Ringer - Virtual stock app

ECE-GY-9953-BK30
Group C :
Chia-An Kuo(Net ID: cak580)
Sai Kiran(Net ID: vc2118)

Index

A: Project Background

The stock market has always been considered a risky business, infact, some might even go to the extent of calling these investments as gambles. Our goal was to do all the heavy lifting on our side and make the user interface simple and intuitive. Ringer is a Web Application that will prove it could be as simple as possible. With access to virtual money and a starting sum of $1000, **Ringer** will enable you to build your very own stock world from scratch. Here, you get a chance to learn about how the stock market operates in the real world, find your way through profits and losses and gain a practical hand on experience of this trade business without the risk of stepping into the real stock exchange.

B: Business case

The main idea of the project is to build a trading application, where a default of $1000 dummy cash is provided to each user. There are a lot of people who will be overwhelmed after hearing “stocks”, but it is far from reality. People who understand the stock market are general people with no specialised skills. So, our aim is to build a simple and easy to use stock application, with no fancy terminologies/requirements, that one should know beforehand to start using the application. Our goal was to do all the heavy lifting on our side and make the user interface simple and intuitive. The idea of this application came when we felt a lot of hurdles while using the “Robinhood” application. A lot of people want to start trading but don’t know where and how to start. It has always been a playground of the rich or risk takers for the most part, but fear no more. **Ringer** (one that enters a competition under false representations) tries to solve this by providing fake currency to invest in and, see your portfolio as if you have invested it in the real world and gain hands on experience.This is a virtual stock trading web application made using React. This application intends to help future investors learn the ups and downs of the stock market and build a portfolio of stocks for themselves. Beginning with a sum of $1000 you get a chance to get accustomed to the buying and selling of stocks, track your progress over a period of time and make yourself ready to conquer the stock market.

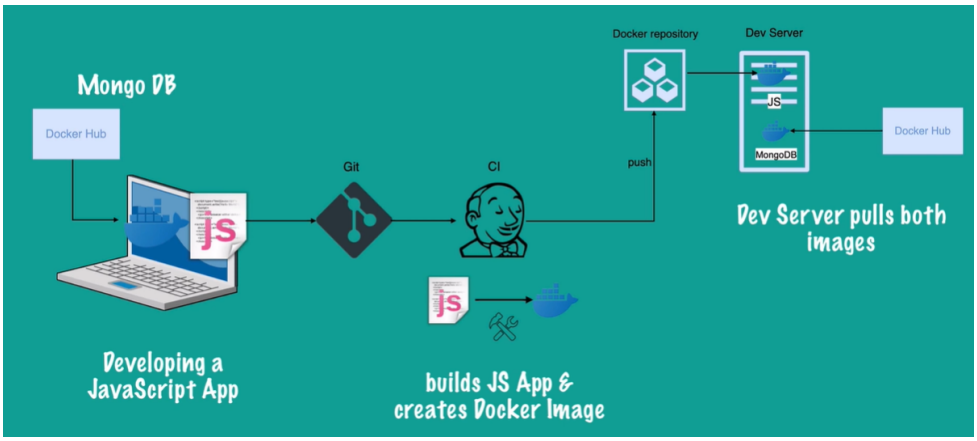
C: Project Objectives

This is a virtual stock trading web application made using React.js hook. This application intends to help future investors learn the ups and downs of the stock market and build a portfolio of stocks for themselves. Beginning with a sum of $1000 you get a chance to get accustomed to the buying and selling of stocks, track your progress over a period of time and make yourself ready to conquer the stock market. The app will be containerized, and could be deployed on kubernetes. We are using jenkins for the CI/ CD pipeline. Whenever the code is modified, the app will deploy the latest version automatically.

D: Project Scope:

We did the following to help reach our goals.

- Understanding of Stock Markets.
- Gathered Stock Prices from IEX API.
- Create OLTP models.
- Host it in RDS.
- Create a Web front end.
- Implement backend in Django.
- Implement CRUD using REST.
- Implement DW using Amazon.
- Develop CICD pipeline for Web.
- BI using Tableau.



