : 4
month : "2024-01"

user_count : 2
month : "2024-02"

user_count : 2
month : "2024-03"

user_count : 6
month : "2024-04"

user_count : 3
month : "2024-05"

user_count : 7
month : "2024-06"
```

5. Inventory structure analysis requires understanding each manufacturer's supply volume and pricing level, providing insights that support procurement decisions and supply chain optimization.

Specifics:
In the `products` collection, group products by manufacturer.

Count the product quantity (`product_count`) and average price (`avg_price`) for each manufacturer.
Only retain manufacturers with `product_count` >= 10.
Finally, sort by `avg_price` from highest to lowest and output the `manufacturer`, `product_count`, and `avg_price` fields.



Stage 1 $group

```
/**
 * _id: The id of the group.
 * fieldN: The first field name.
 */
{
  _id: "$manufacturer",
  product_count: { $sum: 1 },
  avg_price: { $avg: "$price" }
}
```

Stage 2 $match

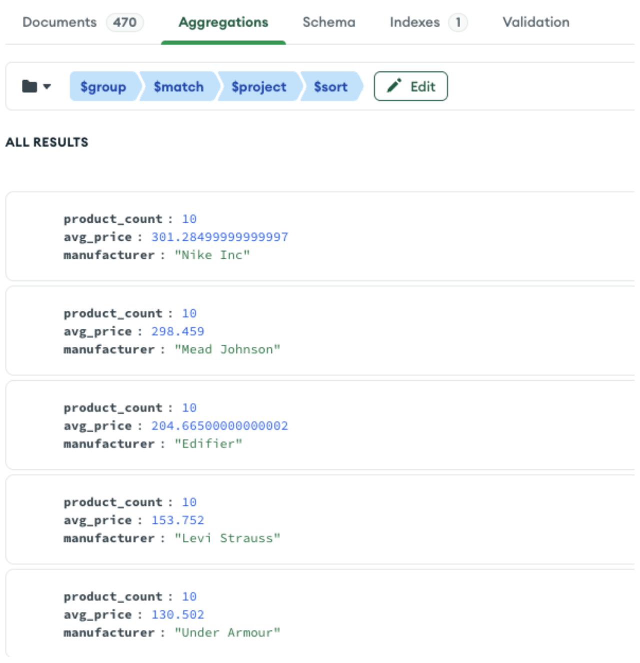
```
/**
 * query: The query in MQL.
 */
{
  product_count: { $gte: 10 }
}
```

Stage 3 $project

```
/**
 * specifications: The fields to
 *   include or exclude.
 */
{
  _id: 0,
  manufacturer: "$_id",
  product_count: 1,
  avg_price: 1
}
```

Stage 4 $sort

