112-1 Database Final Project

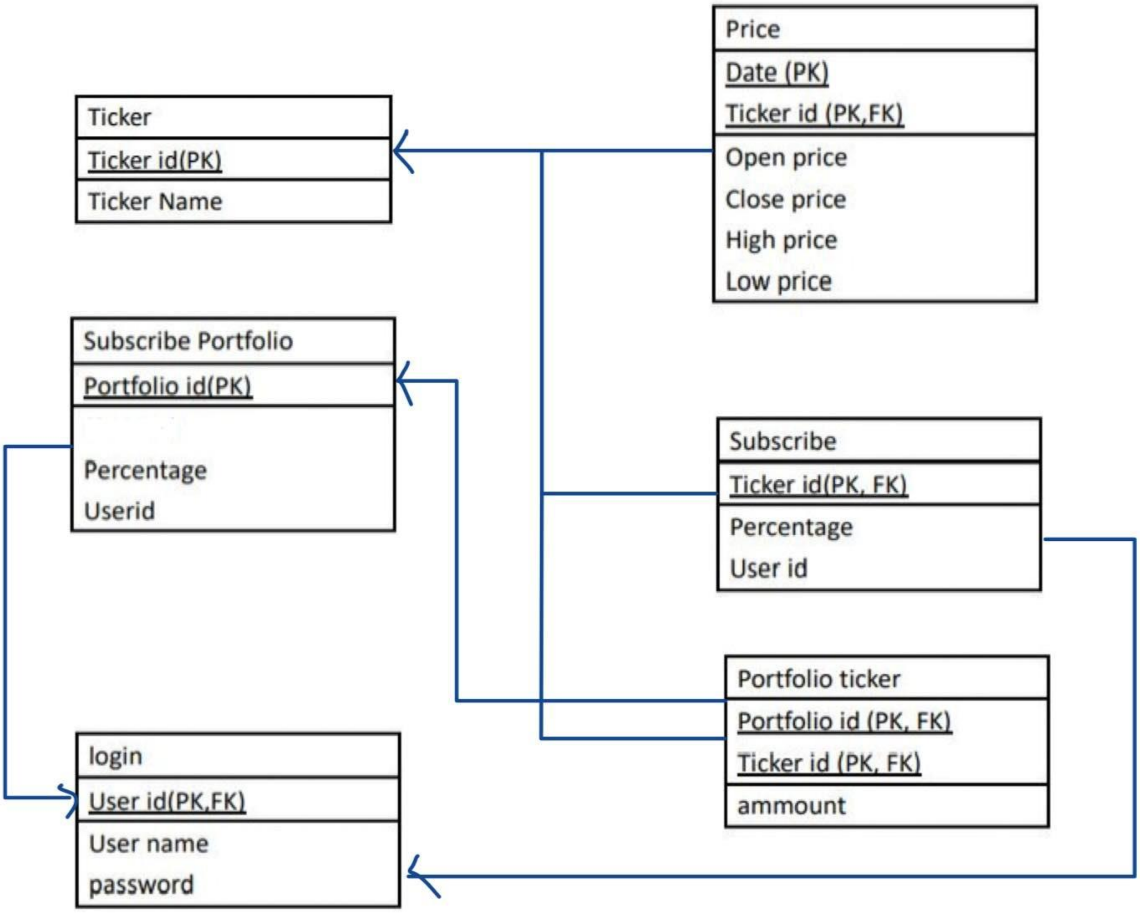
111550098 楊宗儒
111550076 楊子賝
111550124 陳燁
111550007 白冠宸
111550127 郭穎達

1. Introduction

As we age, we increasingly realize the importance of investment and financial management. Among various financial strategies, stock investment stands out due to its popularity and the ease of accessing information. Our database course project this semester offers us a chance to gain fundamental knowledge about stock investment.

By creating and utilizing a database, users can conduct detailed analyses of selected stocks. Moreover, when significant price fluctuations occur in the stocks users are monitoring, the system will issue notifications. This not only facilitates the tracking of their performance but also aids in long-term monitoring of the stock development trends. By utilizing Line Bot and web front-end to interact with users, the system provides functionalities for registration, login, adding, deleting, and querying.

2. Database Design – Schema



Ticker
Ticker id(PK)
Ticker Name

Price
Date (PK)
Ticker id (PK,FK)
Open price
Close price
High price
Low price

Subscribe Portfolio
Portfolio id(PK)
Percentage
Userid

Subscribe
Ticker id(PK, FK)
Percentage
User id

Portfolio ticker
Portfolio id (PK, FK)
Ticker id (PK, FK)
ammount

login
User id(PK,FK)
User name
password

Our system meticulously tracks various tickers, primarily focusing on those listed in the S&P 500. The majority of the schema elements are intuitive and self-explanatory.

Additionally, we have incorporated a feature called ‘subscription_portfolio', which allows users to create a personalized basket of stocks. This portfolio feature is designed to enable users to efficiently track and monitor the price movements of their selected stocks, providing a tailored experience in stock market observation.

3. Database Design – Normal Form

- **Portfolio_ticker**

  Test Normal form: BCNF

  Portfolio id is a candidate key of R, so for all non-trivial functional dependencies in $\alpha \rightarrow \beta$ in F^+, α are super key.

  Therefore, it is BCNF.

- **Subscription_portfolio**

  Test Normal form: BCNF

  {userid,Portfolio id} is a candidate key of R, so for all non-trivial functional dependencies in $\alpha \rightarrow \beta$ in F^+, α are super key.

  Therefore, it is BCNF.

- **Ticker**

  Test Normal form: BCNF

  Ticker id is a candidate key of R

  Ticker name is a candidate key of R

  So for all non-trivial functional dependencies in $\alpha \rightarrow \beta$ in F^+, α are super key.

  Therefore, it is BCNF.

- **Subscribe**

  Test Normal form: BCNF

  Test: {User id, Ticker id} is a candidate key of R, so for all non-trivial functional dependencies in $\alpha \rightarrow \beta$ in F^+, α are super key.

  Therefore, it is BCNF.

- **Price**

  Test Normal form: BCNF

  Test: {ticker id, Date} is a candidate key of R, so for all non-trivial functional dependencies in $\alpha \rightarrow \beta$ in F^+, α are super key.

  Therefore, it is BCNF.

- **login**

  Test Normal form: BCNF

  Test: userid is a candidate key of R, so for all non-trivial functional dependencies in $\alpha \rightarrow \beta$ in F^+, α are super key.

