Precious Metals

Milestone: Project Report

Group 6

Student1 Sanidhya Karnik

Student2 Digvijay Raut

617-407-1206 (Tel of Student 1)

857-492-3195 (Tel of Student 2)

karnik.san@northeastern.edu

raut.di@northeastern.edu

Percentage of Effort Contributed by Student1: 50%

Percentage of Effort Contributed by Student2: 50%

Signature of Student 1 : Sanidhya Karnik

Signature of Student 2 : Digvijay Raut

Submission Date : 12-10-2023

USE CASE STUDY REPORT

Group No.: Group 06

Student Names: Sanidhya Karnik and Digvijay Raut

Executive Summary:

The objective of this study was to design and implement a comprehensive database system to streamline and optimize the operations of the organization. This database system was envisioned to manage customer information, orders, transactions, agent relationships, distributor details, and metal-related data. Ensuring data integrity, security, and efficient retrieval were paramount in the requirements. For data modeling, the schema is incorporated with tables for Web Activity, Orders, Enums (acting as a superclass), Order Status and Payment Status (as subclasses), Customers, Transactions, Wallets, Agents, Distributors, Metals, and Metal Rates. Primary and foreign keys were strategically defined to establish robust connections between entities.

EER and UML were modelled and corresponding Relational model was built using the defined primary and foreign keys. The database was then implemented using a relational model i.e. MySQL, all the tables were created in it along with their relationships and constraints. We used MySQL for efficient data manipulation and retrieval. We used MongoDB to replicate the MySQL database into a NoSQL database and MongoShell was used to test the capabilities of the same.

The system demonstrated resilience in handling complex queries and managing our mid-scale datasets. We used Python to connect to the database and retrieve data directly from MySQL database using relevant libraries and performed basic analysis and data visualization.

I. Introduction

In the present era, there's a notable trend toward digital currency, especially among the newer generation. The digital market is on a serious upswing – it's growing rapidly. What's driving this momentum? Well, people are drawn to the idea that the digital market offers a safer and faster way to handle transactions. Considering these shifts, our study zeroes in on the implementation of a robust database system. The goal is to capitalize on the opportunities presented by the expanding digital market. This isn't just about following the trend; it's about strategically positioning our organization to ensure secure and efficient digital transactions, staying ahead in this dynamic landscape. But we can't ignore the base of economy - Gold. Gold and other precious metals are still valued more. As these metals cannot be replicated. They have been the most stable form of currency since world trading has stared. So, the business model combines both digital world and gold together.

To make such a platform we encountered significant challenges in handling diverse datasets tied to customer interactions, order processing, and financial transactions. As the dataset was getting complicated due to few repeated terms for status in different tables, we created an Enums table which stores all the statuses, column names and table names as a super class, which makes the database more organized.

