Term: Fall 2023 **Subject:** Computer Science & Engineering (CSE) **Number:** 512

Course Title: **D**istributed **D**atabase **S**ystems (CSE 512)

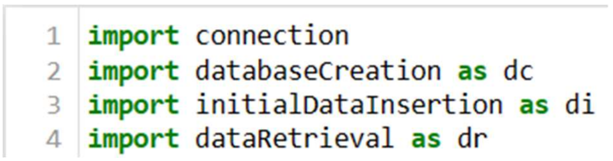
Team: PVS

Part 1: Design and Implementation of a Distributed Database System

Problem Statement: Design and implement a distributed database system for a topic of your choice, capable of handling real-time data updates and queries while ensuring data consistency and availability.

Tasks Completed: Schema design, Table creation, Data distribution, Data Insertion, Data Retrieval

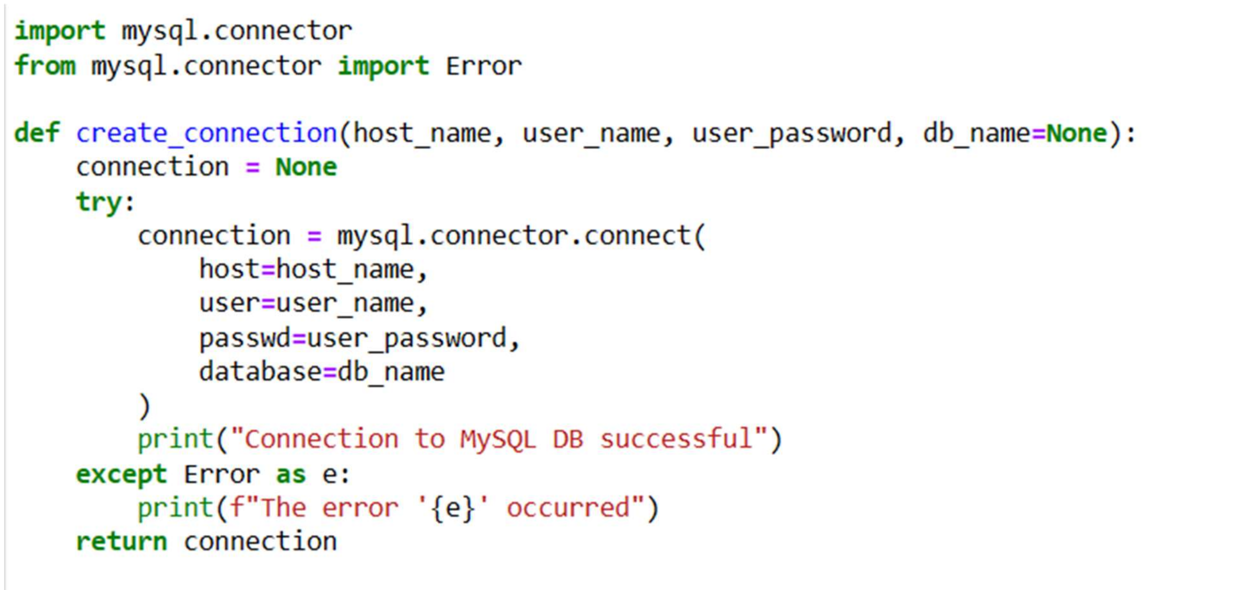
Code and Explanation:



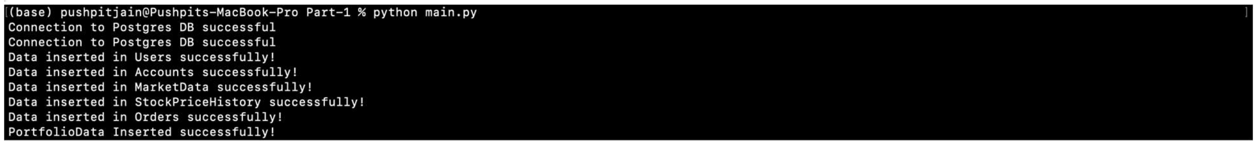
```python
import connection
import databaseCreation as dc
import initialDataInsertion as di
import dataRetrieval as dr
```

Organizing code by importing separate files for functions related to connection establishment, database creation, data insertion, and data retrieval.

1. Connection.py for **Establishing the Connection**:



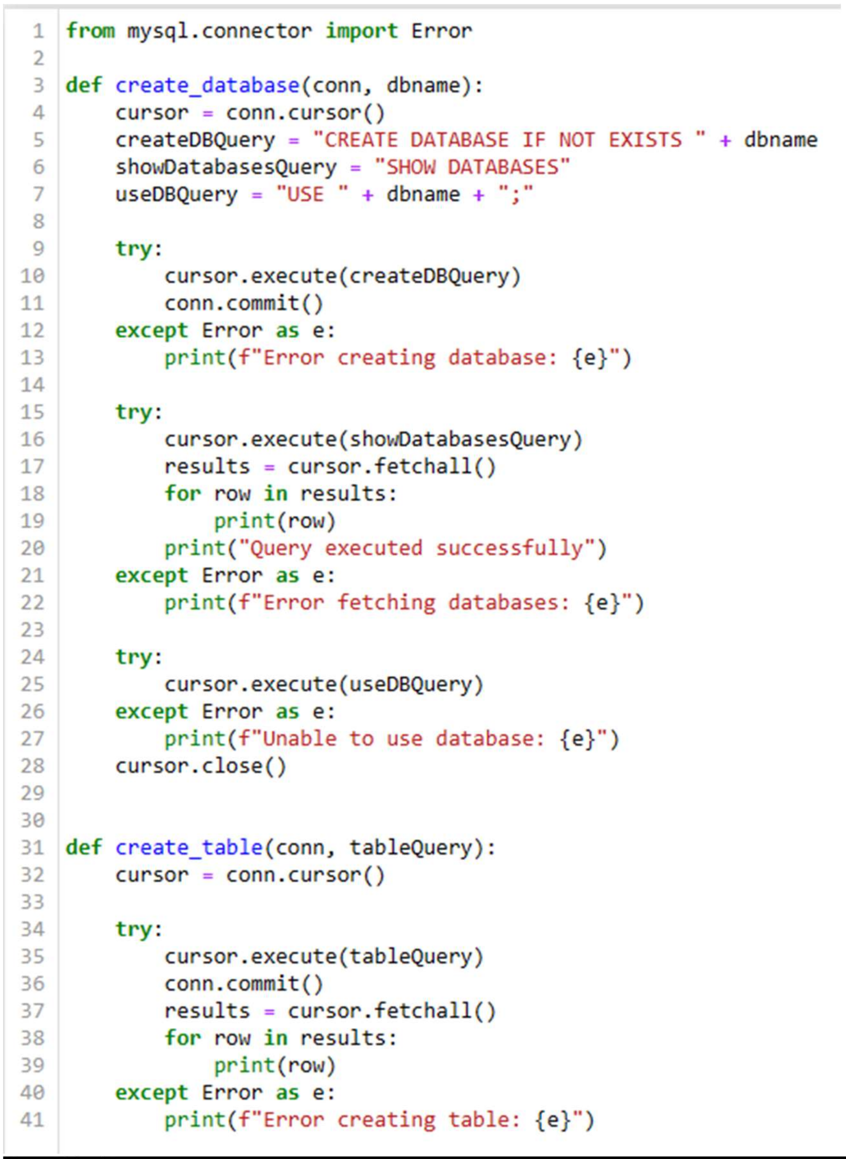
```python
import mysql.connector
from mysql.connector import Error

def create_connection(host_name, user_name, user_password, db_name=None):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host_name,
            user=user_name,
            passwd=user_password,
            database=db_name
        )
        print("Connection to MySQL DB successful")
    except Error as e:
        print(f"The error '{e}' occurred")
    return connection
```



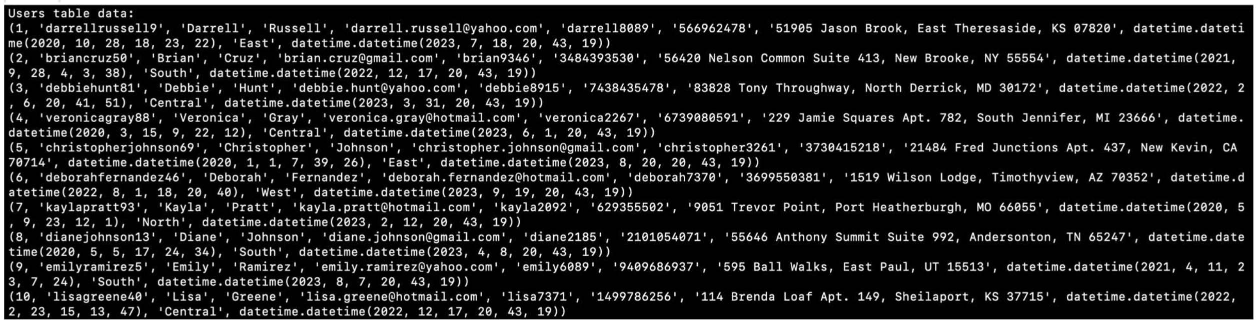
```
(base) pushpitjain@Pushpits-MacBook-Pro Part-1 % python main.py
Connection to Postgres DB successful
Connection to Postgres DB successful
Data inserted in Users successfully!
Data inserted in Accounts successfully!
Data inserted in MarketData successfully!
Data inserted in StockPriceHistory successfully!
Data inserted in Orders successfully!
PortfolioData Inserted successfully!
```

This file facilitates the function call responsible for establishing a database connection. In 'main.py', the 'create_connection' function from this file is imported and invoked. The 'create_connection' function accepts parameters such as 'host_name', 'user_name', 'user_password', and 'db_name', assigning them to their respective variables. Subsequently, these variables are utilized to interact with the MySQL command line via the mysql connector.