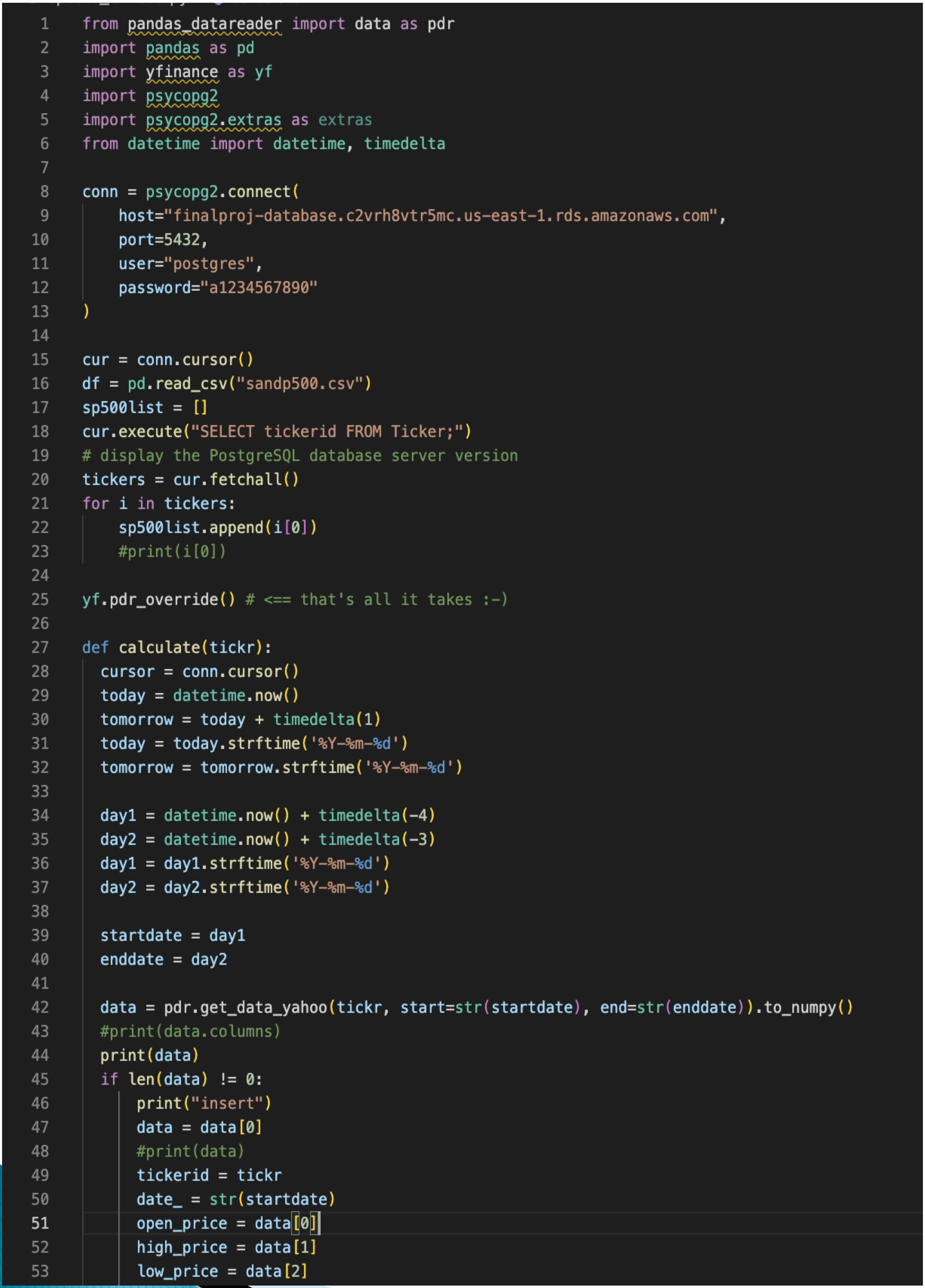
  Therefore, it is BCNF.

4. From the Data Sources to the Database – The Original Format

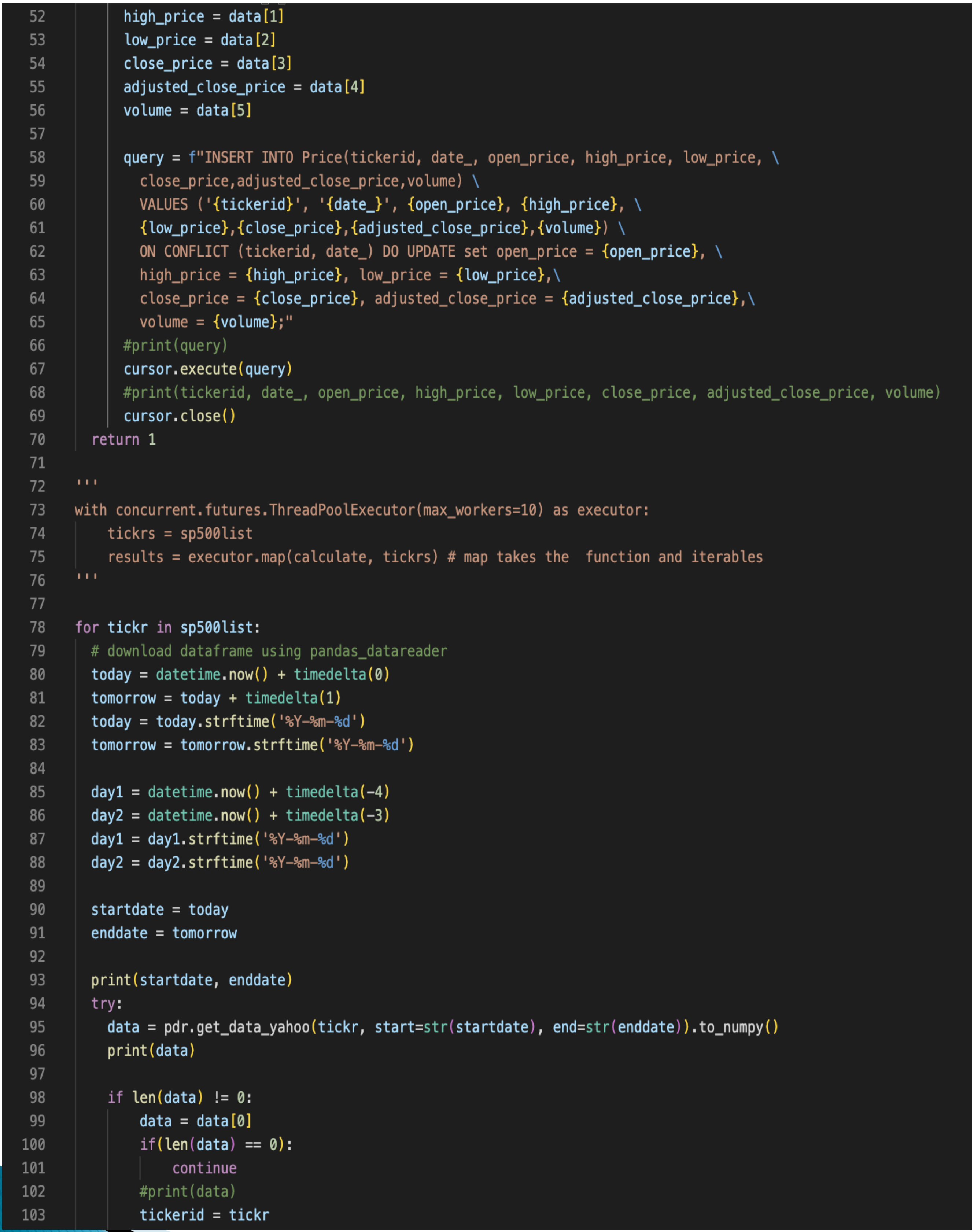
Date	Open	High	Low	Close	Adj Close	Volume
2024-01-03	187.149994	185.880005	183.940002	184.975006	184.975006	6888484