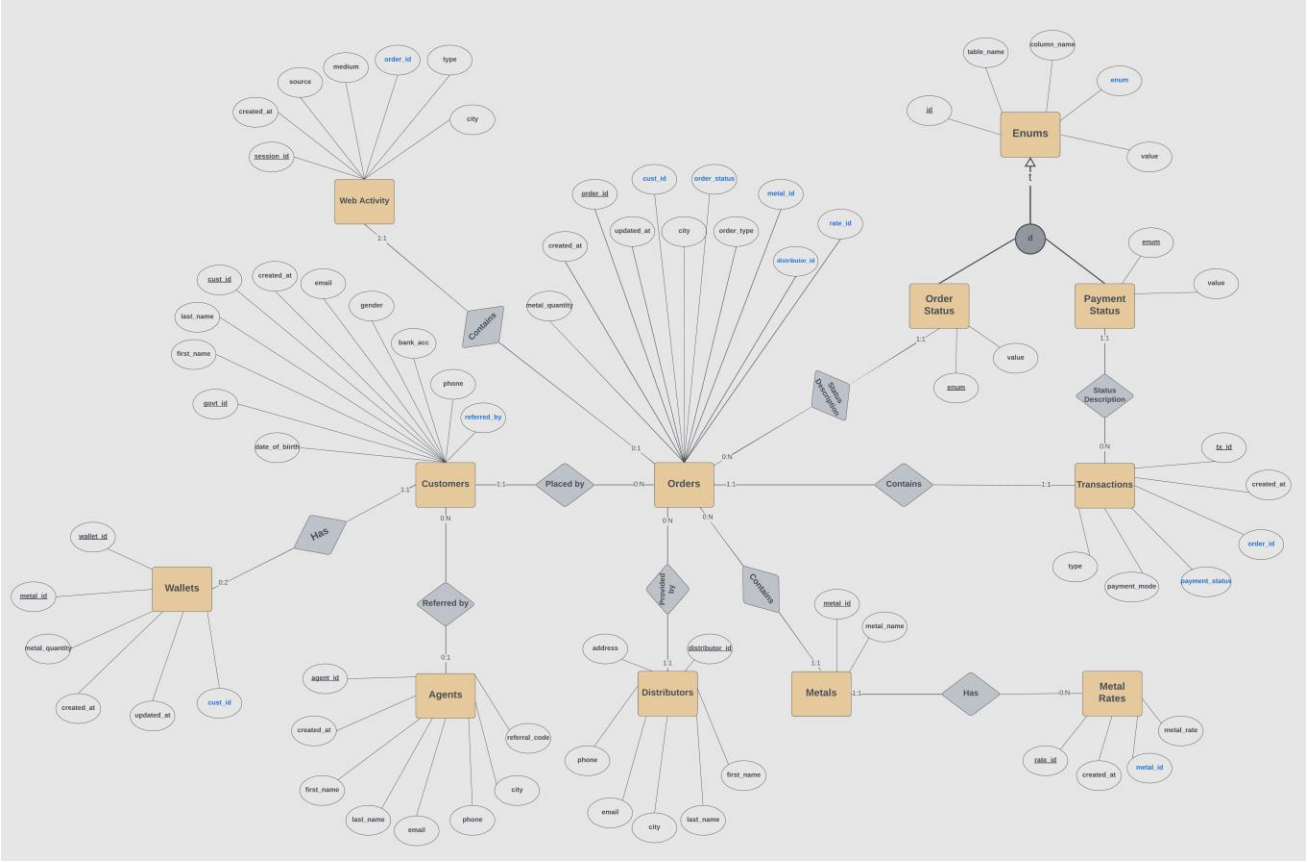
The primary goal of this study is to design and implement a robust database system that addresses the identified challenges. To create a centralized data repository that facilitates seamless information flow across various business processes. This envisioned system aims to improve data accuracy, reduce processing times, and provide a solid foundation for future scalability.

Requirements:

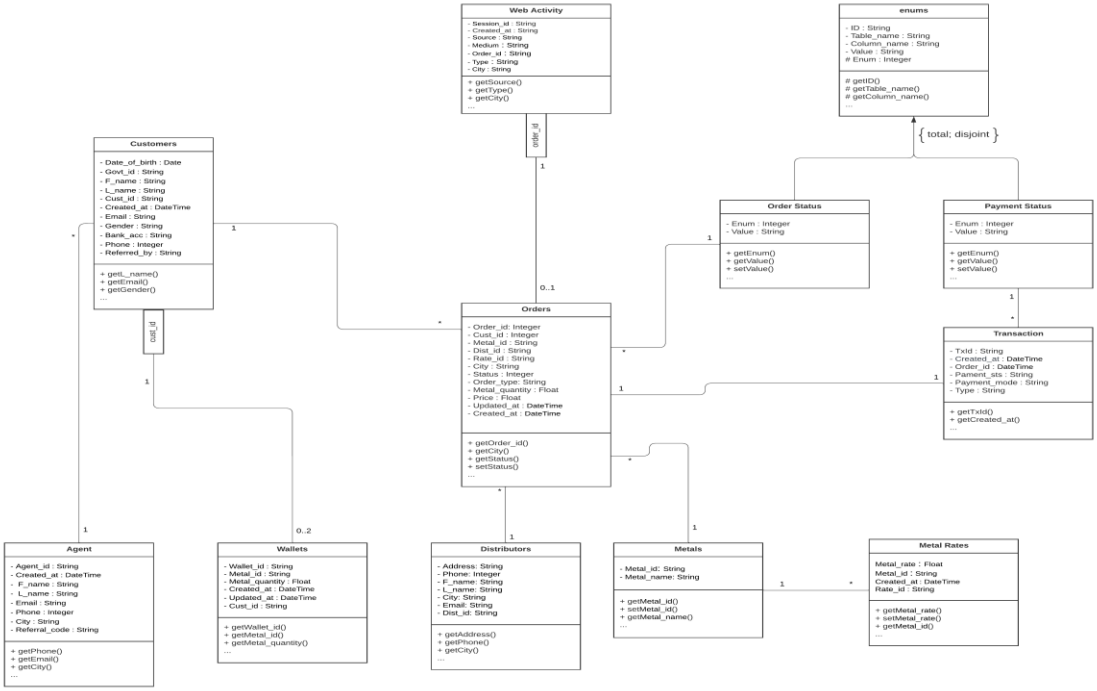
1. A user can hold from 0 to an infinite amount of Gold/Silver
2. A user can buy from 0 to $1000 worth of Gold/Silver in a single transaction
3. A user can have only 1 account on the website
4. A referral agent can refer from 0 to an infinite number of people
5. A distributor can provide from 0 to an infinite amount of Gold/Silver
6. An order can be fulfilled by only 1 distributor
7. An order can be mapped to only 1 referral agent (if any)
8. Amount of allocated Gold/Silver cannot be greater than the amount available in the inventory