2. databaseCreation.py for **Database Creation**:



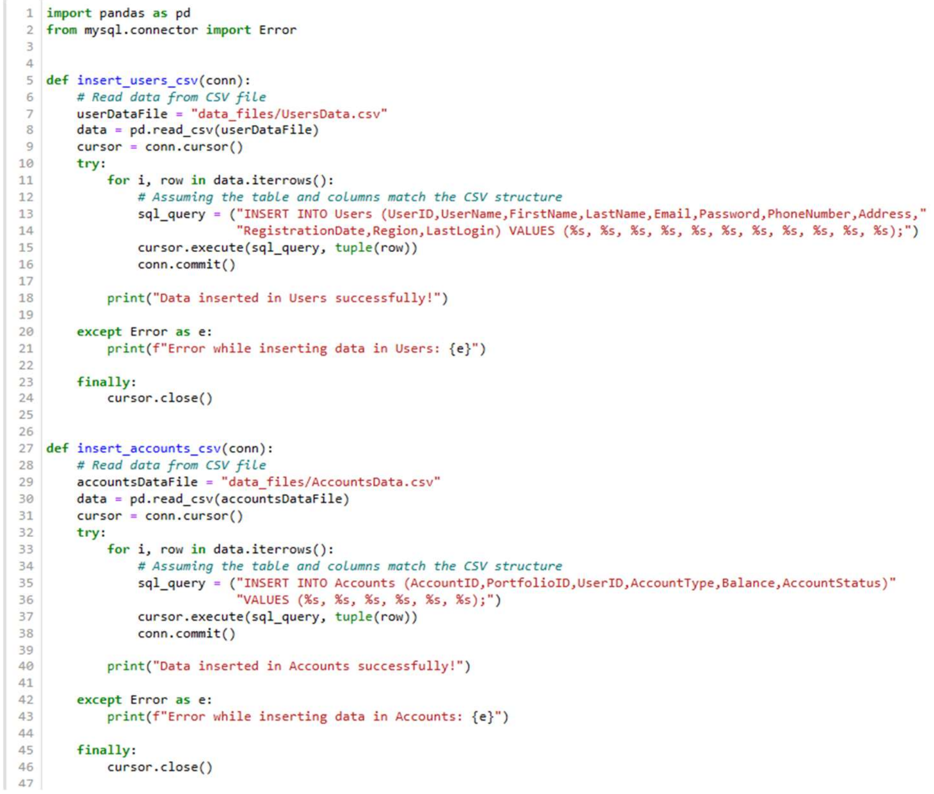
```python
from mysql.connector import Error

def create_database(conn, dbname):
    cursor = conn.cursor()
    createDBQuery = "CREATE DATABASE IF NOT EXISTS " + dbname
    showDatabasesQuery = "SHOW DATABASES"
    useDBQuery = "USE " + dbname + ";"

    try:
        cursor.execute(createDBQuery)
        conn.commit()
    except Error as e:
        print(f"Error creating database: {e}")

    try:
        cursor.execute(showDatabasesQuery)
        results = cursor.fetchall()
        for row in results:
            print(row)
        print("Query executed successfully")
    except Error as e:
        print(f"Error fetching databases: {e}")

    try:
        cursor.execute(useDBQuery)
    except Error as e:
        print(f"Unable to use database: {e}")
    cursor.close()


def create_table(conn, tableQuery):
    cursor = conn.cursor()

    try:
        cursor.execute(tableQuery)
        conn.commit()
        results = cursor.fetchall()
        for row in results:
            print(row)
    except Error as e:
        print(f"Error creating table: {e}")
```



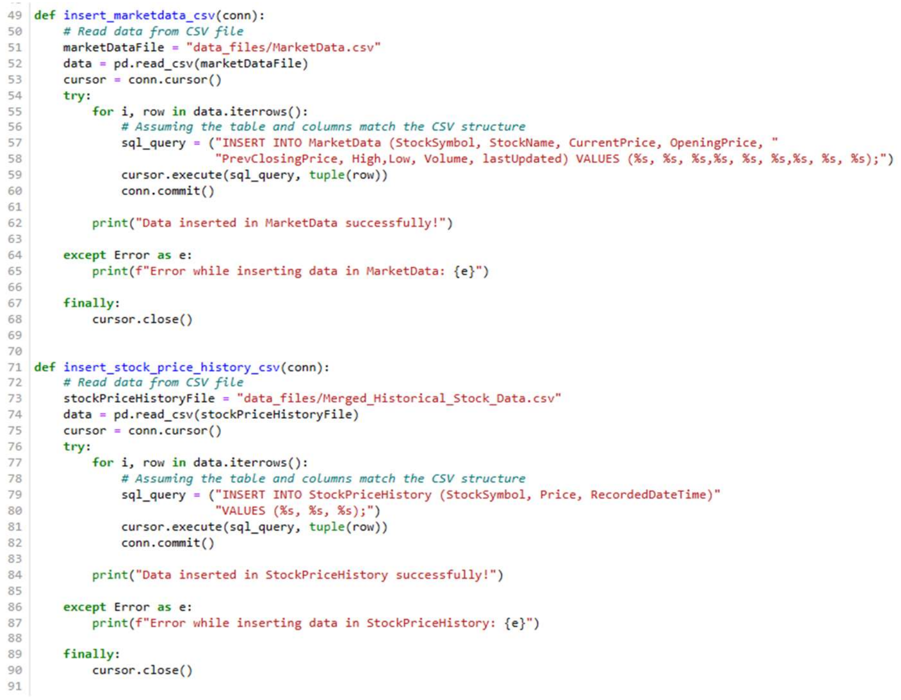
```
Users table data:
(1, 'darrellrussell9', 'Darrell', 'Russell', 'darrell.russell@yahoo.com', 'darrell8089', '566962478', '51905 Jason Brook, East Theresaside, KS 07820', datetime.dateti
me(2020, 10, 28, 18, 23, 22), 'East', datetime.datetime(2023, 7, 18, 20, 43, 19))
(2, 'briancruz50', 'Brian', 'Cruz', 'brian.cruz@gmail.com', 'brian9346', '3484393530', '56420 Nelson Common Suite 413, New Brooke, NY 55554', datetime.datetime(2021,
9, 28, 4, 3, 38), 'South', datetime.datetime(2022, 12, 17, 20, 43, 19))
(3, 'debbiehunt81', 'Debbie', 'Hunt', 'debbie.hunt@yahoo.com', 'debbie8915', '7438435478', '83828 Tony Throughway, North Derrick, MD 30172', datetime.datetime(2022, 2
, 6, 20, 41, 51), 'Central', datetime.datetime(2023, 3, 31, 20, 43, 19))
(4, 'veronicagray88', 'Veronica', 'Gray', 'veronica.gray@hotmail.com', 'veronica2267', '6739080591', '229 Jamie Squares Apt. 782, South Jennifer, MI 23666', datetime.
datetime(2020, 3, 15, 9, 22, 12), 'Central', datetime.datetime(2023, 6, 1, 20, 43, 19))
(5, 'christopherjohnson69', 'Christopher', 'Johnson', 'christopher.johnson@gmail.com', 'christopher3261', '3730415218', '21484 Fred Junctions Apt. 437, New Kevin, CA
70714', datetime.datetime(2020, 1, 1, 7, 39, 26), 'East', datetime.datetime(2023, 8, 20, 20, 43, 19))
(6, 'deborahfernandez46', 'Deborah', 'Fernandez', 'deborah.fernandez@hotmail.com', 'deborah7370', '3699550381', '1519 Wilson Lodge, Timothyview, AZ 70352', datetime.d
atetime(2022, 8, 1, 18, 20, 40), 'West', datetime.datetime(2023, 9, 19, 20, 43, 19))
(7, 'kaylapratt93', 'Kayla', 'Pratt', 'kayla.pratt@hotmail.com', 'kayla2092', '629355502', '9051 Trevor Point, Port Heatherburgh, MO 66055', datetime.datetime(2020, 5
, 9, 23, 12, 1), 'North', datetime.datetime(2023, 2, 12, 20, 43, 19))
(8, 'dianejohnson13', 'Diane', 'Johnson', 'diane.johnson@gmail.com', 'diane2185', '2101054071', '55646 Anthony Summit Suite 992, Andersonton, TN 65247', datetime.date
time(2020, 5, 5, 17, 24, 34), 'South', datetime.datetime(2023, 4, 8, 20, 43, 19))
(9, 'emilyramirez5', 'Emily', 'Ramirez', 'emily.ramirez@yahoo.com', 'emily6089', '9409686937', '595 Ball Walks, East Paul, UT 15513', datetime.datetime(2021, 4, 11, 2
3, 7, 24), 'South', datetime.datetime(2023, 8, 7, 20, 43, 19))
(10, 'lisagreene40', 'Lisa', 'Greene', 'lisa.greene@hotmail.com', 'lisa7371', '1499786256', '114 Brenda Loaf Apt. 149, Sheilaport, KS 37715', datetime.datetime(2022,
2, 23, 15, 13, 47), 'Central', datetime.datetime(2022, 12, 17, 20, 43, 19))
```

This file provides modular functions for database management tasks like creating databases and tables. The `create_database()` function allows for the creation of a database while displaying existing databases and attempting to switch to the newly created or specified database. Meanwhile, `create_table()` handles the creation of tables within the connected database. Both functions handle errors gracefully, providing informative messages for troubleshooting purposes.