E: Product Scope:

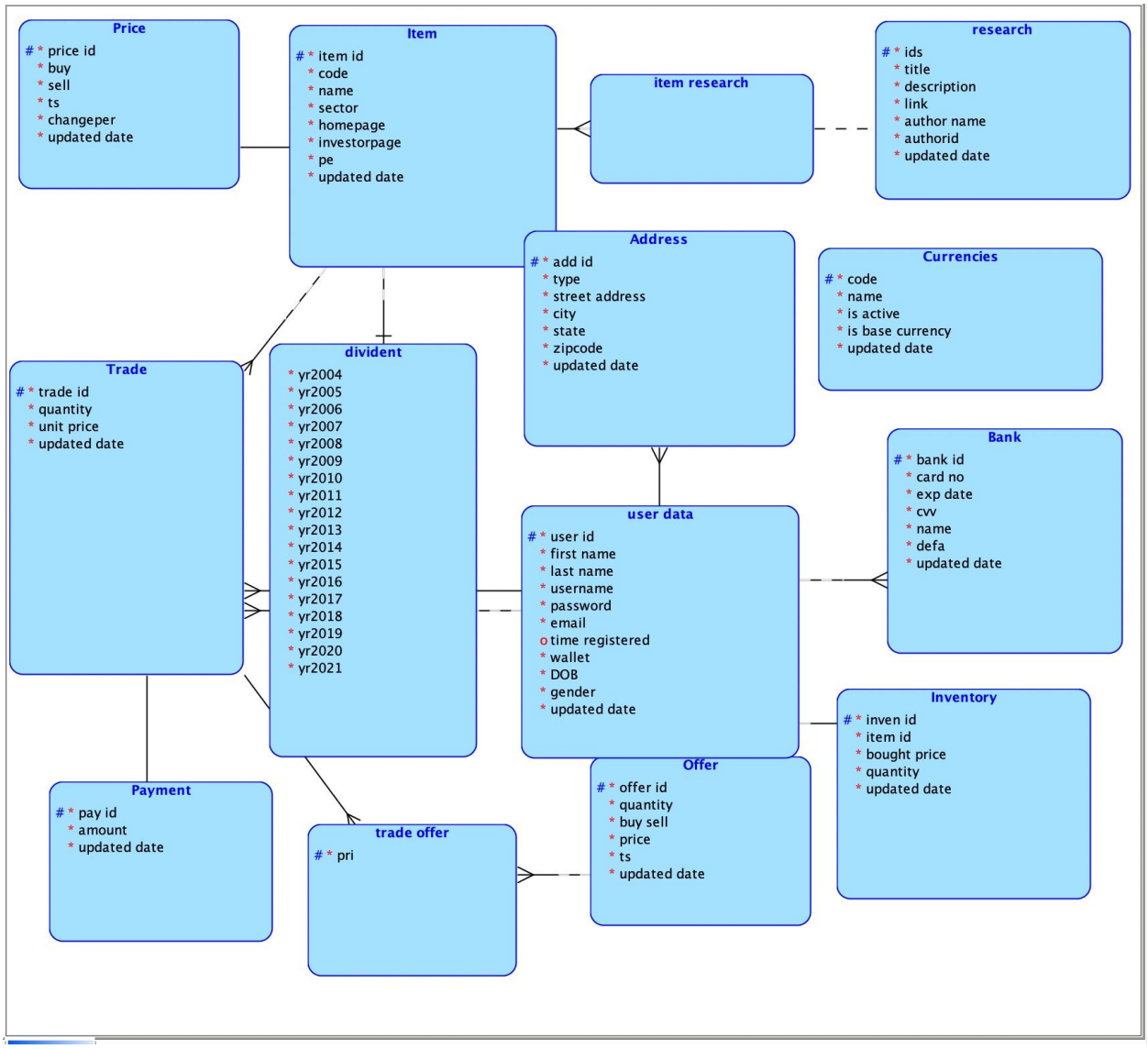
- OLTP
    - Design OLTP.
    - Host in RDS.
    - Implement CRUD.
    - Use IEX API to extract stock prices.
- Web Application
    - Design a Login and registration page.
    - Use Firebase to authenticate users.
    - Design a research and stock page.
    - Implement buy/sell logic.
    - Design History and Dashboard pages.
    - Use CICD using Docker, Jenkins and K8s.
- Data Warehouse
    - Used to extract data from OLTP to OLAP.
    - Copy the CSV to S3.
    - Import data from S3 to Redshift Stage tables.
    - Run Procedure to load data from stage table to DW tables.
- Analysis
    - Connect Redshift to Tableau.
    - Use DBeaver to inspect the tables.