II. Conceptual Data Modeling

EER:



UML:



III. Mapping Conceptual Model to Relational Model

1. **Web Activity** (session_id, created_at, source, medium, **order_id**, type, city)
    - session_id is the primary key
    - order_id is the foreign key referring to order_id from the relation - Orders and can be null
2. **Orders** (order_id, created_at, updated_at, order_type, **metal_id**, metal_quantity, price, **rate_id**, **distributor_id**, **order_status**, **cust_id**, city)
    - order_id is the primary key
    - metal_id is a foreign key referring to metal_id in the relation - Metals and cannot be null
    - rate_id is a foreign key referring to rate_id in the relation - Metal Rates and cannot be null
    - distributor_id is a foreign key referring to distributor_id in the relation - Distributors and cannot be null
    - order_status is a foreign key referring to enum in the relation - Order Status and cannot be null
    - cust_id is a foreign key referring to cust_id in the relation - Customers and cannot be null
3. **Enums** (id, table_name, column_name, **enum**, value)
    - This relation is a superclass
    - id is the primary key
    - enum is the foreign key referring to enum in the subclasses - Order Status and Payment Status and cannot be null
4. **Order Status** (enum, value)
    - This relation is a subclass under the super class - Enums
    - enum is the primary key
5. **Payment Status** (enum, value)
    - This relation is a subclass under the super class - Enums
    - enum is the primary key
6. **Customers** (cust_id, first_name, last_name, created_at, phone, email, date_of_birth, gender, govt_id, bank_acc, **referred_by**)
    - cust_id is the primary key
    - referred_by is the foreign key referring to agent_id in the relation - Agents and can be null
7. **Transactions** (tx_id, created_at, **order_id**, **payment_status**, payment_mode, type
    - tx_id is the primary key
    - order_id is a foreign key referring to order_id in the relation Orders and cannot be null
    - payment_status is a foreign key referring to enum in the relation - Payment Status and cannot be null
8. **Wallets** (wallet_id, metal_id, metal_quantity, created_at, updated_at, **cust_id**)
    - wallet_id is the primary key
    - cust_id is the foreign key referring to cust_id in the relation - Customers

9. **Agents** (agent_id, created_at, first_name, last_name, email, phone, city, referral_code)
    - agent_id is the primary key
10. **Distributors** (distributor_id, first_name, last_name, phone, email, city, address)
    - distributor_id is the primary key
11. **Metals** (metal_id, metal_name)
    - metal_id is the primary key
12. **Metal Rates** (rate_id, created_at, **metal_id**, metal_rate)
    - rate_id is the primary key
    - metal_id is the foreign key referring to metal_id in the relation - Metals

IV. Implementation of Relation Model via MySQL and NoSQL

MySQL Implementation:

Random data was generated for all the tables using Mockaroo. Python was then used to ingest the data into MySQL Database post its creation. Following are few queries that have been run along with their outputs:

Query 1: Fetch the top 5 successful BUY orders based on price

```
select order_id, date(created_at), metal_id, metal_quantity, price
from orders
where order_status = 2 and order_type = 'buy'
order by price desc
limit 5
```

order_id	date(created_at)	metal_id	metal_quantity	price
0336edd6-d392-4825-b1b9-8d1fe6cd470a	2023-04-04	e822ecb5-1690-479f-b353-91e97fbf68d1	4.941	349
2c4c82cf-4f73-4ca5-a3c2-e173bea7b69e	2023-04-06	e822ecb5-1690-479f-b353-91e97fbf68d1	4.848	342
7c837eed-8d0f-48ee-9817-61d0ba0dc9f6	2023-04-15	e822ecb5-1690-479f-b353-91e97fbf68d1	4.655	324
ae740326-4dad-42de-8cac-bd15dcef2e35	2023-04-04	e822ecb5-1690-479f-b353-91e97fbf68d1	4.544	321
bbeaffcd-41c9-43b4-80b7-07b281ea02eb	2023-04-24	e822ecb5-1690-479f-b353-91e97fbf68d1	4.443	311

Query 2: Fetch total quantity bought and sold for each metal successfully

```
select m.metal_name, o.order_type, sum(o.metal_quantity) as total_quantity
from orders o, metals m
where o.metal_id = m.metal_id
and o.order_status = 2
group by 1,2
order by 2,1
```

metal_name	order_type	total_quantity
Gold	buy	138.919
Silver	buy	803.000
Gold	sell	20.017
Silver	sell	78.000

Query 3: Find customers who have spent at least $100 on successful buy transactions

```
select c.first_name, c.last_name,
c.phone, c.email,
m.metal_name, sum(o.price) as total_spent
from orders o
left outer join customers c
on c.cust_id = o.cust_id
left outer join metals m
on m.metal_id = o.metal_id
```

first_name	last_name	phone	email	metal_name	total_spent
Boote	Mudge	256-512-6752	bmudge14@jigsy.com	Gold	655
Marvin	Issakov	134-136-1311	missakov22@mayoclinic.com	Gold	521
Doyle	Svanetti	996-221-4972	dsvanetti1q@123-reg.co.uk	Gold	509
Prudence	Kilgrew	326-368-3341	pkilgrew13@discovery.com	Gold	471
Calley	Curnnok	643-201-7867	ccurnnokd@google.ru	Gold	469
Caroline	Mercey	645-383-2061	cmercey17@a8.net	Gold	446
Allie	Troy	478-994-3898	atroy3@xrea.com	Gold	431
Lucais	Beccera	751-458-7848	lbeccera1s@macromedia.com	Gold	408
Alison	Boakes	393-271-9510	aboakes1d@ameblo.jp	Gold	349

```
where o.order_status = 2
group by 1,2,3,4,5
having sum(o.price) >= 100
order by 6 desc
```

Query 4: Find agents who have referred at least 3 customers

```
select a.first_name, a.last_name, a.phone, a.email, a.city
from agents a
where a.referral_code in (select referred_by
                          from customers
                          group by 1
                          having count(*) >= 3)
```

first_name	last_name	phone	email	city
Salomo	Craddock	505-224-8947	scraddock6@sciencedaily.com	Bradenton
Stillman	Labusch	971-552-8651	slabuschb@unicef.org	Conroe
Milton	Hasely	877-934-2128	mhasely4@nydailynews.com	Milwaukee
Ody	Tembey	308-332-2333	otembey7@plala.or.jp	Waterbury
Stacie	Revie	186-332-9687	srevie2@tiny.cc	Macon
Fran	Elms	570-782-7621	felms1@patch.com	Orlando
Camile	Llorens	823-916-0403	cllorens3@taobao.com	Greeley
Vassily	Deinhardt	962-829-5272	vdeinhardt5@newyorker.com	Simi Valley
Nat	Tague	804-300-8964	ntagued@techcrunch.com	Nashville

Query 5: Find customers who have made at least 4 transactions

```
select distinct c.first_name, c.last_name
from transactions t, orders o, customers c
where t.order_id = o.order_id and c.cust_id = o.cust_id
and 4 <= (select count(*)
          from transactions t2, orders o2, customers c2
          where t2.order_id = o2.order_id
          and o2.cust_id = c2.cust_id
          and c2.cust_id = c.cust_id)
```

first_name	last_name
Inesita	Cassar
Allie	Troy
Stanfield	Banker
Harman	Turbern
Bel	Olenichev
Garnette	Minocchi
Brigg	Semonin
Byron	Stolberger
Trip	Blackie