5. From the Data Sources to the Database – Import and Update the Data

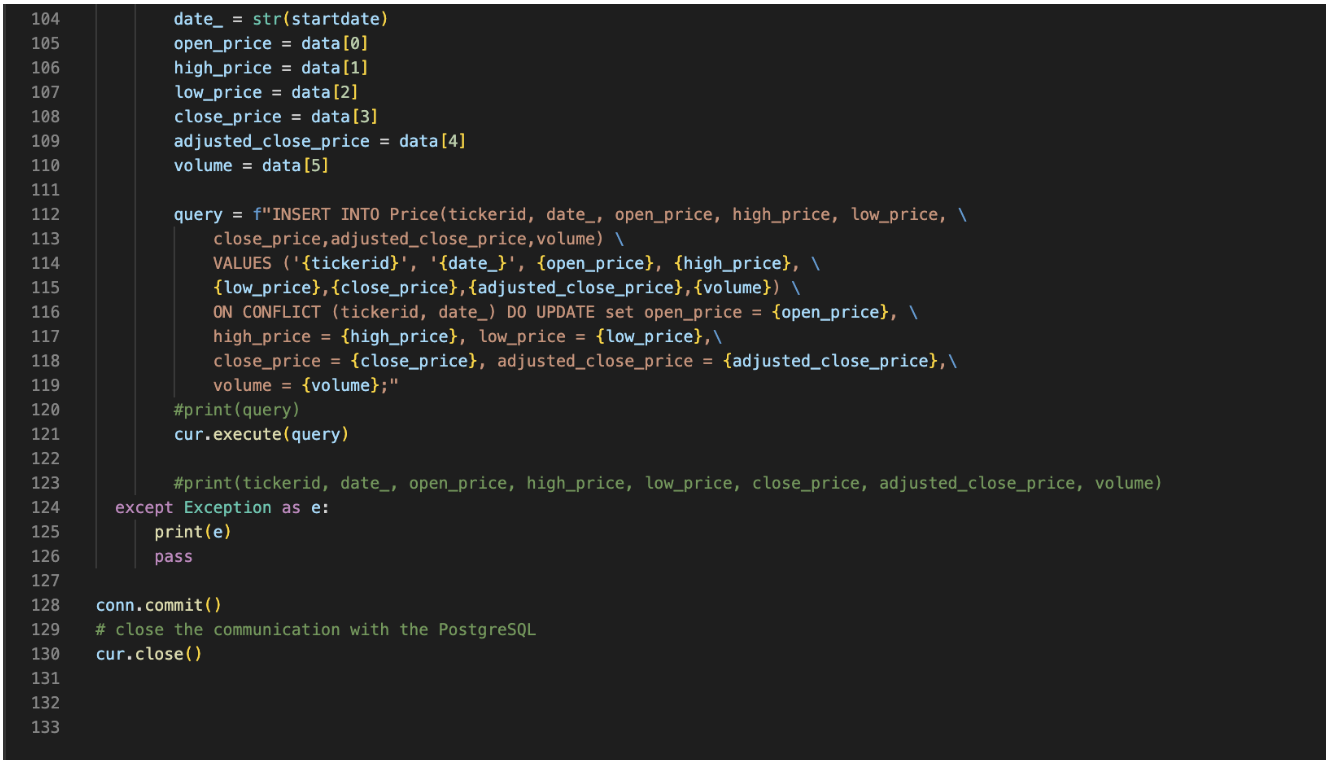
We run the scraping script on EC2 and use crontab to run the script and update the data every 5 minutes. For every 5 minutes, we load the stock list in the database and call the api to get the current price of every stock, and update(or insert) the value into the database.



```python
from pandas_datareader import data as pdr
import pandas as pd
import yfinance as yf
import psycopg2
import psycopg2.extras as extras
from datetime import datetime, timedelta

conn = psycopg2.connect(
    host="finalproj-database.c2vrh8vtr5mc.us-east-1.rds.amazonaws.com",
    port=5432,
    user="postgres",
    password="a1234567890"
)

cur = conn.cursor()
df = pd.read_csv("sandp500.csv")
sp500list = []
cur.execute("SELECT tickerid FROM Ticker;")
# display the PostgreSQL database server version
tickers = cur.fetchall()
for i in tickers:
    sp500list.append(i[0])
    #print(i[0])

yf.pdr_override() # <== that's all it takes :-)

def calculate(tickr):
  cursor = conn.cursor()
  today = datetime.now()
  tomorrow = today + timedelta(1)
  today = today.strftime('%Y-%m-%d')
  tomorrow = tomorrow.strftime('%Y-%m-%d')

  day1 = datetime.now() + timedelta(-4)
  day2 = datetime.now() + timedelta(-3)
  day1 = day1.strftime('%Y-%m-%d')
  day2 = day2.strftime('%Y-%m-%d')

  startdate = day1
  enddate = day2

  data = pdr.get_data_yahoo(tickr, start=str(startdate), end=str(enddate)).to_numpy()
  #print(data.columns)
  print(data)
  if len(data) != 0:
      print("insert")
      data = data[0]
      #print(data)
      tickerid = tickr
      date_ = str(startdate)
      open_price = data[0]
      high_price = data[1]
      low_price = data[2]
```



```
      high_price = data[1]
      low_price = data[2]
      close_price = data[3]
      adjusted_close_price = data[4]
      volume = data[5]

      query = f"INSERT INTO Price(tickerid, date_, open_price, high_price, low_price, \
        close_price,adjusted_close_price,volume) \
        VALUES ('{tickerid}', '{date_}', {open_price}, {high_price}, \
        {low_price},{close_price},{adjusted_close_price},{volume}) \
        ON CONFLICT (tickerid, date_) DO UPDATE set open_price = {open_price}, \
        high_price = {high_price}, low_price = {low_price},\
        close_price = {close_price}, adjusted_close_price = {adjusted_close_price},\
        volume = {volume};"
      #print(query)
      cursor.execute(query)
      #print(tickerid, date_, open_price, high_price, low_price, close_price, adjusted_close_price, volume)
      cursor.close()
  return 1

'''
with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
    tickrs = sp500list
    results = executor.map(calculate, tickrs) # map takes the  function and iterables
'''

for tickr in sp500list:
  # download dataframe using pandas_datareader
  today = datetime.now() + timedelta(0)
  tomorrow = today + timedelta(1)
  today = today.strftime('%Y-%m-%d')
  tomorrow = tomorrow.strftime('%Y-%m-%d')

  day1 = datetime.now() + timedelta(-4)
  day2 = datetime.now() + timedelta(-3)
  day1 = day1.strftime('%Y-%m-%d')
  day2 = day2.strftime('%Y-%m-%d')

  startdate = today
  enddate = tomorrow

  print(startdate, enddate)
  try:
    data = pdr.get_data_yahoo(tickr, start=str(startdate), end=str(enddate)).to_numpy()
    print(data)

    if len(data) != 0:
        data = data[0]
        if(len(data) == 0):
            continue
        #print(data)
        tickerid = tickr
```