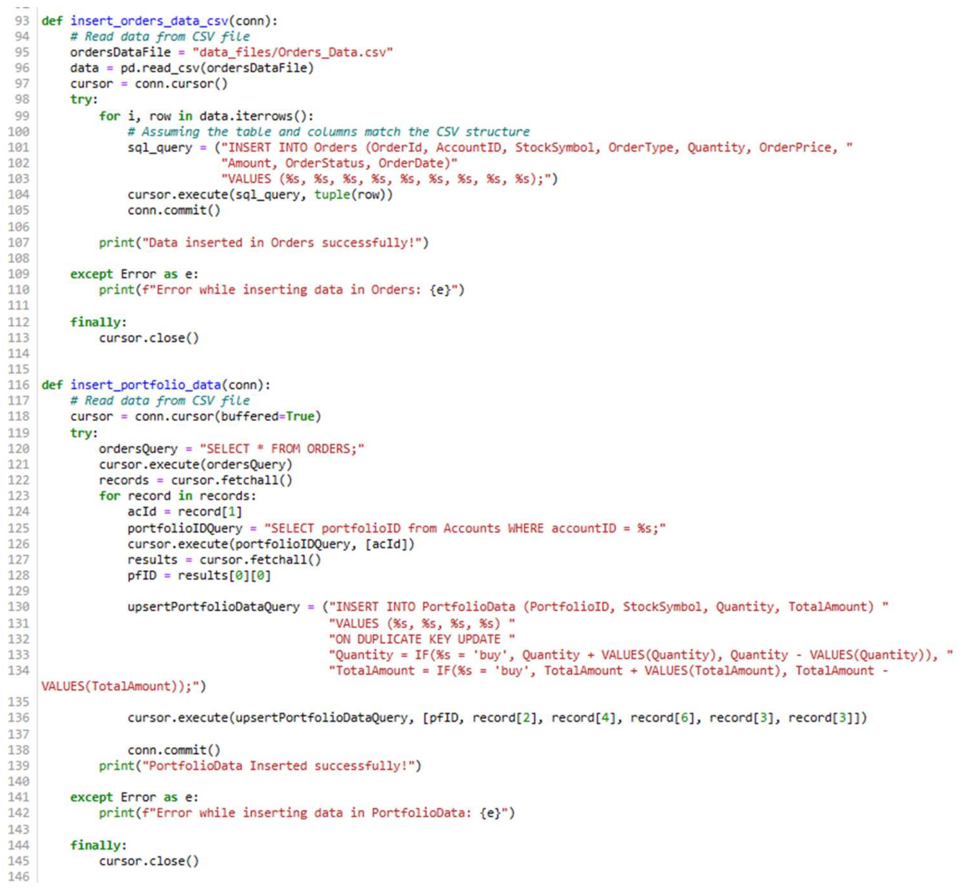
3. initialDataInsertion.py for **Data Insertion**:



```python
import pandas as pd
from mysql.connector import Error


def insert_users_csv(conn):
    # Read data from CSV file
    userDataFile = "data_files/UsersData.csv"
    data = pd.read_csv(userDataFile)
    cursor = conn.cursor()
    try:
        for i, row in data.iterrows():
            # Assuming the table and columns match the CSV structure
            sql_query = ("INSERT INTO Users (UserID,UserName,FirstName,LastName,Email,Password,PhoneNumber,Address,"
                         "RegistrationDate,Region,LastLogin) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);")
            cursor.execute(sql_query, tuple(row))
            conn.commit()

        print("Data inserted in Users successfully!")

    except Error as e:
        print(f"Error while inserting data in Users: {e}")

    finally:
        cursor.close()


def insert_accounts_csv(conn):
    # Read data from CSV file
    accountsDataFile = "data_files/AccountsData.csv"
    data = pd.read_csv(accountsDataFile)
    cursor = conn.cursor()
    try:
        for i, row in data.iterrows():
            # Assuming the table and columns match the CSV structure
            sql_query = ("INSERT INTO Accounts (AccountID,PortfolioID,UserID,AccountType,Balance,AccountStatus)"
                         "VALUES (%s, %s, %s, %s, %s, %s);")
            cursor.execute(sql_query, tuple(row))
            conn.commit()

        print("Data inserted in Accounts successfully!")

    except Error as e:
        print(f"Error while inserting data in Accounts: {e}")

    finally:
        cursor.close()

```

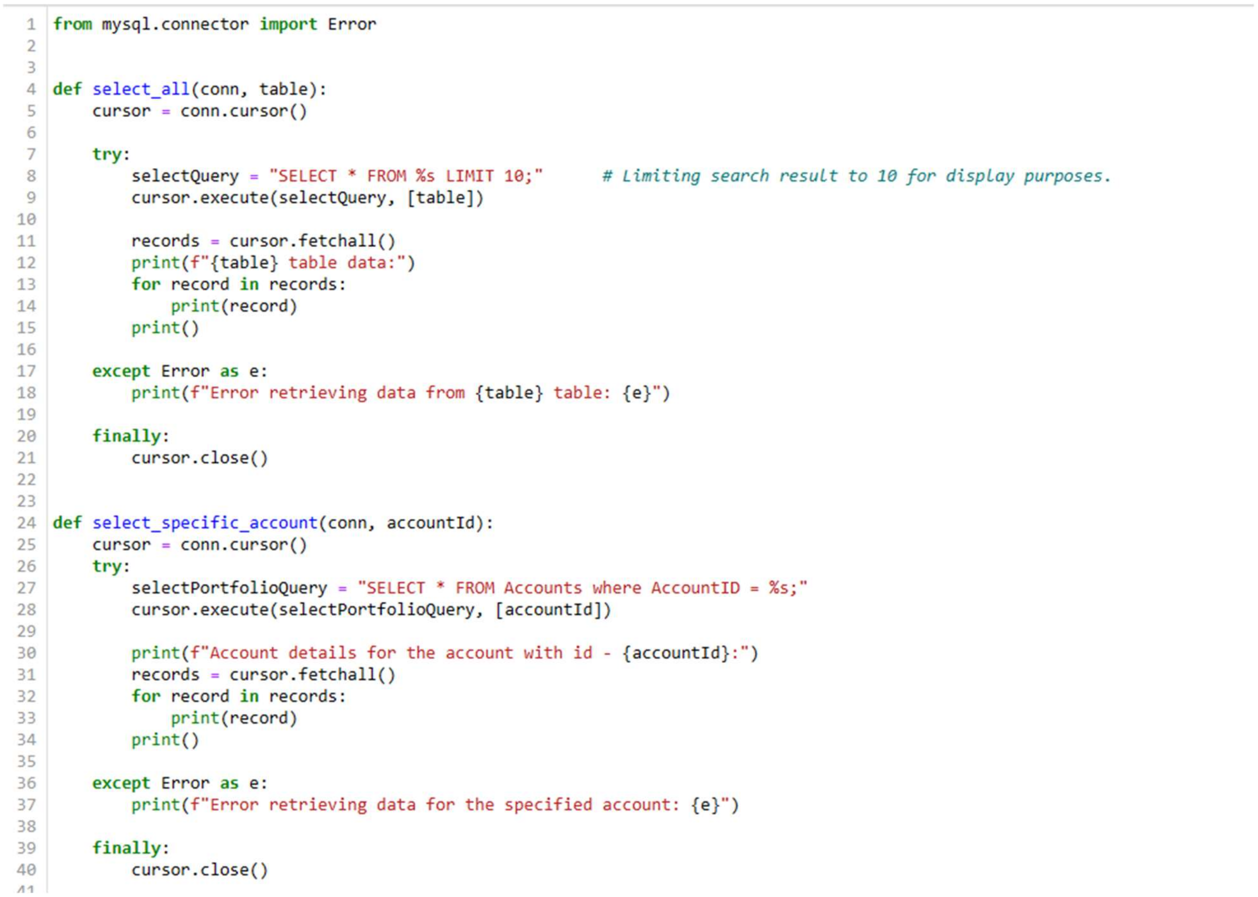


```python
def insert_marketdata_csv(conn):
    # Read data from CSV file
    marketDataFile = "data_files/MarketData.csv"
    data = pd.read_csv(marketDataFile)
    cursor = conn.cursor()
    try:
        for i, row in data.iterrows():
            # Assuming the table and columns match the CSV structure
            sql_query = ("INSERT INTO MarketData (StockSymbol, StockName, CurrentPrice, OpeningPrice, "
                         "PrevClosingPrice, High,Low, Volume, lastUpdated) VALUES (%s, %s, %s,%s, %s, %s,%s, %s, %s);")
            cursor.execute(sql_query, tuple(row))
            conn.commit()

        print("Data inserted in MarketData successfully!")

    except Error as e:
        print(f"Error while inserting data in MarketData: {e}")

    finally:
        cursor.close()


def insert_stock_price_history_csv(conn):
    # Read data from CSV file
    stockPriceHistoryFile = "data_files/Merged_Historical_Stock_Data.csv"
    data = pd.read_csv(stockPriceHistoryFile)
    cursor = conn.cursor()
    try:
        for i, row in data.iterrows():
            # Assuming the table and columns match the CSV structure
            sql_query = ("INSERT INTO StockPriceHistory (StockSymbol, Price, RecordedDateTime)"
                         "VALUES (%s, %s, %s);")
            cursor.execute(sql_query, tuple(row))
            conn.commit()

        print("Data inserted in StockPriceHistory successfully!")

    except Error as e:
        print(f"Error while inserting data in StockPriceHistory: {e}")

    finally:
        cursor.close()

```


```python
def insert_orders_data_csv(conn):
    # Read data from CSV file
    ordersDataFile = "data_files/Orders_Data.csv"
    data = pd.read_csv(ordersDataFile)
    cursor = conn.cursor()
    try:
        for i, row in data.iterrows():
            # Assuming the table and columns match the CSV structure
            sql_query = ("INSERT INTO Orders (OrderId, AccountID, StockSymbol, OrderType, Quantity, OrderPrice, "
                         "Amount, OrderStatus, OrderDate)"
                         "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s);")
            cursor.execute(sql_query, tuple(row))
            conn.commit()

        print("Data inserted in Orders successfully!")

    except Error as e:
        print(f"Error while inserting data in Orders: {e}")

    finally:
        cursor.close()


def insert_portfolio_data(conn):
    # Read data from CSV file
    cursor = conn.cursor(buffered=True)
    try:
        ordersQuery = "SELECT * FROM ORDERS;"
        cursor.execute(ordersQuery)
        records = cursor.fetchall()
        for record in records:
            acId = record[1]
            portfolioIDQuery = "SELECT portfolioID from Accounts WHERE accountID = %s;"
            cursor.execute(portfolioIDQuery, [acId])
            results = cursor.fetchall()
            pfID = results[0][0]

            upsertPortfolioDataQuery = ("INSERT INTO PortfolioData (PortfolioID, StockSymbol, Quantity, TotalAmount) "
                                        "VALUES (%s, %s, %s, %s) "
                                        "ON DUPLICATE KEY UPDATE "
                                        "Quantity = IF(%s = 'buy', Quantity + VALUES(Quantity), Quantity - VALUES(Quantity)), "
                                        "TotalAmount = IF(%s = 'buy', TotalAmount + VALUES(TotalAmount), TotalAmount -
VALUES(TotalAmount));")

            cursor.execute(upsertPortfolioDataQuery, [pfID, record[2], record[4], record[6], record[3], record[3]])

            conn.commit()
        print("PortfolioData Inserted successfully!")

    except Error as e:
        print(f"Error while inserting data in PortfolioData: {e}")

    finally:
        cursor.close()
```