Query 6: Fetch largest successful order for each metal based on quantity

```
select m.metal_name, o.metal_quantity
from orders o, metals m
where o.metal_id = m.metal_id
and o.order_status = 2
and o.metal_quantity >= ALL (select o2.metal_quantity
                             from orders o2
                             where o2.order_status = 2
                             and o2.metal_id = o.metal_id)
order by 1
```

metal_name	metal_quantity
Gold	4.941
Silver	97.000

Query 7: Fetch all BUY orders which were placed in Miami or Los Angeles or have a price of more than $300

```
select o.order_id, o.created_at, m.metal_name, o.metal_quantity, o.price, o.city
from orders o, metals m
where o.metal_id = m.metal_id
```

```
and o.order_status = 2 and o.order_type = 'buy'
and o.price > 300
union
select o.order_id, o.created_at, m.metal_name, o.metal_quantity, o.price, o.city
from orders o, metals m
where o.metal_id = m.metal_id
and o.order_status = 2 and o.order_type = 'buy'
and o.city in ('Los Angeles','Miami')
```

order_id	created_at	metal_name	metal_quantity	price	city
bbeaffcd-41c9-43b4-80b7-07b281ea02eb	2023-04-24 00:00:00	Gold	4.443	311	Laurel
ae740326-4dad-42de-8cac-bd15dcef2e35	2023-04-04 00:00:00	Gold	4.544	321	Aurora
7c837eed-8d0f-48ee-9817-61d0ba0dc9f6	2023-04-15 00:00:00	Gold	4.655	324	Ocala
7ac8e5a1-378e-4eae-9e91-d2c28fb344b6	2023-04-23 00:00:00	Gold	4.396	307	Tulsa
2c4c82cf-4f73-4ca5-a3c2-e173bea7b69e	2023-04-06 00:00:00	Gold	4.848	342	Richmond
1654ff5a-1de1-4b2c-bf6e-6261dc6ac889	2023-04-25 00:00:00	Gold	4.403	309	Charleston
0336edd6-d392-4825-b1b9-8d1fe6cd470a	2023-04-04 00:00:00	Gold	4.941	349	Birmingham
243507a4-33fb-481f-a0db-c344ab333584	2023-04-13 00:00:00	Gold	2.610	183	Miami
14d112d2-4070-44f3-b8c9-ba336091cd9f	2023-04-25 00:00:00	Gold	3.322	233	Los Angeles

Query 8: Fetch day-on-day total buy orders, successful buy orders, sell orders and successful sell orders

```
select distinct date(w.created_at) as date,
(select count(*)
        from orders o
   where date(o.created_at) = date(w.created_at)
   and o.order_type = 'buy') as buy_orders,
(select count(*)
        from orders o2
   where date(o2.created_at) = date(w.created_at)
   and o2.order_type = 'buy'
   and o2.order_status = 2) as successful_buy_orders,
(select count(*)
        from orders o
   where date(o.created_at) = date(w.created_at)
   and o.order_type = 'sell') as sell_orders,
(select count(*)
        from orders o2
   where date(o2.created_at) = date(w.created_at)
   and o2.order_type = 'sell'
   and o2.order_status = 2) as successful_sell_orders
from webactivity w
order by 1
```

date	buy_orders	successful_buy_orders	sell_orders	successful_sell_orders
2023-04-01	13	6	1	0
2023-04-02	1	0	1	1
2023-04-03	7	0	4	1
2023-04-04	6	4	3	2
2023-04-05	7	2	3	1
2023-04-06	13	6	0	0
2023-04-07	4	1	0	0
2023-04-08	6	2	3	0
2023-04-09	5	1	1	0

NoSQL Implementation:

Collections were created for all the MySQL tables on MongoDB. Following are few queries that have been run on MongoShell along with their outputs:

