Precious Metals

Milestone: Implementation in MySQL

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 : 11-12-2023

SQL Queries for Database Tables Creation :

```
CREATE TABLE WebActivity (
    session_id VARCHAR(36) PRIMARY KEY,
    created_at TIMESTAMP, -- Use the appropriate data type for timestamps
    source VARCHAR(20), -- Adjust the length as needed
    medium VARCHAR(20), -- Adjust the length as needed
    order_id VARCHAR(36), -- This is a foreign key
    type VARCHAR(20), -- Adjust the length as needed
    city VARCHAR(20) -- Adjust the length as needed
);

CREATE TABLE Orders (
    order_id VARCHAR(36) PRIMARY KEY,
    created_at TIMESTAMP,
    updated_at TIMESTAMP,
    order_type VARCHAR(4), -- Adjust the length as needed
    metal_id VARCHAR(36) NOT NULL,
    metal_quantity DECIMAL(10, 3), -- Adjust the precision and scale as needed
    rate_id VARCHAR(36) NOT NULL,
    price INT,
    distributor_id VARCHAR(36) NOT NULL,
    order_status INT NOT NULL, -- Adjust the length as needed
    cust_id VARCHAR(36) NOT NULL,
    city VARCHAR(255) -- Adjust the length as needed
);

CREATE TABLE Enums (
    id VARCHAR(36) PRIMARY KEY,
    table_name VARCHAR(20) NOT NULL, -- Name of the subclass table
    column_name VARCHAR(20) NOT NULL, -- Name of the column in the subclass table
    enum INT NOT NULL, -- This is a foreign key referring to the subclasses' enum values
    value VARCHAR(20) NOT NULL -- The actual enum value
);

CREATE TABLE OrderStatus (
    enum INT PRIMARY KEY,
    value VARCHAR(20) NOT NULL
);

CREATE TABLE PaymentStatus (
    enum INT PRIMARY KEY,
    value VARCHAR(20) NOT NULL
);

CREATE TABLE Customers (
    cust_id VARCHAR(36) PRIMARY KEY,
    first_name VARCHAR(20) NOT NULL, -- Adjust the length as needed
    last_name VARCHAR(20) NOT NULL, -- Adjust the length as needed
    created_at TIMESTAMP, -- Use the appropriate data type for timestamps
    phone VARCHAR(20), -- Adjust the length as needed
```

```
    email VARCHAR(50), -- Adjust the length as needed
    date_of_birth DATE, -- Use the appropriate data type for dates
    gender CHAR(1), -- Assuming gender is a single character (e.g., 'M' or 'F')
    govt_id VARCHAR(36), -- Adjust the length as needed
    bank_acc VARCHAR(36), -- Adjust the length as needed
    referred_by VARCHAR(36) -- This is a foreign key
);

CREATE TABLE Transactions (
    tx_id VARCHAR(36) PRIMARY KEY,
    created_at TIMESTAMP,
    order_id VARCHAR(36) NOT NULL, -- This is a foreign key
    payment_status INT NOT NULL, -- This is a foreign key
    payment_mode VARCHAR(20), -- Adjust the length as needed
    type VARCHAR(20) -- Adjust the length as needed
);

CREATE TABLE Wallets (
    wallet_id VARCHAR(36) PRIMARY KEY,
    metal_id VARCHAR(36), -- Adjust data type as needed
    metal_quantity DECIMAL(10, 3), -- Adjust precision and scale as needed
    created_at TIMESTAMP,
    updated_at TIMESTAMP,
    cust_id VARCHAR(36) NOT NULL -- This is a foreign key
);

CREATE TABLE Agents (
    agent_id VARCHAR(36) PRIMARY KEY,
    created_at TIMESTAMP,
    first_name VARCHAR(20), -- Adjust the length as needed
    last_name VARCHAR(20), -- Adjust the length as needed
    email VARCHAR(50), -- Adjust the length as needed
    phone VARCHAR(20), -- Adjust the length as needed
    city VARCHAR(20), -- Adjust the length as needed
    referral_code VARCHAR(36) -- Adjust the length as needed
);

CREATE TABLE Distributors (
    distributor_id VARCHAR(36) PRIMARY KEY,
    first_name VARCHAR(20), -- Adjust the length as needed
    last_name VARCHAR(20), -- Adjust the length as needed
    phone VARCHAR(20), -- Adjust the length as needed
    email VARCHAR(50), -- Adjust the length as needed
    city VARCHAR(20), -- Adjust the length as needed
    address VARCHAR(255) -- Adjust the length as needed
);

CREATE TABLE Metals (
    metal_id VARCHAR(36) PRIMARY KEY,
    metal_name VARCHAR(20) NOT NULL -- Adjust the length as needed
);
```

```
CREATE TABLE MetalRates (
   rate_id VARCHAR(36) PRIMARY KEY,
   created_at TIMESTAMP,
   metal_id VARCHAR(36) NOT NULL, -- This is a foreign key
   metal_rate DECIMAL(10, 3) -- Adjust precision and scale as needed
);


ALTER TABLE WebActivity
ADD CONSTRAINT FK_WebActivity_Orders FOREIGN KEY (order_id) REFERENCES Orders
(order_id);


ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Metal FOREIGN KEY (metal_id) REFERENCES
Metals(metal_id);

ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Rate FOREIGN KEY (rate_id) REFERENCES
MetalRates(rate_id);

ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Distributor FOREIGN KEY (distributor_id) REFERENCES
Distributors(distributor_id);

ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Customer_id FOREIGN KEY (cust_id) REFERENCES
Customers(cust_id);

ALTER TABLE Orders
ADD CONSTRAINT FK_Orders_Customer_sts FOREIGN KEY (order_status) REFERENCES
OrderStatus(enum);

ALTER TABLE Customers
ADD CONSTRAINT FK_Customers_Agents FOREIGN KEY (referred_by) REFERENCES
Agents(referral_code);

ALTER TABLE Transactions
ADD CONSTRAINT FK_Transactions_Orders FOREIGN KEY (order_id) REFERENCES
Orders(order_id);

ALTER TABLE Transactions
ADD CONSTRAINT FK_Transactions_PaymentStatus FOREIGN KEY (payment_status)
REFERENCES PaymentStatus(enum);

ALTER TABLE Wallets
ADD CONSTRAINT FK_Wallets_Customers FOREIGN KEY (cust_id) REFERENCES
Customers(cust_id);

ALTER TABLE MetalRates
```