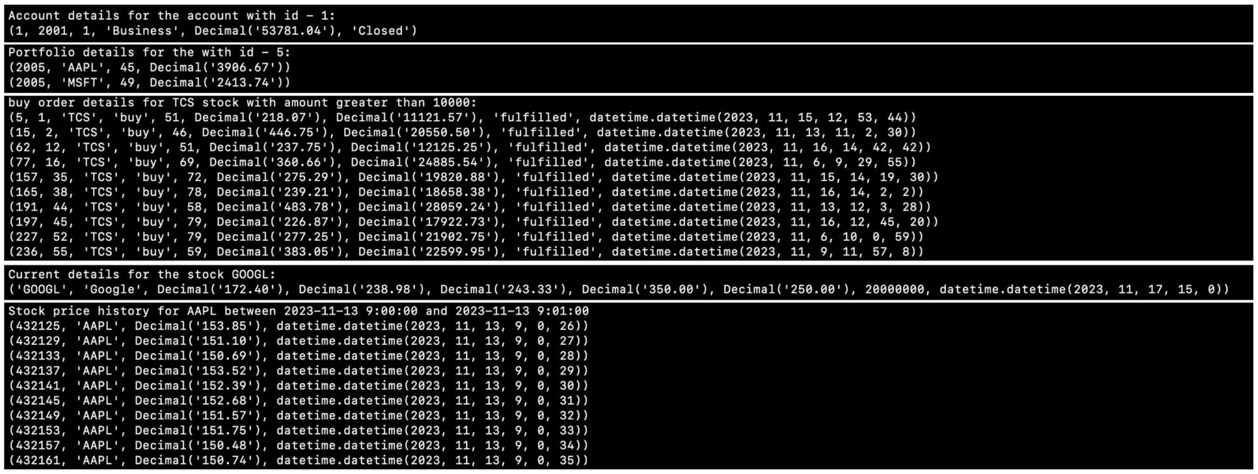
This file handles data insertion into MySQL tables by reading CSV files using Pandas and executing SQL `INSERT` queries through the MySQL Connector in Python. This Python script contains functions responsible for reading data from various CSV files (`UsersData.csv`, `AccountsData.csv`, `MarketData.csv`, `Merged_Historical_Stock_Data.csv`, `Orders_Data.csv`) and inserting it into corresponding MySQL tables (`Users`, `Accounts`, `MarketData`, `StockPriceHistory`, `Orders`, `PortfolioData`). It utilizes the `mysql.connector` library for database connection and execution of SQL queries. The script in this file iterates through the CSV data rows and executes `INSERT` queries to populate the database tables with the retrieved data. Additionally, the `insert_portfolio_data` function performs upsert operations (`INSERT...ON DUPLICATE KEY UPDATE`) based on existing records in the `PortfolioData` table. This script handles potential errors during the data insertion process and ensures proper closing of database cursors.

4. dataRetrieval.py for **Data Retrieval**:



```python
from mysql.connector import Error


def select_all(conn, table):
    cursor = conn.cursor()

    try:
        selectQuery = "SELECT * FROM %s LIMIT 10;"      # Limiting search result to 10 for display purposes.
        cursor.execute(selectQuery, [table])

        records = cursor.fetchall()
        print(f"{table} table data:")
        for record in records:
            print(record)
        print()

    except Error as e:
        print(f"Error retrieving data from {table} table: {e}")

    finally:
        cursor.close()


def select_specific_account(conn, accountId):
    cursor = conn.cursor()
    try:
        selectPortfolioQuery = "SELECT * FROM Accounts where AccountID = %s;"
        cursor.execute(selectPortfolioQuery, [accountId])

        print(f"Account details for the account with id - {accountId}:")
        records = cursor.fetchall()
        for record in records:
            print(record)
        print()

    except Error as e:
        print(f"Error retrieving data for the specified account: {e}")

    finally:
        cursor.close()
```

Data Retrieval:-



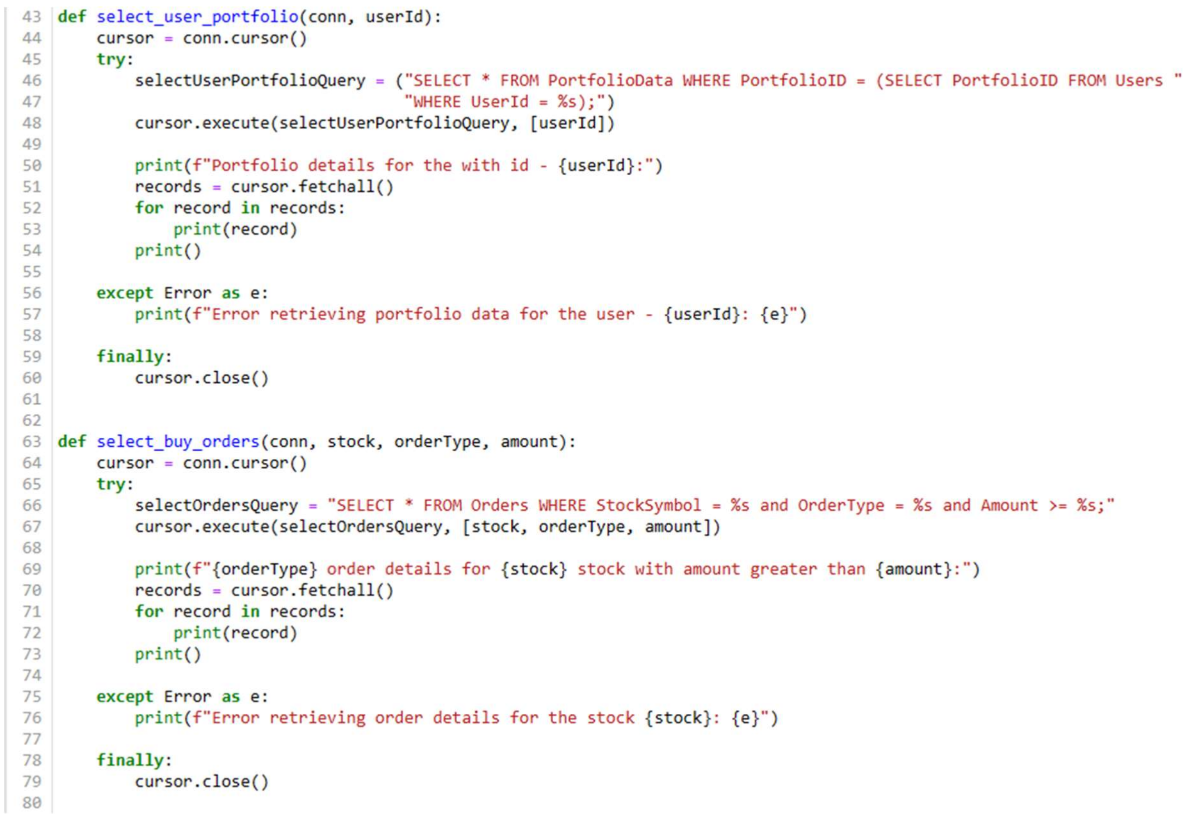
```
Account details for the account with id - 1:
(1, 2001, 1, 'Business', Decimal('53781.04'), 'Closed')
```

```
Portfolio details for the with id - 5:
(2005, 'AAPL', 45, Decimal('3906.67'))
(2005, 'MSFT', 49, Decimal('2413.74'))
```