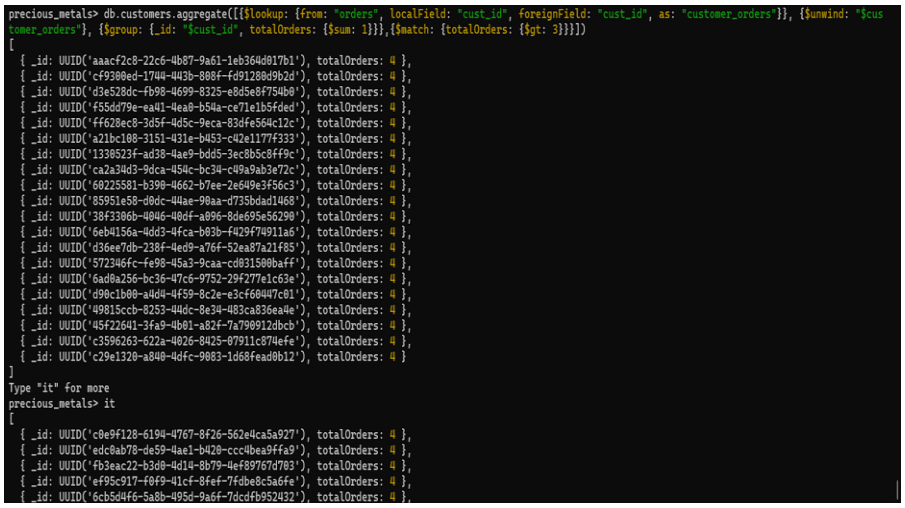
Query 1: Fetch top 2 successful buy orders based on price

```
db.orders.find({order_type: 'buy',
```

order_status: 2
}
).sort({price: -1}).limit(2)

Query 2: Fetch total orders placed by each customer who has ordered more than 3 times

```
db.customers.aggregate([
  {
    $lookup: {
      from: "orders",
      localField: "cust_id",
      foreignField: "cust_id",
      as: "customer_orders"
    }
  },
  {
    $unwind: "$customer_orders"
  },
  {
    $group: {
      _id: "$cust_id",
      totalOrders: { $sum: 1 }
    }
  },
  {
    $match: {
      totalOrders: { $gt: 3 }
    }
  }
])
```



Query 3: Metal quantities grouped by customer and metal_id

```
db.wallets.aggregate([{$group:{_id:{cust_id: "$cust_id", metal_id: "$metal_id"},
```

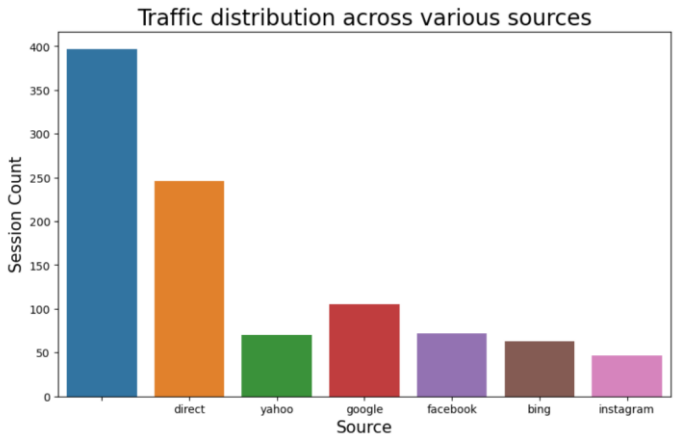
total_quantity: {$sum: "$metal_quantity"}}}])

```
precious_metals> db.wallets.aggregate([{$group: {_id: {cust_id: "$cust_id", metal_id: "$metal_id"}, total_quantity: {$sum: "$metal_quantity"}}}])
[
  {
    _id: {
      cust_id: UUID('c0e9f128-6194-4767-8f26-562e4ca5a927'),
      metal_id: UUID('78abe9db-303e-4878-9d5c-857bc5fb318f')
    },
    total_quantity: 36
  },
  {
    _id: {
      cust_id: UUID('86ad30b4-ab67-49a7-9ef6-68c508d6bd2e'),
      metal_id: UUID('e822ecb5-1690-479f-b353-91e97fbf68d1')
    },
    total_quantity: 4.443
  },
  {
    _id: {
      cust_id: UUID('9d96b639-45a1-444b-91b0-181cadba2f72'),
      metal_id: UUID('e822ecb5-1690-479f-b353-91e97fbf68d1')
    },
    total_quantity: 6.688
  },
  {
    _id: {
      cust_id: UUID('b4beda75-3a8c-4012-8718-7d6ae5addde0'),
      metal_id: UUID('e822ecb5-1690-479f-b353-91e97fbf68d1')
    },
    total_quantity: 9.346
  },
```