```
/**
 * Provide any number of field/order pair
 */
{
  avg_price: -1
}
```

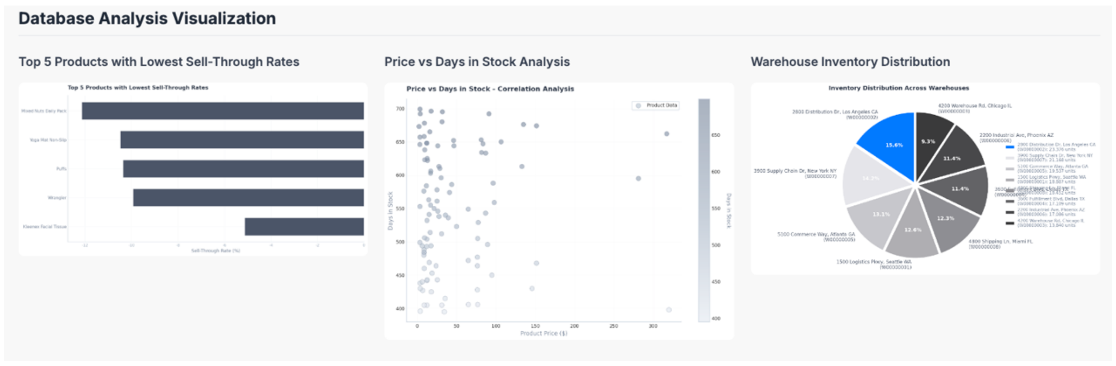


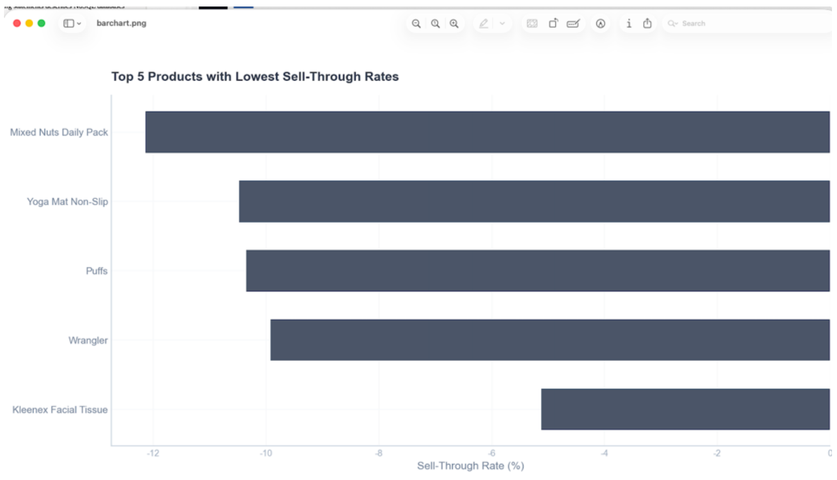
ALL RESULTS

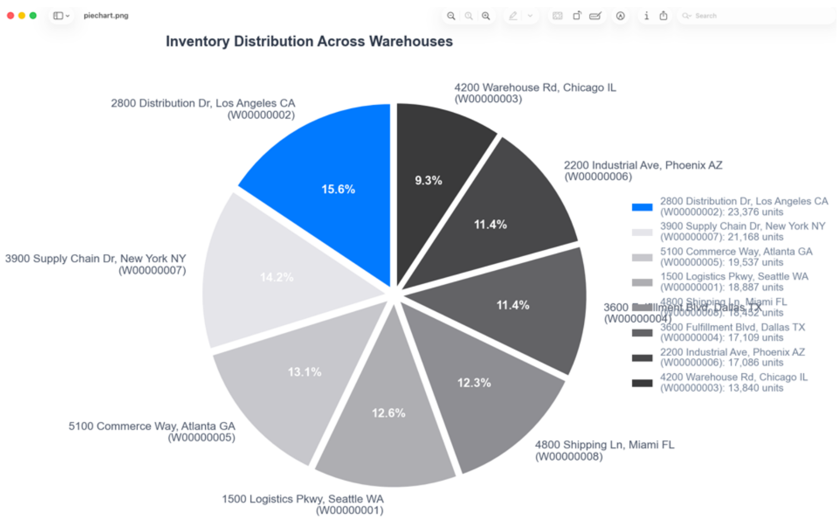
```
product_count : 10
avg_price : 301.28499999999997
manufacturer : "Nike Inc"

product_count : 10
avg_price : 298.459
manufacturer : "Mead Johnson"

product_count : 10
avg_price : 204.66500000000002
manufacturer : "Edifier"

product_count : 10
avg_price : 153.752
manufacturer : "Levi Strauss"