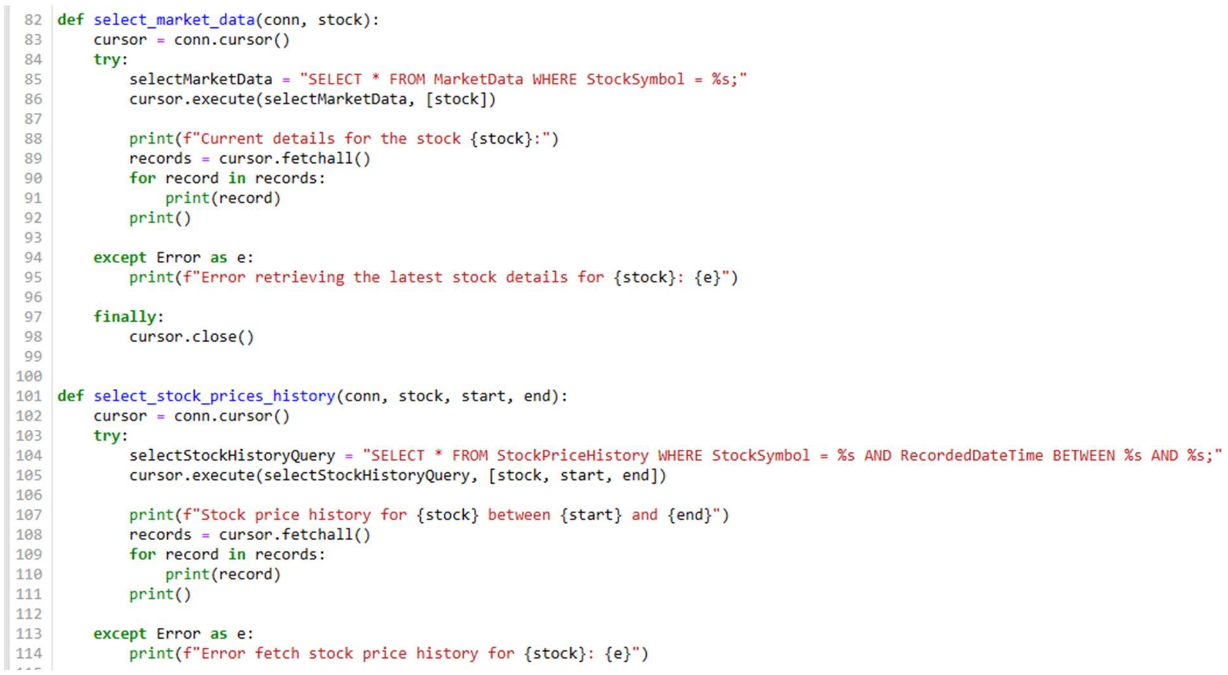
```
buy order details for TCS stock with amount greater than 10000:
(5, 1, 'TCS', 'buy', 51, Decimal('218.07'), Decimal('11121.57'), 'fulfilled', datetime.datetime(2023, 11, 15, 12, 53, 44))
(15, 2, 'TCS', 'buy', 46, Decimal('446.75'), Decimal('20550.50'), 'fulfilled', datetime.datetime(2023, 11, 13, 11, 2, 30))
(62, 12, 'TCS', 'buy', 51, Decimal('237.75'), Decimal('12125.25'), 'fulfilled', datetime.datetime(2023, 11, 16, 14, 42, 42))
(77, 16, 'TCS', 'buy', 69, Decimal('360.66'), Decimal('24885.54'), 'fulfilled', datetime.datetime(2023, 11, 6, 9, 29, 55))
(157, 35, 'TCS', 'buy', 72, Decimal('275.29'), Decimal('19820.88'), 'fulfilled', datetime.datetime(2023, 11, 15, 14, 19, 30))
(165, 38, 'TCS', 'buy', 78, Decimal('239.21'), Decimal('18658.38'), 'fulfilled', datetime.datetime(2023, 11, 16, 14, 2, 2))
(191, 44, 'TCS', 'buy', 58, Decimal('483.78'), Decimal('28059.24'), 'fulfilled', datetime.datetime(2023, 11, 13, 12, 3, 28))
(197, 45, 'TCS', 'buy', 79, Decimal('226.87'), Decimal('17922.73'), 'fulfilled', datetime.datetime(2023, 11, 16, 12, 45, 20))
(227, 52, 'TCS', 'buy', 79, Decimal('277.25'), Decimal('21902.75'), 'fulfilled', datetime.datetime(2023, 11, 6, 10, 0, 59))
(236, 55, 'TCS', 'buy', 59, Decimal('383.05'), Decimal('22599.95'), 'fulfilled', datetime.datetime(2023, 11, 9, 11, 57, 8))
```

```
Current details for the stock GOOGL:
('GOOGL', 'Google', Decimal('172.40'), Decimal('238.98'), Decimal('243.33'), Decimal('350.00'), Decimal('250.00'), 20000000, datetime.datetime(2023, 11, 17, 15, 0))
```

```
Stock price history for AAPL between 2023-11-13 9:00:00 and 2023-11-13 9:01:00
(432125, 'AAPL', Decimal('153.85'), datetime.datetime(2023, 11, 13, 9, 0, 26))
(432129, 'AAPL', Decimal('151.10'), datetime.datetime(2023, 11, 13, 9, 0, 27))
(432133, 'AAPL', Decimal('150.69'), datetime.datetime(2023, 11, 13, 9, 0, 28))
(432137, 'AAPL', Decimal('153.52'), datetime.datetime(2023, 11, 13, 9, 0, 29))
(432141, 'AAPL', Decimal('152.39'), datetime.datetime(2023, 11, 13, 9, 0, 30))
(432145, 'AAPL', Decimal('152.68'), datetime.datetime(2023, 11, 13, 9, 0, 31))
(432149, 'AAPL', Decimal('151.57'), datetime.datetime(2023, 11, 13, 9, 0, 32))
(432153, 'AAPL', Decimal('151.75'), datetime.datetime(2023, 11, 13, 9, 0, 33))
(432157, 'AAPL', Decimal('150.48'), datetime.datetime(2023, 11, 13, 9, 0, 34))
(432161, 'AAPL', Decimal('150.74'), datetime.datetime(2023, 11, 13, 9, 0, 35))
```



```python
def select_user_portfolio(conn, userId):
    cursor = conn.cursor()
    try:
        selectUserPortfolioQuery = ("SELECT * FROM PortfolioData WHERE PortfolioID = (SELECT PortfolioID FROM Users "
                                    "WHERE UserId = %s);")
        cursor.execute(selectUserPortfolioQuery, [userId])

        print(f"Portfolio details for the with id - {userId}:")
        records = cursor.fetchall()
        for record in records:
            print(record)
        print()

    except Error as e:
        print(f"Error retrieving portfolio data for the user - {userId}: {e}")

    finally:
        cursor.close()


def select_buy_orders(conn, stock, orderType, amount):
    cursor = conn.cursor()
    try:
        selectOrdersQuery = "SELECT * FROM Orders WHERE StockSymbol = %s and OrderType = %s and Amount >= %s;"
        cursor.execute(selectOrdersQuery, [stock, orderType, amount])

        print(f"{orderType} order details for {stock} stock with amount greater than {amount}:")
        records = cursor.fetchall()
        for record in records:
            print(record)
        print()

    except Error as e:
        print(f"Error retrieving order details for the stock {stock}: {e}")

    finally:
        cursor.close()

```

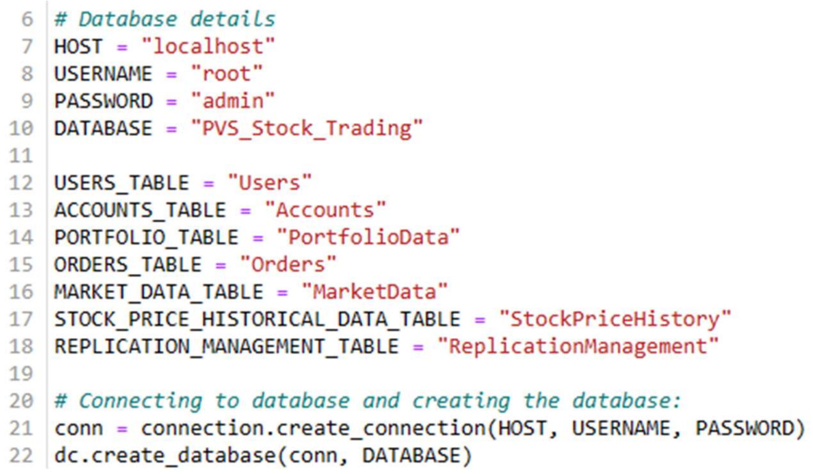


```python
def select_market_data(conn, stock):
    cursor = conn.cursor()
    try:
        selectMarketData = "SELECT * FROM MarketData WHERE StockSymbol = %s;"
        cursor.execute(selectMarketData, [stock])

        print(f"Current details for the stock {stock}:")
        records = cursor.fetchall()
        for record in records:
            print(record)
        print()

    except Error as e:
        print(f"Error retrieving the latest stock details for {stock}: {e}")

    finally:
        cursor.close()


def select_stock_prices_history(conn, stock, start, end):
    cursor = conn.cursor()
    try:
        selectStockHistoryQuery = "SELECT * FROM StockPriceHistory WHERE StockSymbol = %s AND RecordedDateTime BETWEEN %s AND %s;"
        cursor.execute(selectStockHistoryQuery, [stock, start, end])

        print(f"Stock price history for {stock} between {start} and {end}")
        records = cursor.fetchall()
        for record in records:
            print(record)
        print()

    except Error as e:
        print(f"Error fetch stock price history for {stock}: {e}")
```

This file comprises functions responsible for executing various SQL `SELECT` queries using the MySQL Connector in Python. The Python script contains functions that facilitate the retrieval of data from MySQL database tables using different SQL `SELECT` queries. The script offers functionalities to:

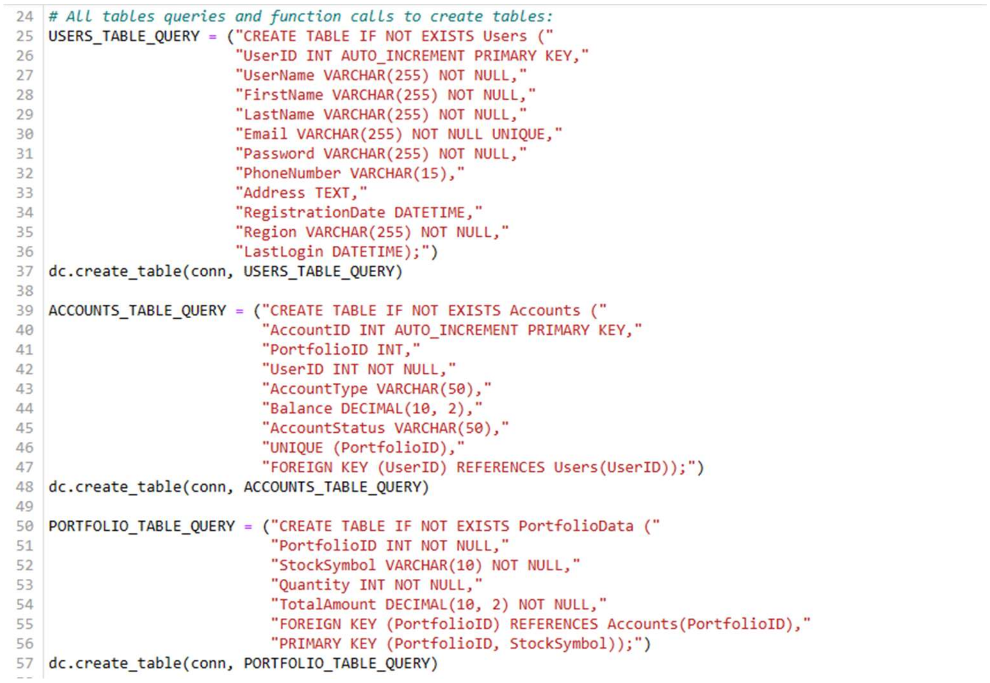
- Retrieve all data from a specified table (`select_all` function), limited to 10 records for display purposes.
- Fetch details of a specific account (`select_specific_account` function) based on the provided account ID.
- Access portfolio data for a given user (`select_user_portfolio` function) by linking the user ID with their portfolio ID.
- Retrieve specific buy orders (`select_buy_orders` function) for a particular stock and order type with an amount greater than or equal to a specified value.
- Access current market data (`select_market_data` function) for a specified stock.
- Retrieve historical stock prices (`select_stock_prices_history` function) for a particular stock within a specified time range.

Each function uses the MySQL Connector to execute the provided SQL queries against the connected database and fetches the resultant data. Error handling within these functions ensures appropriate handling and display of any encountered errors during the retrieval process.