ADD CONSTRAINT FK_MetalRates_Metals FOREIGN KEY (metal_id) REFERENCES Metals(metal_id);

Data Ingestion Code (Python) :

```
# -*- coding: utf-8 -*-
"""
@author: karnik

"""


import mysql.connector as mysql
import pymysql
from sqlalchemy import create_engine
import pandas as pd
import datetime
import pygsheets
import numpy
from sqlalchemy import create_engine


gc = pygsheets.authorize(service_file = r'C:\Users\karni\OneDrive\Desktop\Niwish files\niwish-analytics-a5704af3efcc.json')


sh=gc.open_by_url('https://docs.google.com/spreadsheets/d/1xgCb6QWl3LgSlX4RbwQpn8rpbSVvj83AzaS_25t-24Y/edit#gid=0')


# Importing Web Activity Table
wks1 = sh.worksheet('title','Web Activity')
web_activity = wks1.get_as_df(start='A1', end='G1001')
web_activity['created_at'] = pd.to_datetime(web_activity['created_at'])


# Importing Orders Table
wks2 = sh.worksheet('title','Orders')
orders = wks2.get_as_df(start='A1', end='L262')
orders['created_at'] = pd.to_datetime(orders['created_at'])
orders['updated_at'] = pd.to_datetime(orders['updated_at'])


# Importing Transactions Table
wks3 = sh.worksheet('title','Transactions')
transactions = wks3.get_as_df(start='A1', end='F262')
transactions['created_at'] = pd.to_datetime(transactions['created_at'])


# Importing Wallets Table
```

```
wks4 = sh.worksheet('title','Wallets')
wallets = wks4.get_as_df(start='A1', end='F62')
wallets['created_at'] = pd.to_datetime(wallets['created_at'])
wallets['updated_at'] = pd.to_datetime(wallets['updated_at'])


# Importing Customers Table
wks5 = sh.worksheet('title','Customers')
customers = wks5.get_as_df(start='A1', end='K76')
customers['created_at'] = pd.to_datetime(customers['created_at'])
customers['date_of_birth'] = pd.to_datetime(customers['date_of_birth'])


# Importing Agents Table
wks6 = sh.worksheet('title','Agents')
agents = wks6.get_as_df(start='A1', end='H16')
agents['created_at'] = pd.to_datetime(agents['created_at'])


# Importing Distributors Table
wks7 = sh.worksheet('title','Distributors')
distributors = wks7.get_as_df(start='A1', end='G21')


# Importing Metals Table
wks8 = sh.worksheet('title','Metals')
metals = wks8.get_as_df(start='A1', end='B3')


# Importing Metal Rates Table
wks9 = sh.worksheet('title','Metal Rates')
metal_rates = wks9.get_as_df(start='A1', end='D61')
metal_rates['created_at'] = pd.to_datetime(metal_rates['created_at'])


# Importing Enums Table
wks10 = sh.worksheet('title','Enums')
enums = wks10.get_as_df(start='A1', end='E10')


# Importing Order Status Table
wks11 = sh.worksheet('title','Order Status')
order_status = wks11.get_as_df(start='A1', end='B6')


# Importing Payment Status Table
wks12 = sh.worksheet('title','Payment Status')
payment_status = wks12.get_as_df(start='A1', end='B5')


# Connecting to the database
```

```
engine =
create_engine("mysql+pymysql://{user}:{pw}@127.0.0.1/{db}".format(user="sk",pw="1Q2w3E4r",db="preciousmetals"))
conn = engine.connect()
print('Database connected')


# Ingesting Metals data
metals.to_sql('metals', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into metals table")


# Ingesting Metal Rates data
metal_rates.to_sql('metalrates', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into metalrates table")


# Ingesting Order Status data
order_status.to_sql('orderstatus', con = conn, if_exists = 'append', chunksize = 2000, index =
False)
print("Data ingested into orderstatus table")


# Ingesting Payment Status data
payment_status.to_sql('paymentstatus', con = conn, if_exists = 'append', chunksize = 2000, index
= False)
print("Data ingested into paymentstatus table")


# Ingesting Enums data
enums.to_sql('enums', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into enums table")


# Ingesting Agents data
agents.to_sql('agents', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into agents table")


# Ingesting Distributors data
distributors.to_sql('distributors', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into distributors table")


# Ingesting Customers data
customers.to_sql('customers', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into customers table")


# Ingesting Wallets data
wallets.to_sql('wallets', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into wallets table")
```

```
# Ingesting Orders data
orders.to_sql('orders', con = conn, if_exists = 'append', chunksize = 2000, index = False)
print("Data ingested into orders table")


# Ingesting Transactions data
transactions.to_sql('transactions', con = conn, if_exists = 'append', chunksize = 2000, index =
False)
print("Data ingested into transactions table")


# Ingesting Web Activity data
web_activity.to_sql('webactivity', con = conn, if_exists = 'append', chunksize = 2000, index =
False)
print("Data ingested into webactivity table")


conn.close()
```