product_count : 10
avg_price : 130.502
manufacturer : "Under Armour"
```

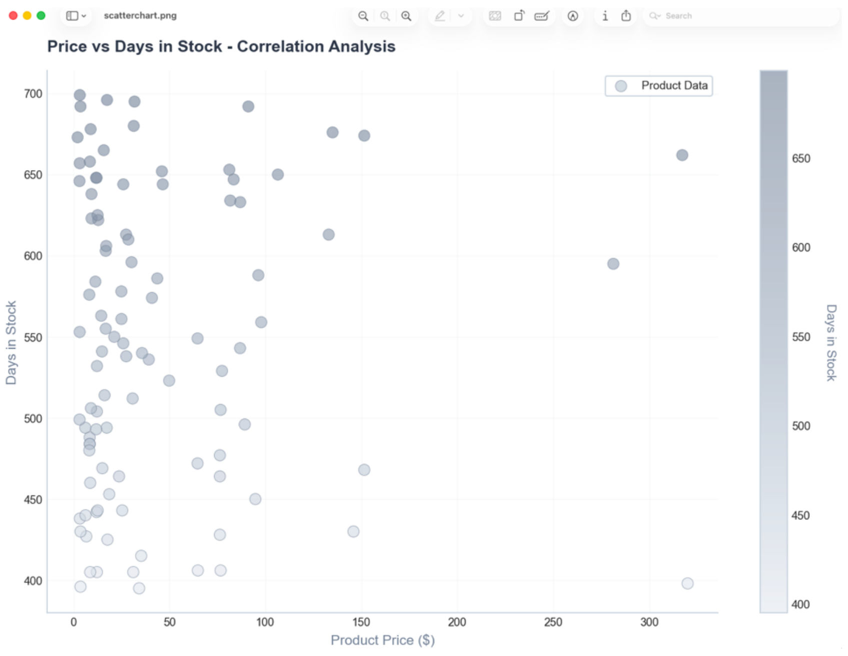
V. Database Access via Python

The database is accessed using Python, and the visualization of the analyzed data is shown below. The connection to MySQL is established through **mysql.connector**, after which SQL queries are executed using cursor.execute() and retrieved using fetchall(). The resulting dataset is then visualized using **Matplotlib**, where the system generates analytical charts. These plots are encoded into **Base64** and transmitted to the HTML frontend for in-browser rendering, while PNG copies of the figures are also saved locally for archival and evaluation purposes.


Database Analysis Visualization
Top 5 Products with Lowest Sell-Through Rates
Price vs Days in Stock Analysis
Warehouse Inventory Distribution
Price vs Days in Stock - Correlation Analysis
Days in Stock
Product Price ($)
Product Data
Inventory Distribution Across Warehouses





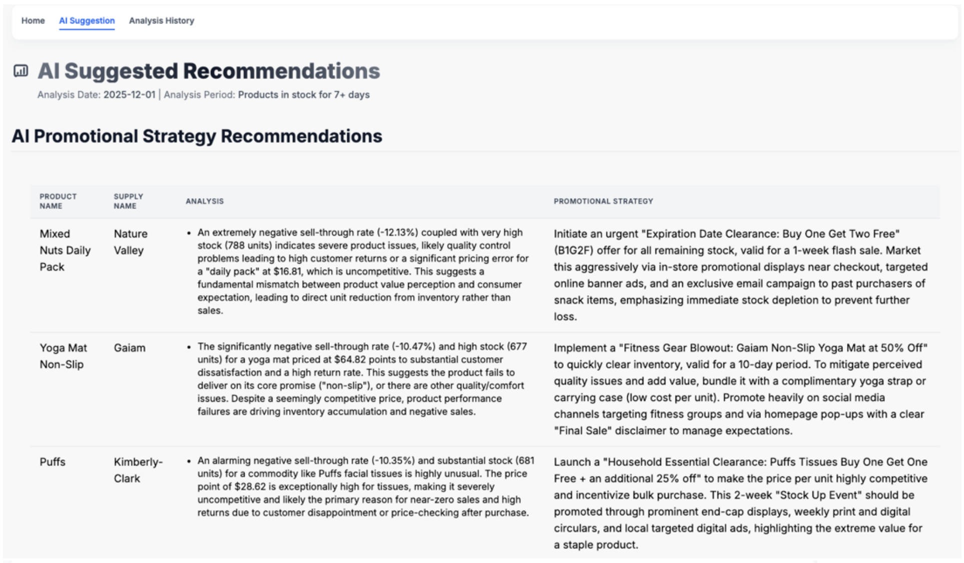


VI. Application Demonstration

The application consists of a full-stack system built with HTML/CSS/JavaScript for the frontend and Flask (Python) for the backend. The system retrieves product data from a MySQL database, identifies the 50 products with the highest unsold rate, selects the top 5, and sends these items to Gemini-2.5-Flash for automated analysis.

Gemini provides explanations for why these products are underperforming and generates actionable recommendations. Additionally, a machine learning demand forecasting module built with Random Forest regression provides real-time predictions of order quantities at different price points, testing multiple discount scenarios to quantify their impact on sales volume and revenue, thereby enabling data-driven pricing optimization for slow-moving inventory.

AI-generated insights are stored in MongoDB for historical tracking, and the frontend interface displays both the latest AI analysis and the historical records saved in MongoDB, while ML pricing predictions are generated on-demand for interactive decision support. The system includes an automated scheduler that triggers the analysis pipeline every 24 hours, ensuring merchants receive up-to-date recommendations.

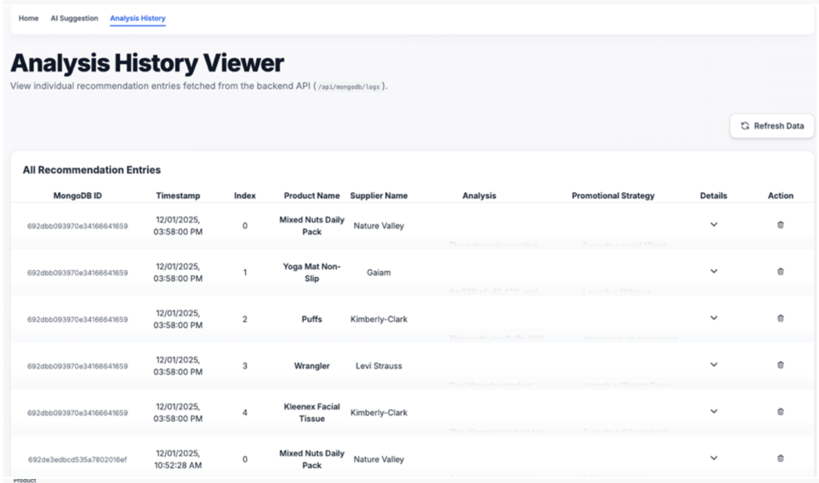


AI Suggested Recommendations

Analysis Date: 2025-12-01 | Analysis Period: Products in stock for 7+ days

AI Promotional Strategy Recommendations

PRODUCT NAME	SUPPLY NAME	ANALYSIS	PROMOTIONAL STRATEGY
Mixed Nuts Daily Pack	Nature Valley	• An extremely negative sell-through rate (-12.13%) coupled with very high stock (788 units) indicates severe product issues, likely quality control problems leading to high customer returns or a significant pricing error for a "daily pack" at $16.81, which is uncompetitive. This suggests a fundamental mismatch between product value perception and consumer expectation, leading to direct unit reduction from inventory rather than sales.	Initiate an urgent "Expiration Date Clearance: Buy One Get Two Free" (B1G2F) offer for all remaining stock, valid for a 1-week flash sale. Market this aggressively via in-store promotional displays near checkout, targeted online banner ads, and an exclusive email campaign to past purchasers of snack items, emphasizing immediate stock depletion to prevent further loss.