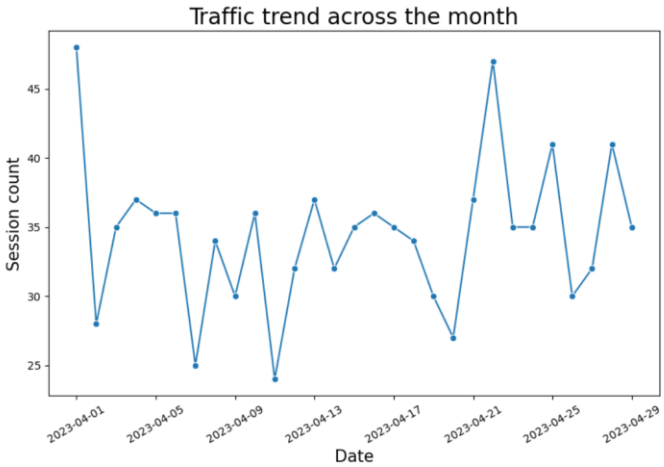
V. Database Access via R or Python

Python is used to access the MySQL database using the libraries mysql.connector and pymysql. The connection is made by creating an engine and then using engine.connect(). The read_sql function from Pandas library is then used to retrieve the data generated by the query into a dataframe. Matplotlib and Seaborn libraries are used to create visualizations for the retrieved data as shown below:

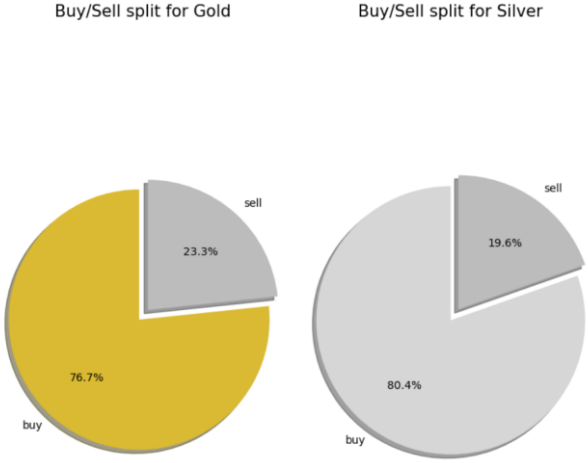
Graph 1



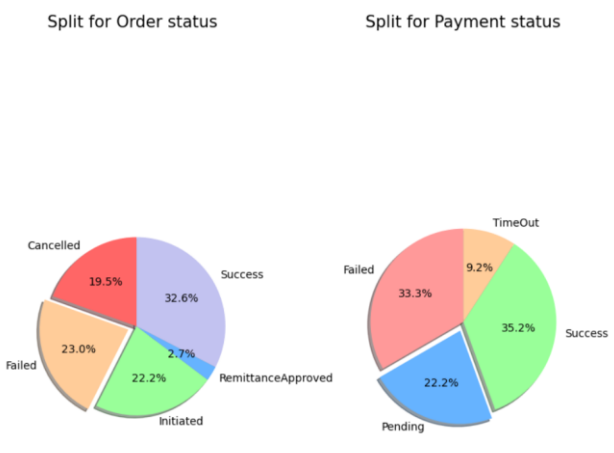
Graph 2



Graph 3



Graph 3



VII. Summary and recommendation

The implementation of the database system has played a pivotal role in transforming the management of precious metals data. The transition to a centralized database platform has streamlined processes related to precious metal transactions, order processing, and customer interactions. Significant achievements include heightened data accuracy, accelerated processing times, and improved adaptability to the dynamic landscape of the precious metals market.

Advantages:

1. **Centralized Precious Metals Management:** The database platform effectively consolidates precious metals data, eliminating silos and offering a centralized hub for efficient management.
2. **Enhanced Data Accuracy and Consistency:** Improved validation mechanisms have led to increased accuracy, reducing inconsistencies and errors in precious metal-related information.
3. **Scalability for the Precious Metals Market:** The designed architecture supports scalability, allowing for seamless adaptation to the evolving demands of the precious metals market.

Shortcomings:

1. **Learning Curve:** The introduction of a new database system may pose a learning curve for users, necessitating comprehensive training and support.
2. **Implementation Costs:** While the long-term benefits are evident, there are initial implementation costs associated with adopting the new database system.

Recommendations:

1. **Cost-Benefit Analysis:** Conduct periodic cost-benefit analyses to evaluate the ongoing relevance and effectiveness of the database system.
2. **Feedback Mechanism for Stakeholders:** Establish a feedback loop involving stakeholders in the precious metals industry to gather insights, enabling continuous improvement and refinement of the database.