Tables Description :

Database	Entity Type (Table Name)	#Attribute Types (#Columns)	#Entities (#Rows)
preciousmetals	agents	8	15
preciousmetals	customers	11	75
preciousmetals	distributors	7	20
preciousmetals	enums	5	9
preciousmetals	metalrates	4	60
preciousmetals	metals	2	2
preciousmetals	orders	12	261
preciousmetals	orderstatus	2	5
preciousmetals	paymentstatus	2	4
preciousmetals	transactions	6	261
preciousmetals	wallets	6	61
preciousmetals	webactivity	7	1000

Example Queries :

1. Traffic count grouped by source and medium :

```
select source, medium, count(*) as session_count
from webactivity
group by 1,2
```

2. Traffic count grouped by date :

```
select date(created_at) as created_at, count(*) as session_count
from webactivity
group by 1
order by 1
```

3. Traffic count grouped by city :

```
select city, count(*) as session_count
from webactivity
group by 1
order by 1
```

4. Count of Buy and Sell orders grouped by metals :

```
select m.metal_name, o.order_type, count(*) as order_count
from orders o, metals m
where o.metal_id = m.metal_id
group by 1,2
order by 1,2
```

5. Count of Orders grouped by order_status and payment_status :

```
select os.value as order_status, ps.value as payment_status, count(*) as order_count
from orders o, transactions t, orderstatus os, paymentstatus ps
where o.order_id = t.order_id
and os.enum = o.order_status
and ps.enum = t.payment_status
group by 1,2
```

6. Customer with largest metal quantity :

```
select cust_id, metal_quantity
from Orders
where metal_quantity >= All (
        select metal_quantity
        from Orders
        )
```

7. Identify Agents and Their Referred Customers :

```
select a.agent_id, a.first_name, a.last_name, COUNT(c.cust_id) as
referred_customers
from Agents as a
left join Customers as c on a.referral_code = c.referred_by
group by a.agent_id
```

8. Total price of successful orders :

```
select SUM(price) AS total_completed_orders_price
from Orders
where order_status = ANY (
        select enum
        from OrderStatus
        where value = 'Success'
        )
```

9. Retrieve Agents Who Have Not Referred Any Customers :

```
select a.agent_id, a.first_name, a.last_name
from Agents as a
left join Customers ON a.referral_code = Customers.referred_by
where Customers.cust_id IS NULL
```

10. Find the Total Metal Quantity Ordered by Each Customer :

```
select cust_id, SUM(metal_quantity) as total_ordered_quantity
from Orders
group by cust_id
```