Yoga Mat Non-Slip	Gaiam	• The significantly negative sell-through rate (-10.47%) and high stock (677 units) for a yoga mat priced at $64.82 points to substantial customer dissatisfaction and a high return rate. This suggests the product fails to deliver on its core promise ("non-slip"), or there are other quality/comfort issues. Despite a seemingly competitive price, product performance failures are driving inventory accumulation and negative sales.	Implement a "Fitness Gear Blowout: Gaiam Non-Slip Yoga Mat at 50% Off" to quickly clear inventory, valid for a 10-day period. To mitigate perceived quality issues and add value, bundle it with a complimentary yoga strap or carrying case (low cost per unit). Promote heavily on social media channels targeting fitness groups and via homepage pop-ups with a clear "Final Sale" disclaimer to manage expectations.
Puffs	Kimberly-Clark	• An alarming negative sell-through rate (-10.35%) and substantial stock (681 units) for a commodity like Puffs facial tissues is highly unusual. The price point of $28.62 is exceptionally high for tissues, making it severely uncompetitive and likely the primary reason for near-zero sales and high returns due to customer disappointment or price-checking after purchase.	Launch a "Household Essential Clearance: Puffs Tissues Buy One Get One Free + an additional 25% off" to make the price per unit highly competitive and incentivize bulk purchase. This 2-week "Stock Up Event" should be promoted through prominent end-cap displays, weekly print and digital circulars, and local targeted digital ads, highlighting the extreme value for a staple product.

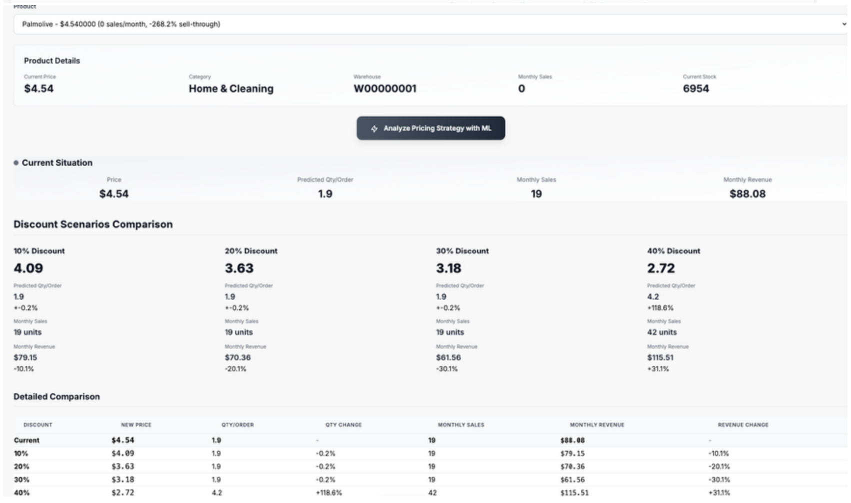


Analysis History Viewer

View individual recommendation entries fetched from the backend API (`/api/mongodb/logs`).

All Recommendation Entries

MongoDB ID	Timestamp	Index	Product Name	Supplier Name	Analysis	Promotional Strategy	Details	Action
692dbb093970e34166641659	12/01/2025, 03:58:00 PM	0	Mixed Nuts Daily Pack	Nature Valley				
692dbb093970e34166641659	12/01/2025, 03:58:00 PM	1	Yoga Mat Non-Slip	Gaiam				
692dbb093970e34166641659	12/01/2025, 03:58:00 PM	2	Puffs	Kimberly-Clark				
692dbb093970e34166641659	12/01/2025, 03:58:00 PM	3	Wrangler	Levi Strauss				
692dbb093970e34166641659	12/01/2025, 03:58:00 PM	4	Kleenex Facial Tissue	Kimberly-Clark				
692de3edbcd535a7802016ef	12/01/2025, 10:52:28 AM	0	Mixed Nuts Daily Pack	Nature Valley				



Product

Palmolive - $4.540000 (0 sales/month, -268.2% sell-through)

Product Details

Current Price	Category	Warehouse	Monthly Sales	Current Stock
$4.54	Home & Cleaning	W00000001	0	6954

● Current Situation

Price	Predicted Qty/Order	Monthly Sales	Monthly Revenue
$4.54	1.9	19	$88.08

Discount Scenarios Comparison

	10% Discount	20% Discount	30% Discount	40% Discount
	4.09	3.63	3.18	2.72
Predicted Qty/Order	1.9 +-0.2%	1.9 +-0.2%	1.9 +-0.2%	4.2 +118.6%
Monthly Sales	19 units	19 units	19 units	42 units
Monthly Revenue	$79.15 -10.1%	$70.36 -20.1%	$61.56 -30.1%	$115.51 +31.1%