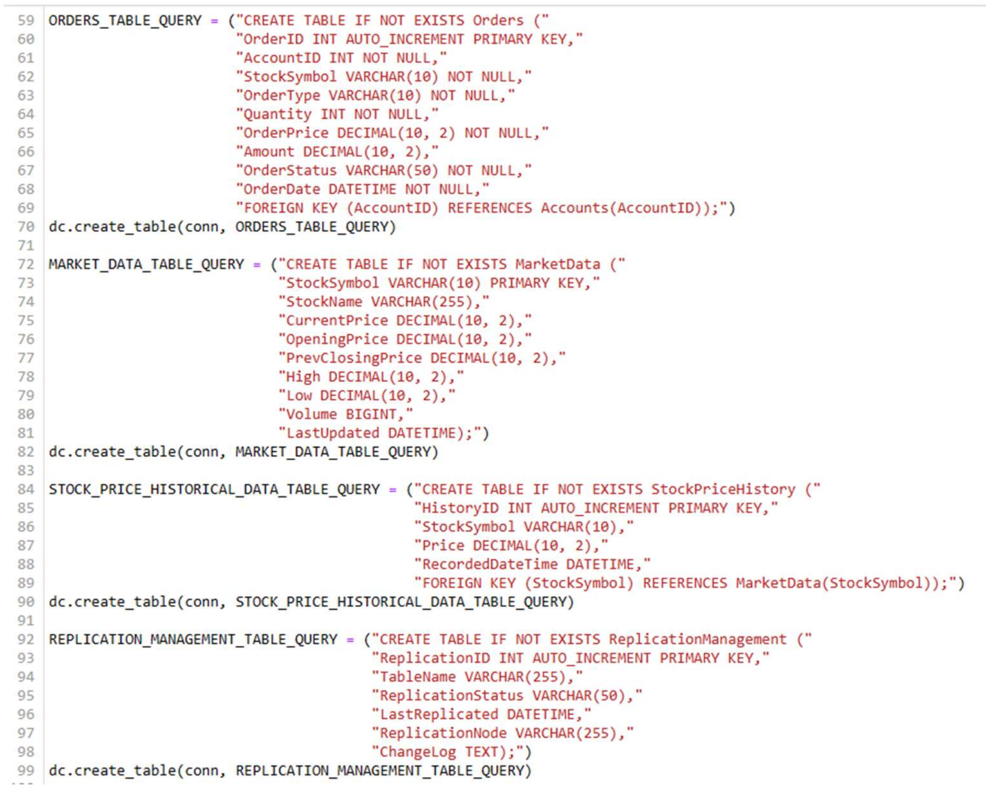


```python
# Database details
HOST = "localhost"
USERNAME = "root"
PASSWORD = "admin"
DATABASE = "PVS_Stock_Trading"

USERS_TABLE = "Users"
ACCOUNTS_TABLE = "Accounts"
PORTFOLIO_TABLE = "PortfolioData"
ORDERS_TABLE = "Orders"
MARKET_DATA_TABLE = "MarketData"
STOCK_PRICE_HISTORICAL_DATA_TABLE = "StockPriceHistory"
REPLICATION_MANAGEMENT_TABLE = "ReplicationManagement"

# Connecting to database and creating the database:
conn = connection.create_connection(HOST, USERNAME, PASSWORD)
dc.create_database(conn, DATABASE)
```

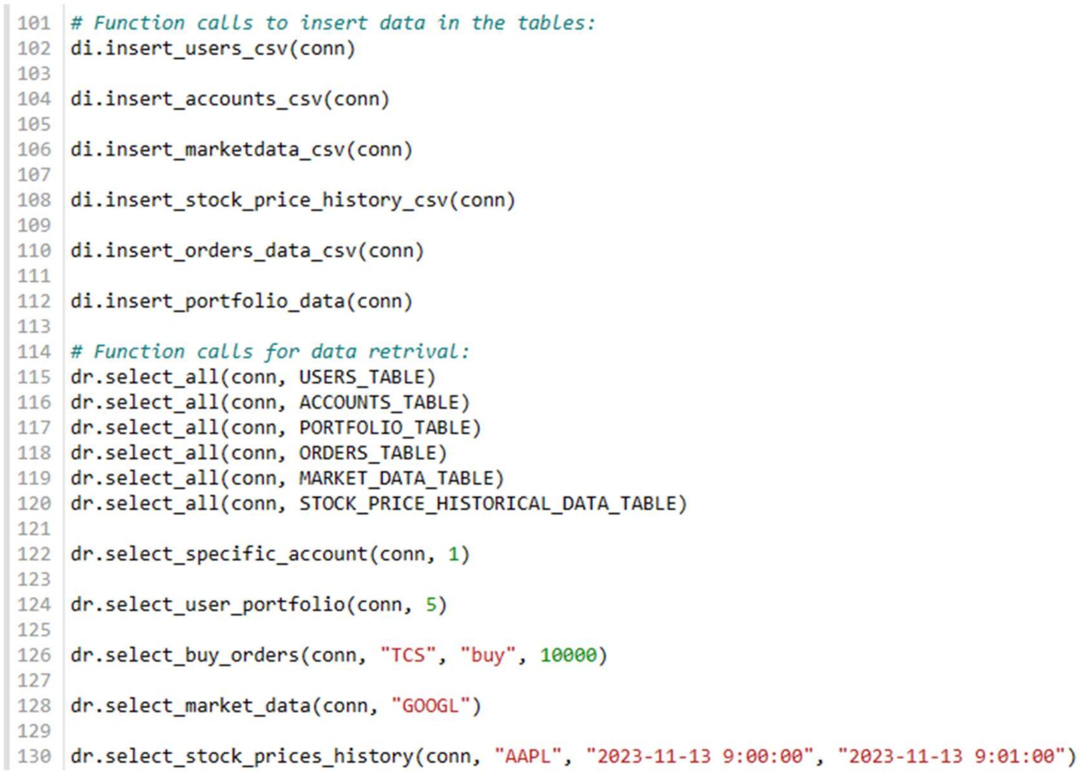
The code snippet establishes a connection to a MySQL database and creates the specified database using the provided credentials (HOST, USERNAME, PASSWORD). The code also defines various table names relevant to our "PVS_Stock_Trading" database, including tables for users, accounts, portfolio data, orders, market data, stock price historical data, and replication management. However, the provided code snippet does not explicitly create these tables. It merely initializes variables storing the table names for subsequent use within the script.



```python
# All tables queries and function calls to create tables:
USERS_TABLE_QUERY = ("CREATE TABLE IF NOT EXISTS Users ("
                     "UserID INT AUTO_INCREMENT PRIMARY KEY,"
                     "UserName VARCHAR(255) NOT NULL,"
                     "FirstName VARCHAR(255) NOT NULL,"
                     "LastName VARCHAR(255) NOT NULL,"
                     "Email VARCHAR(255) NOT NULL UNIQUE,"
                     "Password VARCHAR(255) NOT NULL,"
                     "PhoneNumber VARCHAR(15),"
                     "Address TEXT,"
                     "RegistrationDate DATETIME,"
                     "Region VARCHAR(255) NOT NULL,"
                     "LastLogin DATETIME);")
dc.create_table(conn, USERS_TABLE_QUERY)

ACCOUNTS_TABLE_QUERY = ("CREATE TABLE IF NOT EXISTS Accounts ("
                        "AccountID INT AUTO_INCREMENT PRIMARY KEY,"
                        "PortfolioID INT,"
                        "UserID INT NOT NULL,"
                        "AccountType VARCHAR(50),"
                        "Balance DECIMAL(10, 2),"
                        "AccountStatus VARCHAR(50),"
                        "UNIQUE (PortfolioID),"
                        "FOREIGN KEY (UserID) REFERENCES Users(UserID));")
dc.create_table(conn, ACCOUNTS_TABLE_QUERY)

PORTFOLIO_TABLE_QUERY = ("CREATE TABLE IF NOT EXISTS PortfolioData ("
                         "PortfolioID INT NOT NULL,"
                         "StockSymbol VARCHAR(10) NOT NULL,"
                         "Quantity INT NOT NULL,"
                         "TotalAmount DECIMAL(10, 2) NOT NULL,"
                         "FOREIGN KEY (PortfolioID) REFERENCES Accounts(PortfolioID),"
                         "PRIMARY KEY (PortfolioID, StockSymbol));")
dc.create_table(conn, PORTFOLIO_TABLE_QUERY)
```

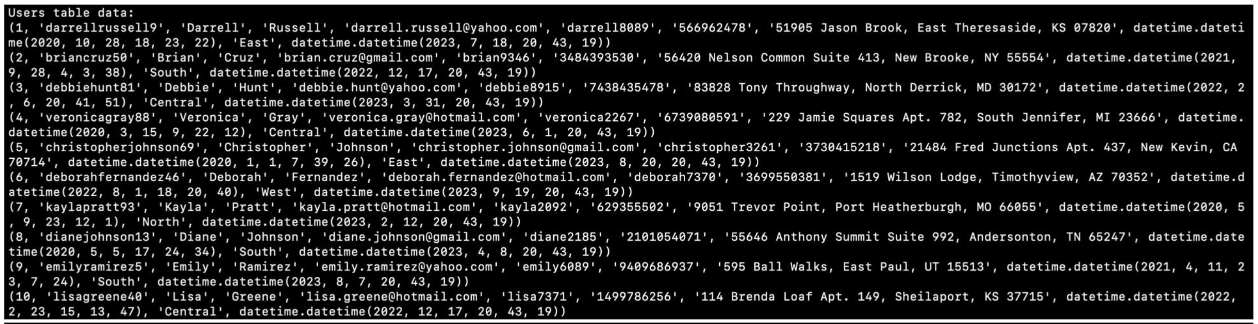


```python
ORDERS_TABLE_QUERY = ("CREATE TABLE IF NOT EXISTS Orders ("
                      "OrderID INT AUTO_INCREMENT PRIMARY KEY,"
                      "AccountID INT NOT NULL,"
                      "StockSymbol VARCHAR(10) NOT NULL,"
                      "OrderType VARCHAR(10) NOT NULL,"
                      "Quantity INT NOT NULL,"
                      "OrderPrice DECIMAL(10, 2) NOT NULL,"
                      "Amount DECIMAL(10, 2),"
                      "OrderStatus VARCHAR(50) NOT NULL,"
                      "OrderDate DATETIME NOT NULL,"
                      "FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID));")
dc.create_table(conn, ORDERS_TABLE_QUERY)

MARKET_DATA_TABLE_QUERY = ("CREATE TABLE IF NOT EXISTS MarketData ("
                           "StockSymbol VARCHAR(10) PRIMARY KEY,"
                           "StockName VARCHAR(255),"
                           "CurrentPrice DECIMAL(10, 2),"
                           "OpeningPrice DECIMAL(10, 2),"
                           "PrevClosingPrice DECIMAL(10, 2),"
                           "High DECIMAL(10, 2),"
                           "Low DECIMAL(10, 2),"
                           "Volume BIGINT,"
                           "LastUpdated DATETIME);")
dc.create_table(conn, MARKET_DATA_TABLE_QUERY)

STOCK_PRICE_HISTORICAL_DATA_TABLE_QUERY = ("CREATE TABLE IF NOT EXISTS StockPriceHistory ("
                                           "HistoryID INT AUTO_INCREMENT PRIMARY KEY,"
                                           "StockSymbol VARCHAR(10),"
                                           "Price DECIMAL(10, 2),"
                                           "RecordedDateTime DATETIME,"
                                           "FOREIGN KEY (StockSymbol) REFERENCES MarketData(StockSymbol));")
dc.create_table(conn, STOCK_PRICE_HISTORICAL_DATA_TABLE_QUERY)

REPLICATION_MANAGEMENT_TABLE_QUERY = ("CREATE TABLE IF NOT EXISTS ReplicationManagement ("
                                      "ReplicationID INT AUTO_INCREMENT PRIMARY KEY,"
                                      "TableName VARCHAR(255),"
                                      "ReplicationStatus VARCHAR(50),"
                                      "LastReplicated DATETIME,"
                                      "ReplicationNode VARCHAR(255),"
                                      "ChangeLog TEXT);")
dc.create_table(conn, REPLICATION_MANAGEMENT_TABLE_QUERY)
```