F: Project Milestones

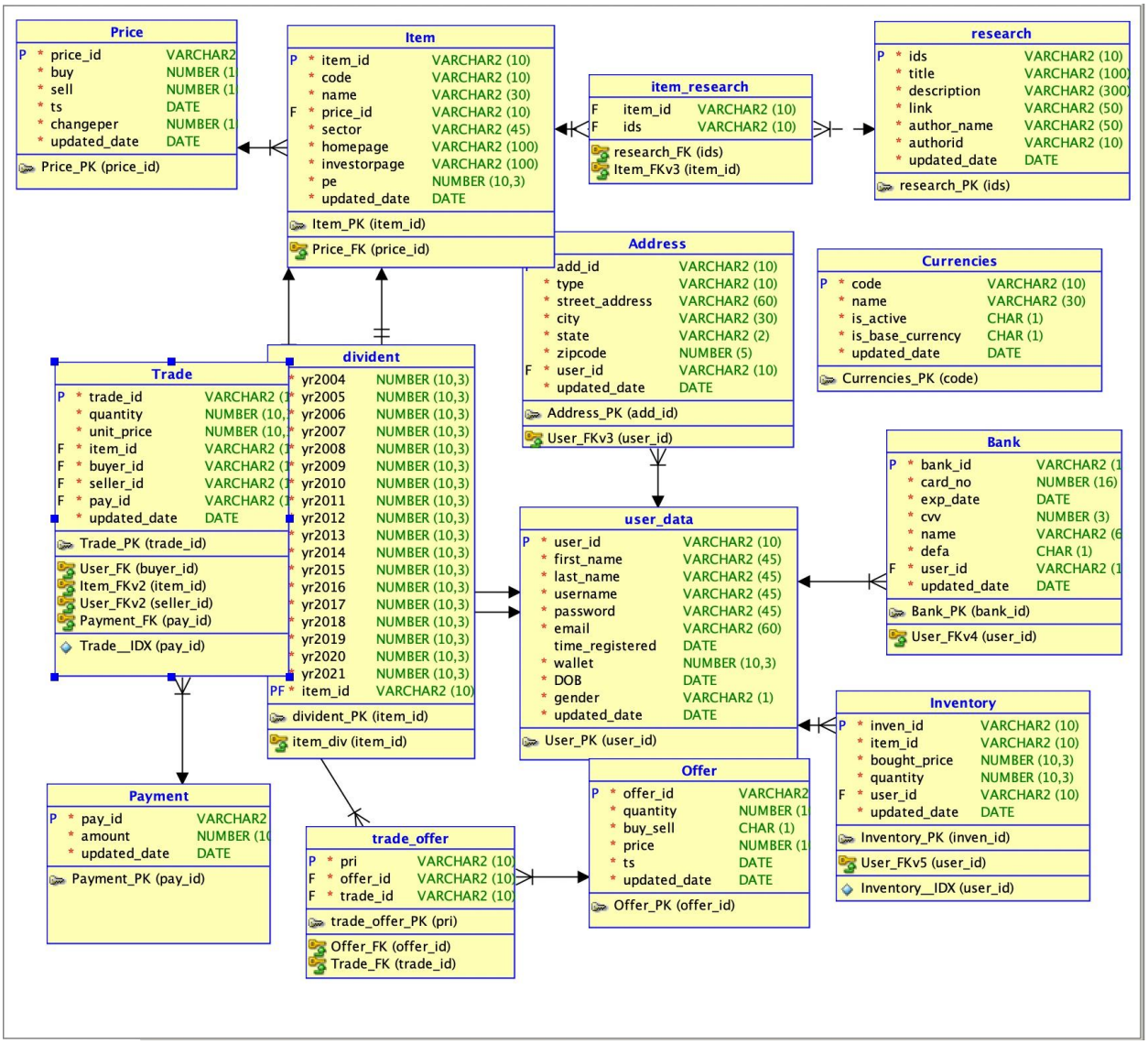
Milestone	**Time to complete(**1 day = 2 hrs**)**
Finalise OLTP Design	2 Weeks
Understand and Implement Stock API	4 days
Design OLTP	2 days
Login, Signup	1 Week
Firebase Integration	2 days
Dashboard, Research, Dividends	2 Weeks
Profile, buy/sell, history,dashboard	2 Weeks

Demo data insertion	1 Week
Django integration	3 days
Database CRUD	1 Week
DW Design	2 days
ETL, Redshift, External Tables, Tableau	2 Weeks
Integration of all, RDS, S3	1 Week
CICD-Docker,k8s,Jenkins	2 Weeks