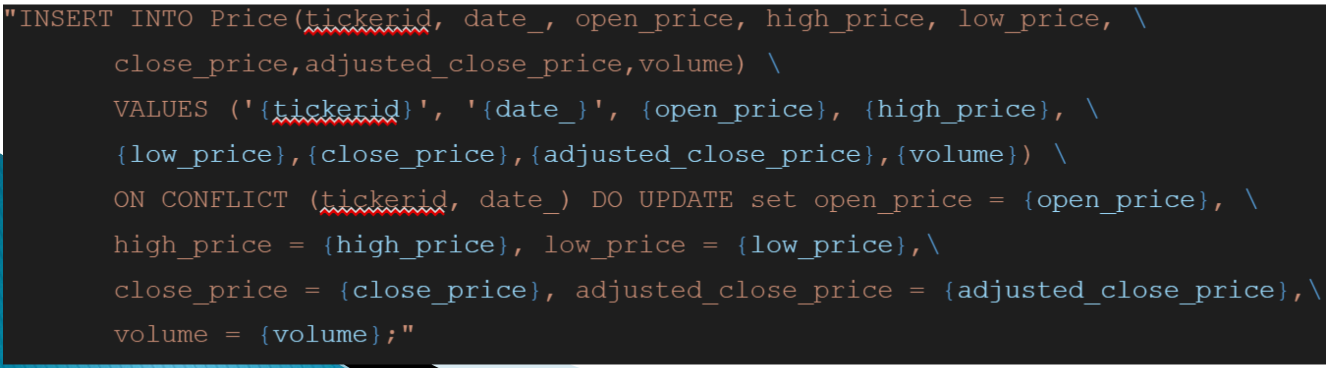
```
        date_ = str(startdate)
        open_price = data[0]
        high_price = data[1]
        low_price = data[2]
        close_price = data[3]
        adjusted_close_price = data[4]
        volume = data[5]

        query = f"INSERT INTO Price(tickerid, date_, open_price, high_price, low_price, \
            close_price,adjusted_close_price,volume) \
            VALUES ('{tickerid}', '{date_}', {open_price}, {high_price}, \
            {low_price},{close_price},{adjusted_close_price},{volume}) \
            ON CONFLICT (tickerid, date_) DO UPDATE set open_price = {open_price}, \
            high_price = {high_price}, low_price = {low_price},\
            close_price = {close_price}, adjusted_close_price = {adjusted_close_price},\
            volume = {volume};"
        #print(query)
        cur.execute(query)

        #print(tickerid, date_, open_price, high_price, low_price, close_price, adjusted_close_price, volume)
  except Exception as e:
      print(e)
      pass

conn.commit()
# close the communication with the PostgreSQL
cur.close()
```

6. Why our Application needs a Database?

1) To hold the historical data of a stock to provide the historical price of a portfolio.
2) To maintain the portfolio that the users create.
3) To maintain the subscription that users made.

7. Application with Database
– How the queries performed

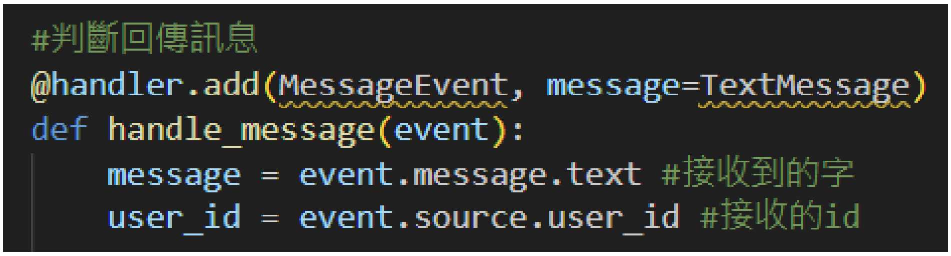
We use the psycopg2 python library to connect to our database and do the query, here is how we update the price data.



```
"INSERT INTO Price(tickerid, date_, open_price, high_price, low_price, \
      close_price,adjusted_close_price,volume) \
      VALUES ('{tickerid}', '{date_}', {open_price}, {high_price}, \
      {low_price},{close_price},{adjusted_close_price},{volume}) \
      ON CONFLICT (tickerid, date_) DO UPDATE set open_price = {open_price}, \
      high_price = {high_price}, low_price = {low_price},\
      close_price = {close_price}, adjusted_close_price = {adjusted_close_price},\
      volume = {volume};"
```

To design dialogues with customers using the LineBot-API, we detect keywords needed for adding, deleting, and querying functionalities. These keywords are then integrated into Python's psycopg2 library to execute queries. After executing the queries, the returned tables are formatted into text and sent back to the customer.

To recognize and process customer data, our tables use Line's user_id as the primary key. Additionally, the API's Message event can receive the user's id.

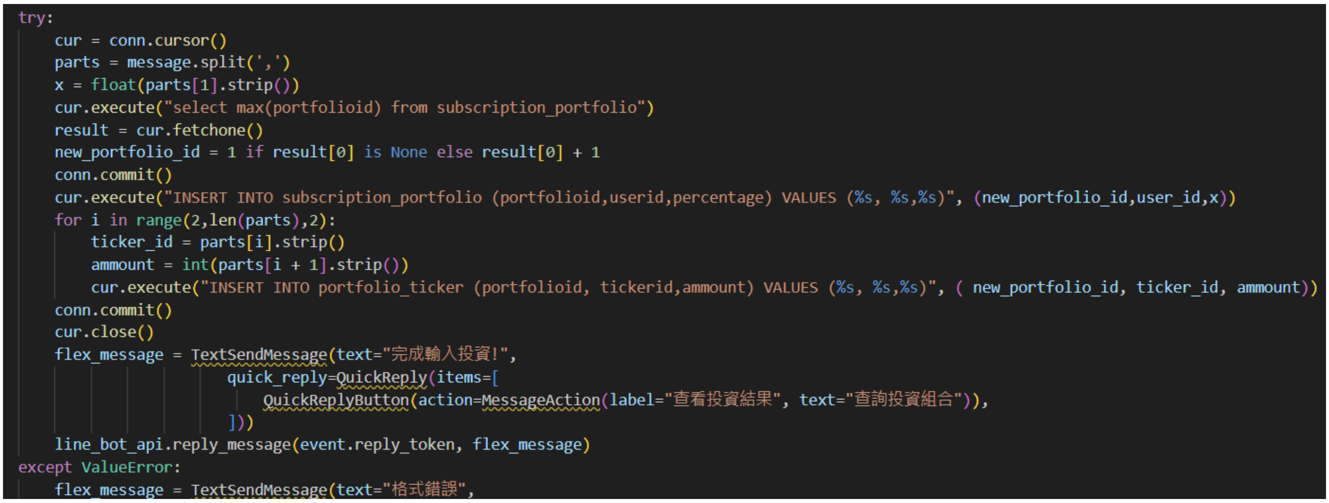


```
#判斷回傳訊息
@handler.add(MessageEvent, message=TextMessage)
def handle_message(event):
    message = event.message.text #接收到的字
    user_id = event.source.user_id #接收的id
```