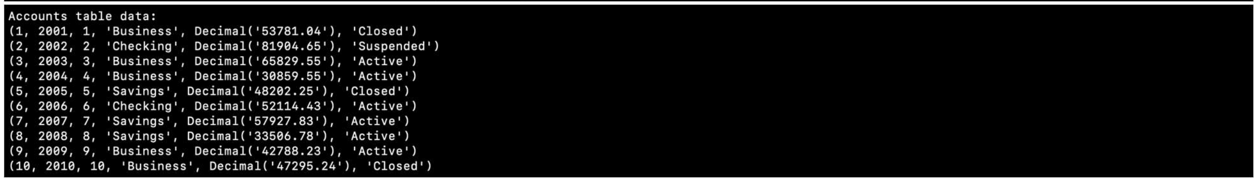


```python
# Function calls to insert data in the tables:
di.insert_users_csv(conn)

di.insert_accounts_csv(conn)

di.insert_marketdata_csv(conn)

di.insert_stock_price_history_csv(conn)

di.insert_orders_data_csv(conn)

di.insert_portfolio_data(conn)

# Function calls for data retrival:
dr.select_all(conn, USERS_TABLE)
dr.select_all(conn, ACCOUNTS_TABLE)
dr.select_all(conn, PORTFOLIO_TABLE)
dr.select_all(conn, ORDERS_TABLE)
dr.select_all(conn, MARKET_DATA_TABLE)
dr.select_all(conn, STOCK_PRICE_HISTORICAL_DATA_TABLE)

dr.select_specific_account(conn, 1)

dr.select_user_portfolio(conn, 5)

dr.select_buy_orders(conn, "TCS", "buy", 10000)

dr.select_market_data(conn, "GOOGL")

dr.select_stock_prices_history(conn, "AAPL", "2023-11-13 9:00:00", "2023-11-13 9:01:00")
```

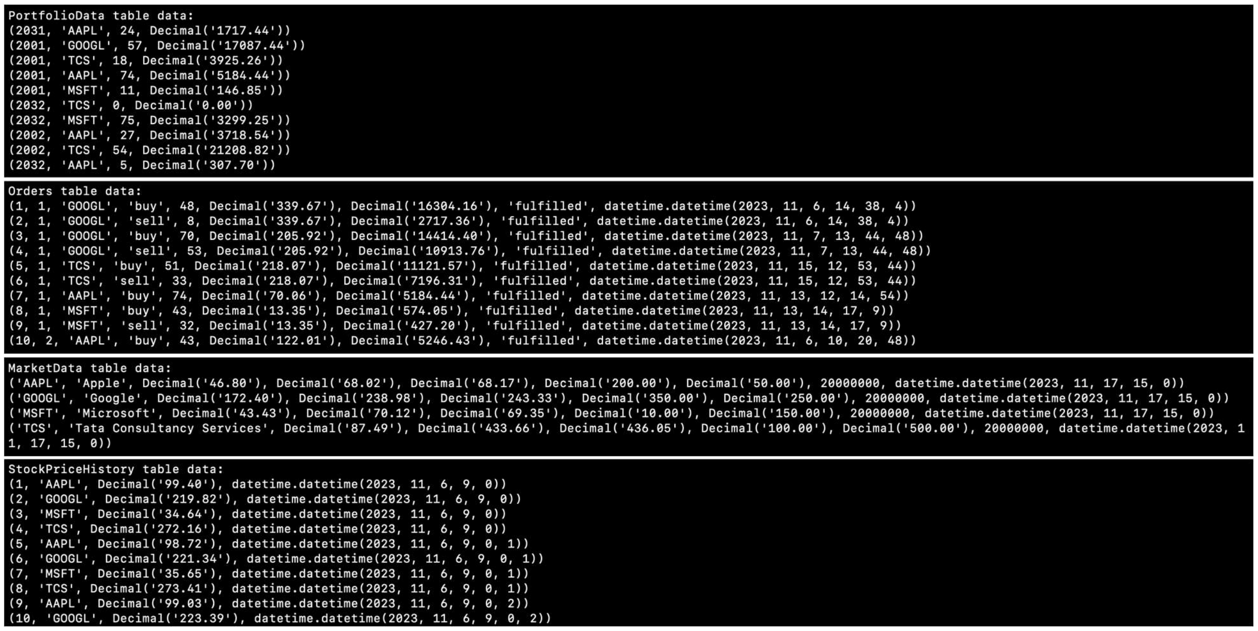
The script includes:

- Creation of tables such as 'Users', 'Accounts', 'PortfolioData', 'Orders', 'MarketData', 'StockPriceHistory', and 'ReplicationManagement' using SQL CREATE TABLE queries.
- Data insertion into these tables using functions like insert_users_csv, insert_accounts_csv, insert_marketdata_csv, insert_stock_price_history_csv, insert_orders_data_csv, and insert_portfolio_data which read data from corresponding CSV files.
- Data retrieval using functions like select_all, select_specific_account, select_user_portfolio, select_buy_orders, select_market_data, and select_stock_prices_history to fetch and display information from specific database tables based on predefined conditions or queries.
- 

```
Users table data:
(1, 'darrellrussell9', 'Darrell', 'Russell', 'darrell.russell@yahoo.com', 'darrell8089', '566962478', '51905 Jason Brook, East Theresaside, KS 07820', datetime.dateti
me(2020, 10, 28, 18, 23, 22), 'East', datetime.datetime(2023, 7, 18, 20, 43, 19))
(2, 'briancruz50', 'Brian', 'Cruz', 'brian.cruz@gmail.com', 'brian9346', '3484393530', '56420 Nelson Common Suite 413, New Brooke, NY 55554', datetime.datetime(2021,
9, 28, 4, 3, 38), 'South', datetime.datetime(2022, 12, 17, 20, 43, 19))
(3, 'debbiehunt81', 'Debbie', 'Hunt', 'debbie.hunt@yahoo.com', 'debbie8915', '7438435478', '83828 Tony Throughway, North Derrick, MD 30172', datetime.datetime(2022, 2
, 6, 20, 41, 51), 'Central', datetime.datetime(2023, 3, 31, 20, 43, 19))
(4, 'veronicagray88', 'Veronica', 'Gray', 'veronica.gray@hotmail.com', 'veronica2267', '6739080591', '229 Jamie Squares Apt. 782, South Jennifer, MI 23666', datetime.
datetime(2020, 3, 15, 9, 22, 12), 'Central', datetime.datetime(2023, 6, 1, 20, 43, 19))
(5, 'christopherjohnson69', 'Christopher', 'Johnson', 'christopher.johnson@gmail.com', 'christopher3261', '3730415218', '21484 Fred Junctions Apt. 437, New Kevin, CA
70714', datetime.datetime(2020, 1, 1, 7, 39, 26), 'East', datetime.datetime(2023, 8, 20, 20, 43, 19))
(6, 'deborahfernandez46', 'Deborah', 'Fernandez', 'deborah.fernandez@hotmail.com', 'deborah7370', '3699550381', '1519 Wilson Lodge, Timothyview, AZ 70352', datetime.d
atetime(2022, 8, 1, 18, 20, 40), 'West', datetime.datetime(2023, 9, 19, 20, 43, 19))
(7, 'kaylapratt93', 'Kayla', 'Pratt', 'kayla.pratt@hotmail.com', 'kayla2092', '629355502', '9051 Trevor Point, Port Heatherburgh, MO 66055', datetime.datetime(2020, 5
, 9, 23, 12, 1), 'North', datetime.datetime(2023, 2, 12, 20, 43, 19))
(8, 'dianejohnson13', 'Diane', 'Johnson', 'diane.johnson@gmail.com', 'diane2185', '2101054071', '55646 Anthony Summit Suite 992, Andersonton, TN 65247', datetime.date
time(2020, 5, 5, 17, 24, 34), 'South', datetime.datetime(2023, 4, 8, 20, 43, 19))
(9, 'emilyramirez5', 'Emily', 'Ramirez', 'emily.ramirez@yahoo.com', 'emily6089', '9409686937', '595 Ball Walks, East Paul, UT 15513', datetime.datetime(2021, 4, 11, 2
3, 7, 24), 'South', datetime.datetime(2023, 8, 7, 20, 43, 19))
(10, 'lisagreene40', 'Lisa', 'Greene', 'lisa.greene@hotmail.com', 'lisa7371', '1499786256', '114 Brenda Loaf Apt. 149, Sheilaport, KS 37715', datetime.datetime(2022,
2, 23, 15, 13, 47), 'Central', datetime.datetime(2022, 12, 17, 20, 43, 19))
```



```
Accounts table data:
(1, 2001, 1, 'Business', Decimal('53781.04'), 'Closed')
(2, 2002, 2, 'Checking', Decimal('81904.65'), 'Suspended')
(3, 2003, 3, 'Business', Decimal('65829.55'), 'Active')
(4, 2004, 4, 'Business', Decimal('30859.55'), 'Active')
(5, 2005, 5, 'Savings', Decimal('48202.25'), 'Closed')
(6, 2006, 6, 'Checking', Decimal('52114.43'), 'Active')
(7, 2007, 7, 'Savings', Decimal('57927.83'), 'Active')
(8, 2008, 8, 'Savings', Decimal('33506.78'), 'Active')
(9, 2009, 9, 'Business', Decimal('42788.23'), 'Active')
(10, 2010, 10, 'Business', Decimal('47295.24'), 'Closed')
```