Detailed Comparison

DISCOUNT	NEW PRICE	QTY/ORDER	QTY CHANGE	MONTHLY SALES	MONTHLY REVENUE	REVENUE CHANGE
Current	$4.54	1.9	-	19	$88.08	-
10%	$4.09	1.9	-0.2%	19	$79.15	-10.1%
20%	$3.63	1.9	-0.2%	19	$70.36	-20.1%
30%	$3.18	1.9	-0.2%	19	$61.56	-30.1%
40%	$2.72	4.2	+118.6%	42	$115.51	+31.1%

VII. Summary and Recommendation

The Warehouse Management System successfully integrates MySQL, MongoDB, Flask, machine learning, and Google's Gemini API to provide AI-powered promotional recommendations for slow-moving inventory. The system demonstrates advanced SQL query capabilities through three Python-generated visualizations (bar chart, scatter plot, pie chart) that directly connect to the database, meeting academic requirements while providing practical business value. A machine learning pricing optimization module provides quantitative demand forecasting to complement AI-generated promotional strategies.

The dual-database architecture combines MySQL's transactional integrity with MongoDB's flexibility for analytics storage, deployed on Render cloud platform with RESTful API architecture. ML predictions could be enhanced by tracking post-discount performance for continuous model refinement.

An area for improvement is implementing campaign effectiveness tracking. Currently, the system generates promotional recommendations but does not measure whether strategies were implemented or their sales impact, preventing ROI analysis and iterative AI model refinement. Additionally, automated product data validation through barcode scanning or external databases (GS1) would reduce manual entry errors and improve data quality consistency across the system.

A key limitation is the synchronous AI processing architecture, which introduces 10-15 second latency for initial requests. While caching mitigates this for subsequent queries, the approach may not scale efficiently for high-volume operations. Furthermore, the system analyzes warehouses independently without considering inter-warehouse inventory rebalancing, potentially missing cost-effective alternatives to promotional discounting.

Despite these limitations, the modular design provides a solid foundation for future enhancements and demonstrates successful integration of database management, AI AI and ML capabilities, and modern web technologies for intelligent inventory management.

VIII. Appendix

Source Code Repository

The complete source code on GitHub:
https://github.com/WAN519/Warehouse
The repository includes:
- Flask backend application (`app.py`)
- Database schema and SQL queries(`SalesAnalyzer.py`)
- AI integration modules (`PromotionAdvisor.py`)
- ML data extract and model training (`ML_extract_data.py`,`train_model.py`)
- ML-trained model (`demand_forecast_model.py`)
- Chart generation module (`ChartGenerator.py`)
- Frontend HTML/CSS/JavaScript files (`index.html`,`ai_suggestion.html`,`analysis_reports.html`, `ML_extract_data.py`)
- MongoDB integration (`mongoDB.py`)
- Timed trigger device(`Scheduler.py`)
- Requirements and deployment documentation(`requirements.txt`)

Deployment URL (Live Demo):
https://wan519.github.io/Warehouse/