We have uploaded the code related to LineBot to GitHub: https://github.com/David810209/database-FinalProject

Here are all the features that utilize queries:

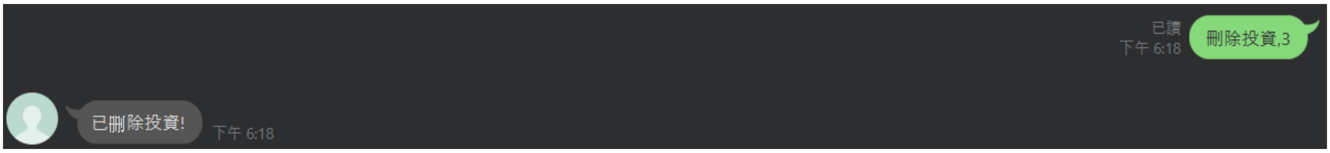
1) Adding an Investment Portfolio:



```
try:
    cur = conn.cursor()
    parts = message.split(',')
    x = float(parts[1].strip())
    cur.execute("select max(portfolioid) from subscription_portfolio")
    result = cur.fetchone()
    new_portfolio_id = 1 if result[0] is None else result[0] + 1
    conn.commit()
    cur.execute("INSERT INTO subscription_portfolio (portfolioid,userid,percentage) VALUES (%s, %s,%s)", (new_portfolio_id,user_id,x))
    for i in range(2,len(parts),2):
        ticker_id = parts[i].strip()
        ammount = int(parts[i + 1].strip())
        cur.execute("INSERT INTO portfolio_ticker (portfolioid, tickerid,ammount) VALUES (%s, %s,%s)", ( new_portfolio_id, ticker_id, ammount))
    conn.commit()
    cur.close()
    flex_message = TextSendMessage(text="完成輸入投資!",
                       quick_reply=QuickReply(items=[
                           QuickReplyButton(action=MessageAction(label="查看投資結果", text="查詢投資組合")),
                       ]))
    line_bot_api.reply_message(event.reply_token, flex_message)
except ValueError:
    flex_message = TextSendMessage(text="格式錯誤",
```

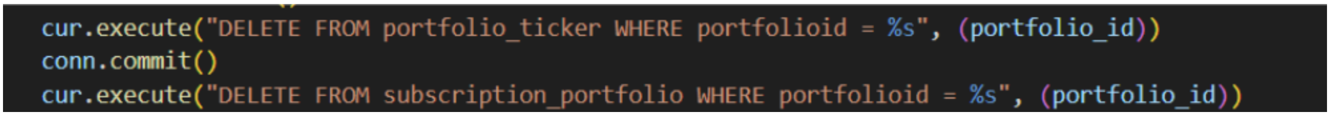
First, select the current largest id from the database table and increment it by one to generate a unique portfolio_id for each entry. Then, insert the column content obtained from the customer into the query and insert it into two tables: subscription_portfolio and portfolio_ticker, using psycopg2 for execution. Additionally, we have designed error handling for incorrect user input formats.

2) Deleting an Investment Portfolio:



已讀
下午 6:18
刪除投資,3

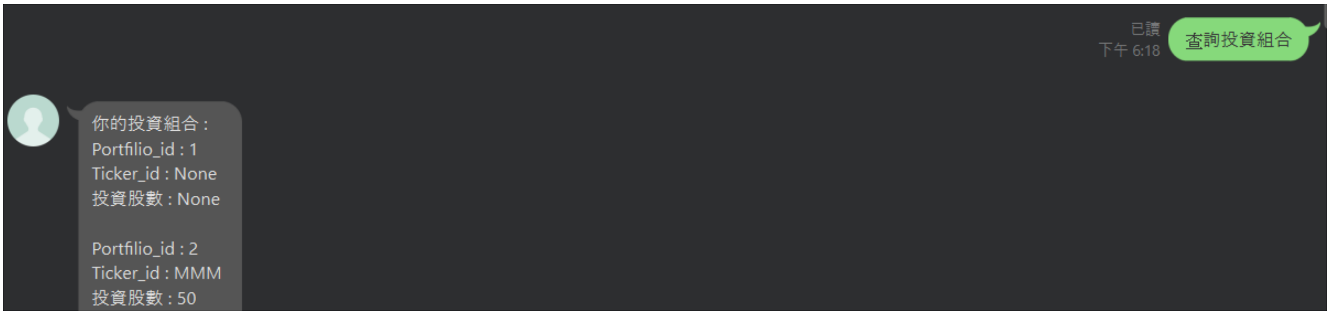
已刪除投資!
下午 6:18



```
cur.execute("DELETE FROM portfolio_ticker WHERE portfolioid = %s", (portfolio_id))
conn.commit()
cur.execute("DELETE FROM subscription_portfolio WHERE portfolioid = %s", (portfolio_id))
```

To delete an investment portfolio, first remove entries from the portfolio_ticker table, then from the subscription_portfolio table, due to their foreign key relationship. A confirmation message appears post-deletion. Error handling is included in this process.

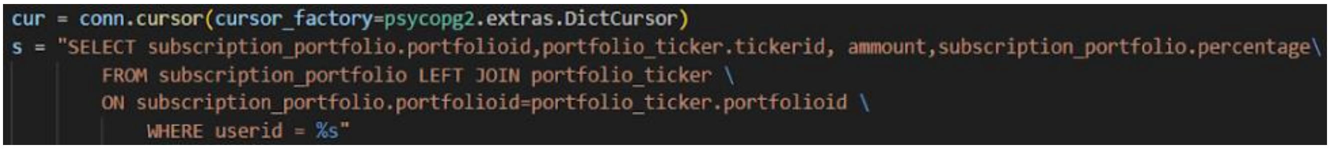
3) Querying Investment Portfolio Contents:



已讀
下午 6:18
查詢投資組合

你的投資組合：
Portfilio_id : 1
Ticker_id : None
投資股數 : None

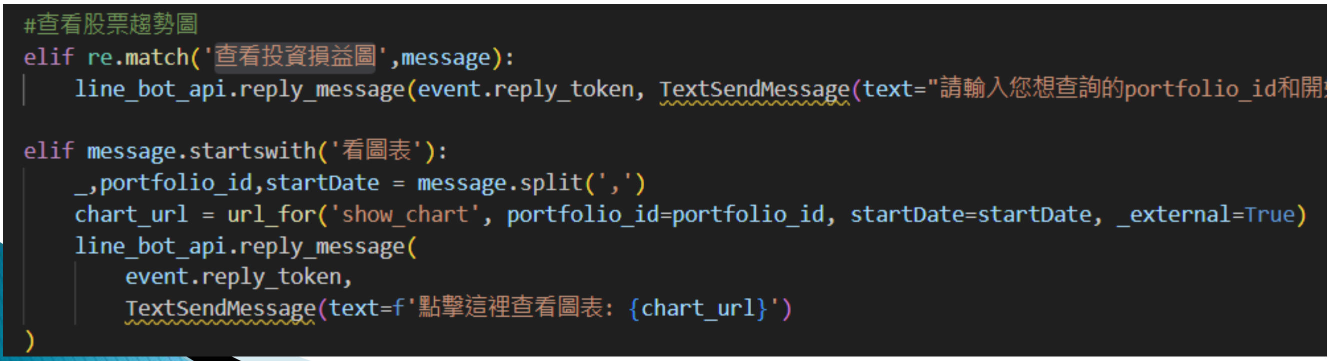
Portfilio_id : 2
Ticker_id : MMM
投資股數 : 50



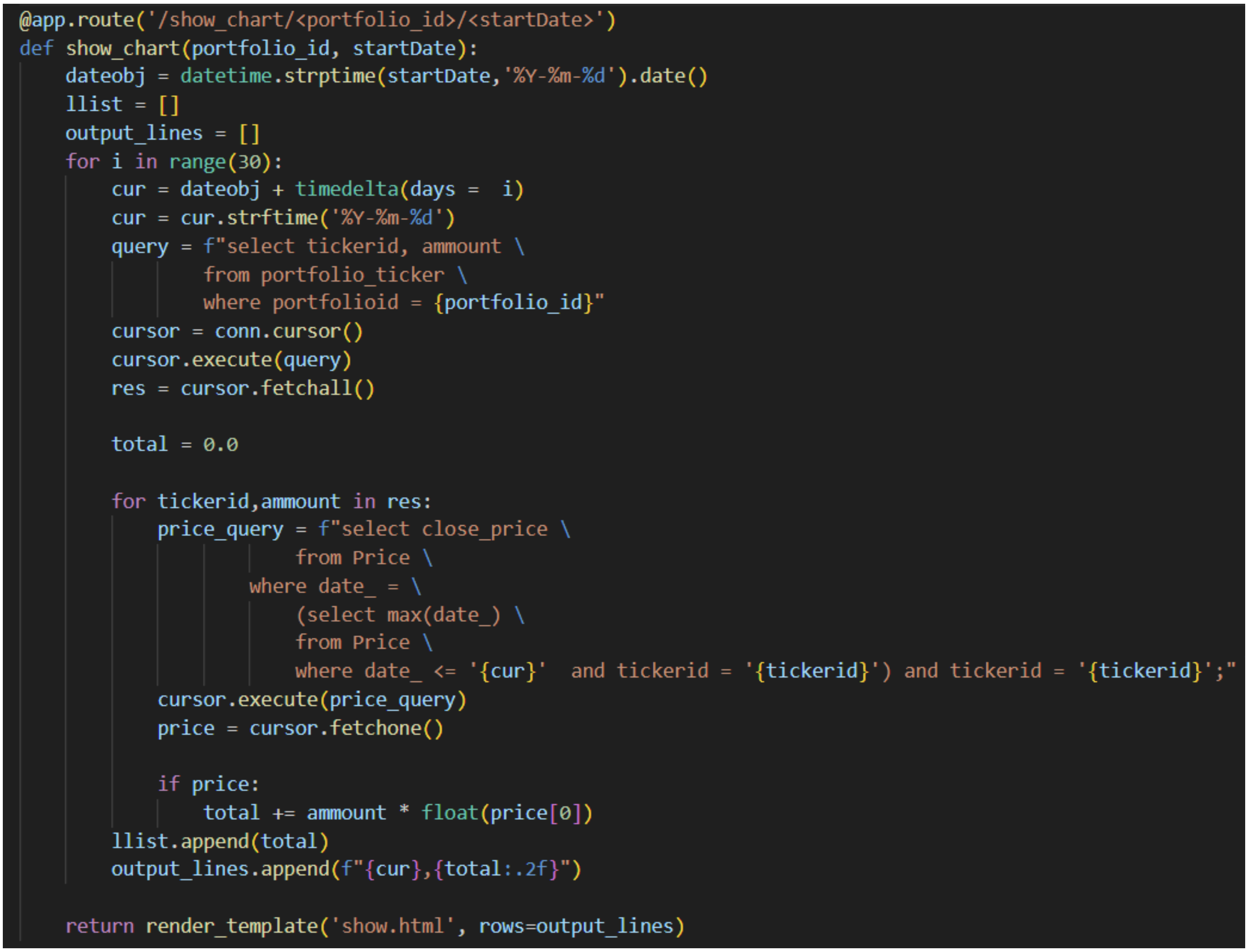
```
cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
s = "SELECT subscription_portfolio.portfolioid,portfolio_ticker.tickerid, ammount,subscription_portfolio.percentage\
        FROM subscription_portfolio LEFT JOIN portfolio_ticker \
        ON subscription_portfolio.portfolioid=portfolio_ticker.portfolioid \
            WHERE userid = %s"
```

To ensure that each table meets the Boyce-Codd Normal Form (BCNF) standards, the table contents are segmented finely. Therefore, when detailed information is required, a 'left join' approach is used to retrieve more data.

4) Viewing Investment Profit and Loss Chart:

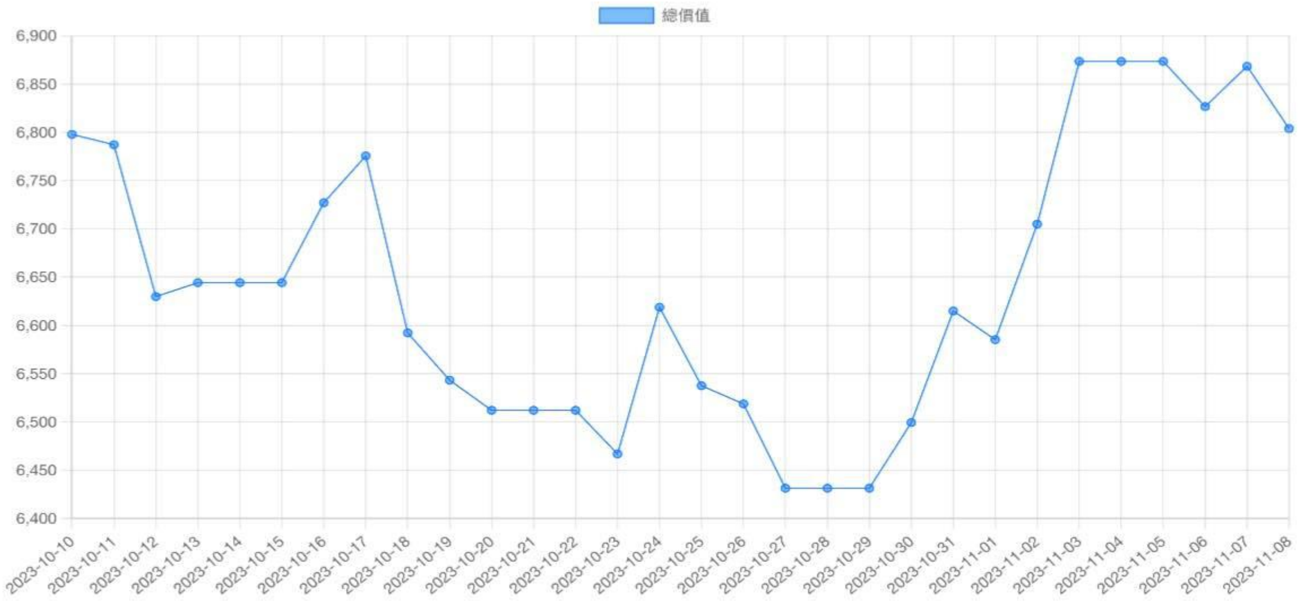


```
#查看股票趨勢圖
elif re.match('查看投資損益圖',message):
    line_bot_api.reply_message(event.reply_token, TextSendMessage(text="請輸入您想查詢的portfolio_id和開

elif message.startswith('看圖表'):
    _,portfolio_id,startDate = message.split(',')
    chart_url = url_for('show_chart', portfolio_id=portfolio_id, startDate=startDate, _external=True)
    line_bot_api.reply_message(
        event.reply_token,
        TextSendMessage(text=f'點擊這裡查看圖表: {chart_url}')
)
```

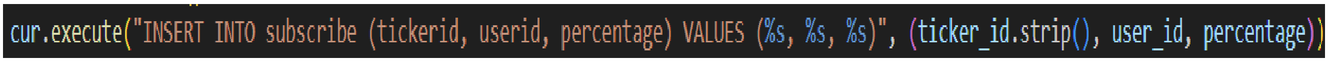


```
@app.route('/show_chart/<portfolio_id>/<startDate>')
def show_chart(portfolio_id, startDate):
    dateobj = datetime.strptime(startDate,'%Y-%m-%d').date()
    llist = []
    output_lines = []
    for i in range(30):
        cur = dateobj + timedelta(days =  i)
        cur = cur.strftime('%Y-%m-%d')
        query = f"select tickerid, ammount \
                from portfolio_ticker \
                where portfolioid = {portfolio_id}"
        cursor = conn.cursor()
        cursor.execute(query)
        res = cursor.fetchall()

        total = 0.0

        for tickerid,ammount in res:
            price_query = f"select close_price \
                        from Price \
                    where date_ = \
                        (select max(date_) \
                        from Price \
                        where date_ <= '{cur}'  and tickerid = '{tickerid}') and tickerid = '{tickerid}';"
            cursor.execute(price_query)
            price = cursor.fetchone()

            if price:
                total += ammount * float(price[0])
        llist.append(total)
        output_lines.append(f"{cur},{total:.2f}")

    return render_template('show.html', rows=output_lines)
```

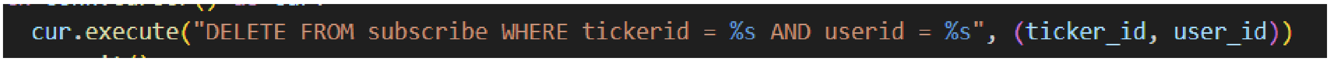
We have designed a feature that calculates the 30-day profit of an investment portfolio. However, Linebot cannot directly display the drawn charts to users. Therefore, we append the query date and id to a URL, and link it to a frontend template using the Python Flask package. The chart is then drawn using JavaScript. This allows customers to view the chart by clicking on the link.



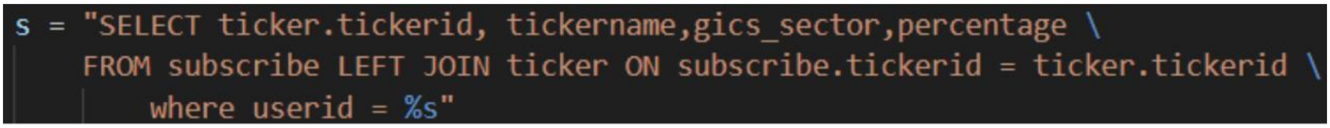
5)~7) Subscribing to Stocks, Deleting Stock Subscriptions, and Querying Subscribed Stocks:



```
cur.execute("INSERT INTO subscribe (tickerid, userid, percentage) VALUES (%s, %s, %s)", (ticker_id.strip(), user_id, percentage))
```



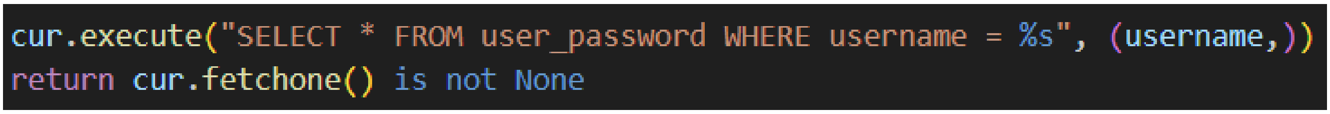
```
cur.execute("DELETE FROM subscribe WHERE tickerid = %s AND userid = %s", (ticker_id, user_id))
```



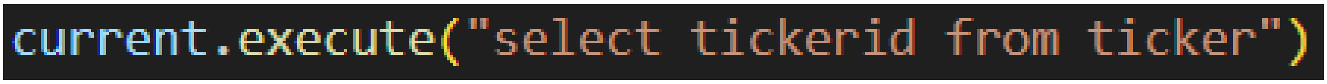
```
s = "SELECT ticker.tickerid, tickername,gics_sector,percentage \
    FROM subscribe LEFT JOIN ticker ON subscribe.tickerid = ticker.tickerid \
        where userid = %s"
```

The functionalities are similar to those used for the investment portfolio, so they won't be elaborated further. Similar designs for user dialogue reception, quick buttons, and error handling are implemented as well.