G. Logical Model (OLTP)


Price
* price id
* buy
* sell
* ts
* changeper
* updated date
Item
* item id
* code
* name
* sector
* homepage
* investorpage
* pe
* updated date
item research
research
* ids
* title
* description
* link
* author name
* authorid
* updated date
Address
* add id
* type
* street address
* city
* state
* zipcode
* updated date
Currencies
* code
* name
* is active
* is base currency
* updated date
Trade
* trade id
* quantity
* unit price
* updated date
divident
* yr2004
* yr2005
* yr2006
* yr2007
* yr2008
* yr2009
* yr2010
* yr2011
* yr2012
* yr2013
* yr2014
* yr2015
* yr2016
* yr2017
* yr2018
* yr2019
* yr2020
* yr2021
user data
* user id
* first name
* last name
* username
* password
* email
o time registered
* wallet
* DOB
* gender
* updated date
Bank
* bank id
* card no
* exp date
* cvv
* name
* defa
* updated date
Inventory
* inven id
* item id
* bought price
* quantity
* updated date
Payment
* pay id
* amount
* updated date
trade offer
* pri
Offer
* offer id
* quantity
* buy sell
* price
* ts
* updated date

H. Relational Model (OLTP)



I. Assumptions and Constraints

- Users can have one or more banks and addresses.
- Buyer_id and Seller_id are user_ids of buyer and seller from the user_data table.
- Single buy/sell can happen in multiple trades and offers until the deal.
- Hence, single buy/sell can have multiple payments.

- Transactions only happen in a particular currency if “is_active” is true.
- Single item can have multiple researches and, similarly, a research can belong to multiple stocks.
- Prices of stocks varry, not the code or company name, etc, hence split and store them in the “Price” table.
- Data is inserted into Offer, once a trade is found, so it's a many to many relationship.
- All primary keys are varchar with length of 10.
- Updated_date is added in every table for ETL.
- Zip Code should be a 5 digit number, and state 2 Characters.

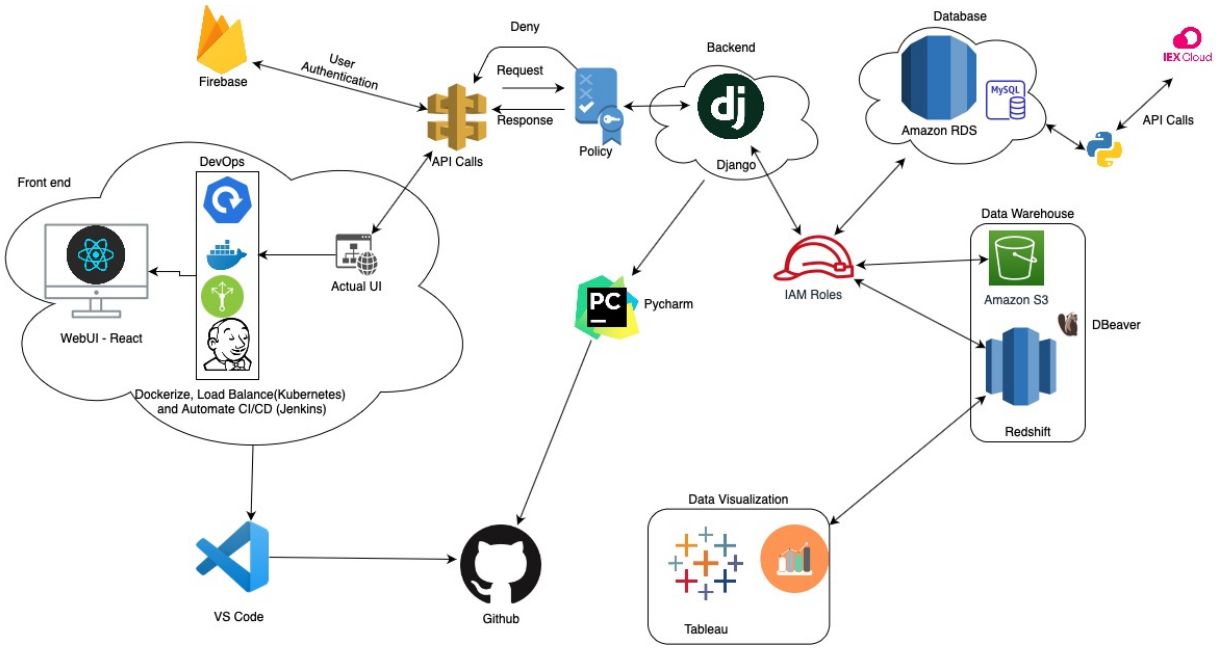
J. Infrastructure

OLTP : MySQL
mysql Ver 8.0.22 for Linux on x86_64 (MySQL Community Server - GPL)

DW: Amazon Redshift
Reporting Tool: Tableau
Front end: React.js, JavaScript, HTML, CSS