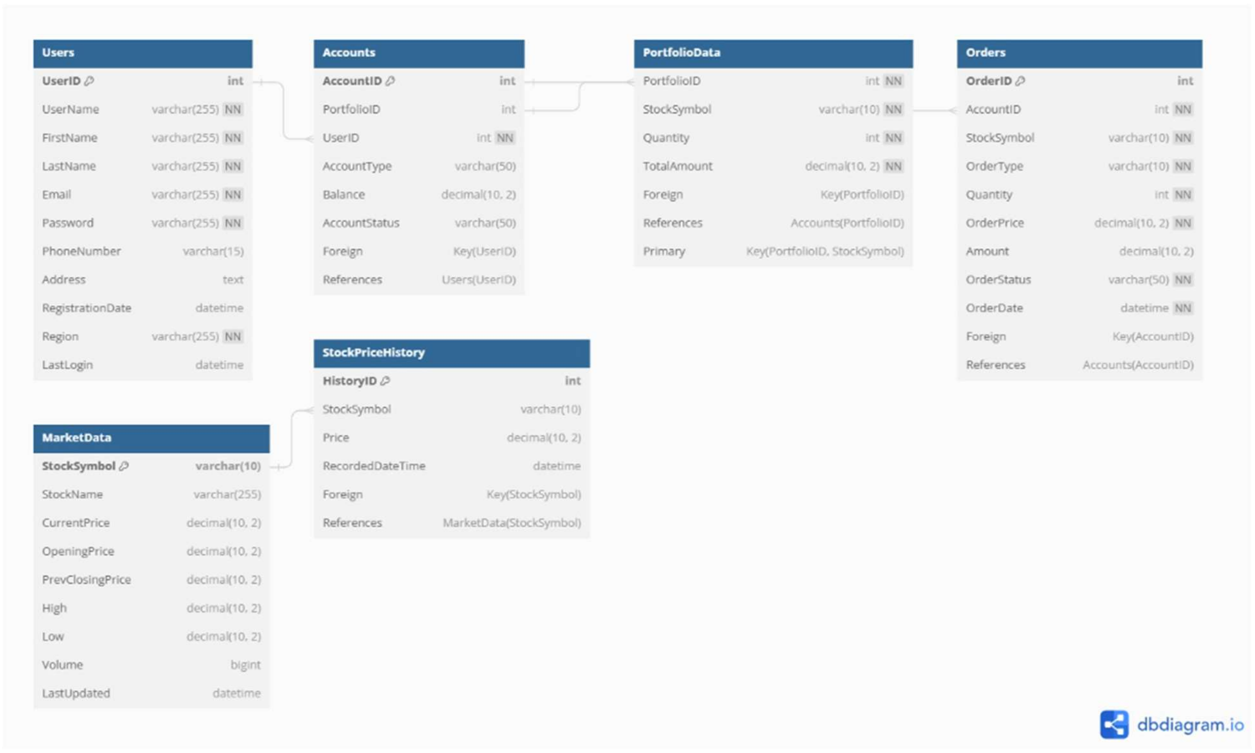
```
PortfolioData table data:
(2031, 'AAPL', 24, Decimal('1717.44'))
(2001, 'GOOGL', 57, Decimal('17087.44'))
(2001, 'TCS', 18, Decimal('3925.26'))
(2001, 'AAPL', 74, Decimal('5184.44'))
(2001, 'MSFT', 11, Decimal('146.85'))
(2032, 'TCS', 0, Decimal('0.00'))
(2032, 'MSFT', 75, Decimal('3299.25'))
(2002, 'AAPL', 27, Decimal('3718.54'))
(2002, 'TCS', 54, Decimal('21208.82'))
(2032, 'AAPL', 5, Decimal('307.70'))
```

```
Orders table data:
(1, 1, 'GOOGL', 'buy', 48, Decimal('339.67'), Decimal('16304.16'), 'fulfilled', datetime.datetime(2023, 11, 6, 14, 38, 4))
(2, 1, 'GOOGL', 'sell', 8, Decimal('339.67'), Decimal('2717.36'), 'fulfilled', datetime.datetime(2023, 11, 6, 14, 38, 4))
(3, 1, 'GOOGL', 'buy', 70, Decimal('205.92'), Decimal('14414.40'), 'fulfilled', datetime.datetime(2023, 11, 7, 13, 44, 48))
(4, 1, 'GOOGL', 'sell', 53, Decimal('205.92'), Decimal('10913.76'), 'fulfilled', datetime.datetime(2023, 11, 7, 13, 44, 48))
(5, 1, 'TCS', 'buy', 51, Decimal('218.07'), Decimal('11121.57'), 'fulfilled', datetime.datetime(2023, 11, 15, 12, 53, 44))
(6, 1, 'TCS', 'sell', 33, Decimal('218.07'), Decimal('7196.31'), 'fulfilled', datetime.datetime(2023, 11, 15, 12, 53, 44))
(7, 1, 'AAPL', 'buy', 74, Decimal('70.06'), Decimal('5184.44'), 'fulfilled', datetime.datetime(2023, 11, 13, 12, 14, 54))
(8, 1, 'MSFT', 'buy', 43, Decimal('13.35'), Decimal('574.05'), 'fulfilled', datetime.datetime(2023, 11, 13, 14, 17, 9))
(9, 1, 'MSFT', 'sell', 32, Decimal('13.35'), Decimal('427.20'), 'fulfilled', datetime.datetime(2023, 11, 13, 14, 17, 9))
(10, 2, 'AAPL', 'buy', 43, Decimal('122.01'), Decimal('5246.43'), 'fulfilled', datetime.datetime(2023, 11, 6, 10, 20, 48))
```

```
MarketData table data:
('AAPL', 'Apple', Decimal('46.80'), Decimal('68.02'), Decimal('68.17'), Decimal('200.00'), Decimal('50.00'), 20000000, datetime.datetime(2023, 11, 17, 15, 0))
('GOOGL', 'Google', Decimal('172.40'), Decimal('238.98'), Decimal('243.33'), Decimal('350.00'), Decimal('250.00'), 20000000, datetime.datetime(2023, 11, 17, 15, 0))
('MSFT', 'Microsoft', Decimal('43.43'), Decimal('70.12'), Decimal('69.35'), Decimal('10.00'), Decimal('150.00'), 20000000, datetime.datetime(2023, 11, 17, 15, 0))
('TCS', 'Tata Consultancy Services', Decimal('87.49'), Decimal('433.66'), Decimal('436.05'), Decimal('100.00'), Decimal('500.00'), 20000000, datetime.datetime(2023, 1
1, 17, 15, 0))
```

```
StockPriceHistory table data:
(1, 'AAPL', Decimal('99.40'), datetime.datetime(2023, 11, 6, 9, 0))
(2, 'GOOGL', Decimal('219.82'), datetime.datetime(2023, 11, 6, 9, 0))
(3, 'MSFT', Decimal('34.64'), datetime.datetime(2023, 11, 6, 9, 0))
(4, 'TCS', Decimal('272.16'), datetime.datetime(2023, 11, 6, 9, 0))
(5, 'AAPL', Decimal('98.72'), datetime.datetime(2023, 11, 6, 9, 0, 1))
(6, 'GOOGL', Decimal('221.34'), datetime.datetime(2023, 11, 6, 9, 0, 1))
(7, 'MSFT', Decimal('35.65'), datetime.datetime(2023, 11, 6, 9, 0, 1))
(8, 'TCS', Decimal('273.41'), datetime.datetime(2023, 11, 6, 9, 0, 1))
(9, 'AAPL', Decimal('99.03'), datetime.datetime(2023, 11, 6, 9, 0, 2))
(10, 'GOOGL', Decimal('223.39'), datetime.datetime(2023, 11, 6, 9, 0, 2))
```

Database Schema:


Users
UserID int
UserName varchar(255) NN
FirstName varchar(255) NN
LastName varchar(255) NN
Email varchar(255) NN
Password varchar(255) NN
PhoneNumber varchar(15)
Address text
RegistrationDate datetime
Region varchar(255) NN
LastLogin datetime
Accounts
AccountID int
PortfolioID int
UserID int NN
AccountType varchar(50)
Balance decimal(10, 2)
AccountStatus varchar(50)
Foreign Key(UserID)
References Users(UserID)
PortfolioData
PortfolioID int NN
StockSymbol varchar(10) NN
Quantity int NN
TotalAmount decimal(10, 2) NN
Foreign Key(PortfolioID)
References Accounts(PortfolioID)
Primary Key(PortfolioID, StockSymbol)
Orders
OrderID int
AccountID int NN
StockSymbol varchar(10) NN
OrderType varchar(10) NN
Quantity int NN
OrderPrice decimal(10, 2) NN
Amount decimal(10, 2)
OrderStatus varchar(50) NN
OrderDate datetime NN
Foreign Key(AccountID)
References Accounts(AccountID)
StockPriceHistory
HistoryID int
StockSymbol varchar(10)
Price decimal(10, 2)
RecordedDateTime datetime
Foreign Key(StockSymbol)
References MarketData(StockSymbol)
MarketData
StockSymbol varchar(10)
StockName varchar(255)
CurrentPrice decimal(10, 2)
OpeningPrice decimal(10, 2)
PrevClosingPrice decimal(10, 2)
High decimal(10, 2)
Low decimal(10, 2)
Volume bigint
LastUpdated datetime
dbdiagram.io