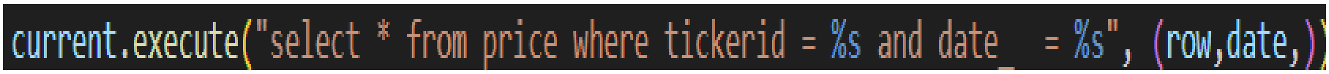
8) Automatic Update:



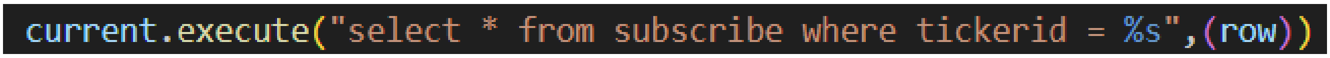
```
cur.execute("SELECT * FROM user_password WHERE username = %s", (username,))
return cur.fetchone() is not None
```



```
current.execute("select tickerid from ticker")
```



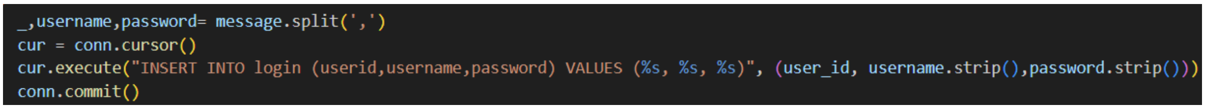
```
current.execute("select * from price where tickerid = %s and date_ = %s", (row,date,))
```



```
current.execute("select * from subscribe where tickerid = %s",(row))
```

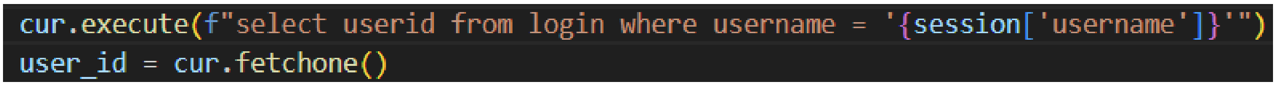
For all stocks, assess their current price fluctuations (ups and downs) and correspond this information with the subscribed list.

9) front-end register and login system



```
_,username,password= message.split(',')
cur = conn.cursor()
cur.execute("INSERT INTO login (userid,username,password) VALUES (%s, %s, %s)", (user_id, username.strip(),password.strip()))
conn.commit()
```

User can register account from Line Bot by entering username and password. Then we have a table to store line_user_id,username and password to check whether password is correct.



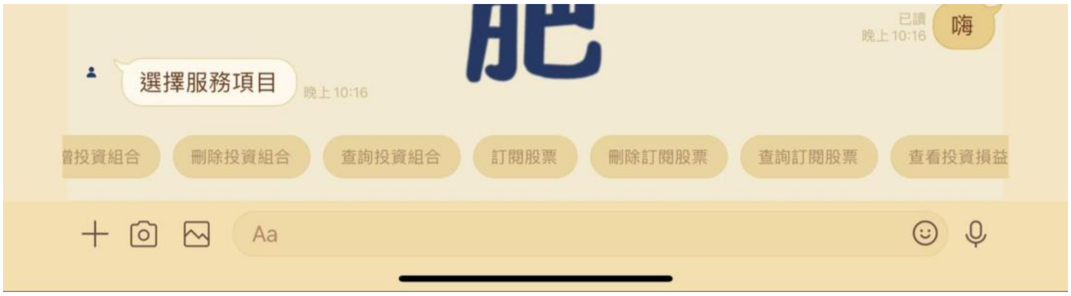
```
cur.execute(f"select userid from login where username = '{session['username']}'")
user_id = cur.fetchone()
```

When user login, we use “login” table to get the userid, which is the primary key of our table, and use the same query to show the data.

8.All the other Details

1) UI:

To design a user-friendly UI, Quick Buttons are added to many response messages, allowing users to quickly access different functions by pressing these buttons.

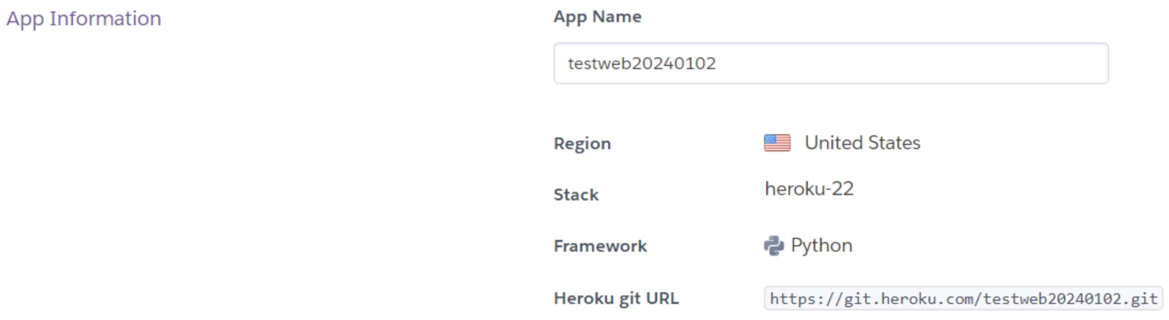


已讀
晚上10:16
嗨

選擇服務項目
晚上10:16

2) Backend Integration with Cloud Services:

Our backend uses Python‘s Flask package, with the backend program hosted on a Heroku app. This app’s URL is linked to LineBot‘s webhook for use. We also use the same back-end to creation of a continuously online frontend website by calling templates and static folders, using HTML, CSS, and JavaScript. Data display and graph visualization are all accomplished seamlessly by this application.



App Information

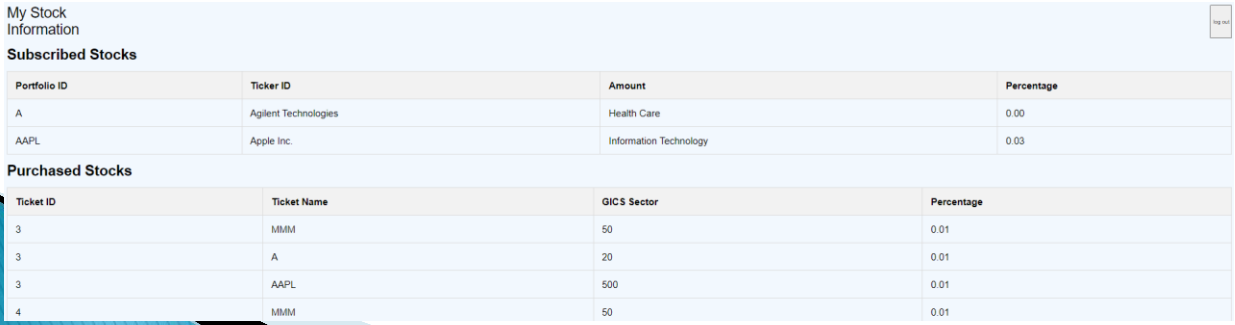
App Name: testweb20240102

Region: United States

Stack: heroku-22

Framework: Python

Heroku git URL: https://git.heroku.com/testweb20240102.git



My Stock
Information

Subscribed Stocks

Portfolio ID	Ticker ID	Amount	Percentage
A	Agilent Technologies	Health Care	0.00
AAPL	Apple Inc.	Information Technology	0.03

Purchased Stocks

Ticket ID	Ticket Name	GICS Sector	Percentage
3	MMM	50	0.01
3	A	20	0.01
3	AAPL	500	0.01
4	MMM	50	0.01

DEMO 影片連結:

https://www.youtube.com/watch?v=sWgeiSi0Nz8

Github 連結:

https://github.com/David810209/database-FinalProject/?fbclid=IwAR1jz0zLhqYvgOqKqnN17gVV158P5BjqsK0o7H-R7dD_Din1hQRHL-Fuyg8