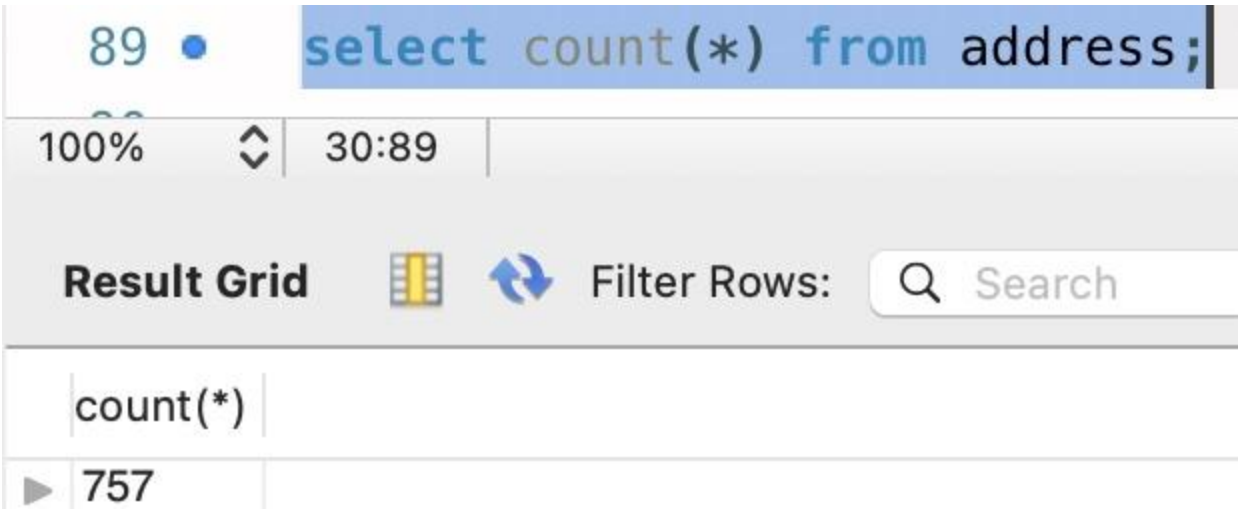
Back end: Django
Database: Amazon RDS - MySQL

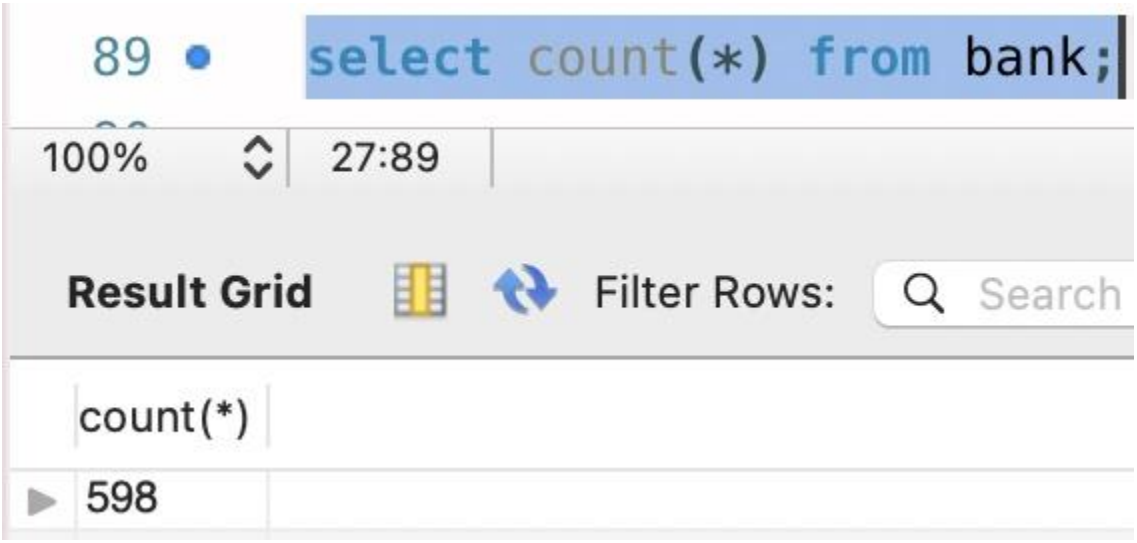
Other tools : Amazon S3, Lamda, API gateway, Firebase, Django, Docker, Jenkins, Kubernetes, Python, GIT, JDBC, Pycharm, Visual Studio, IEX services.

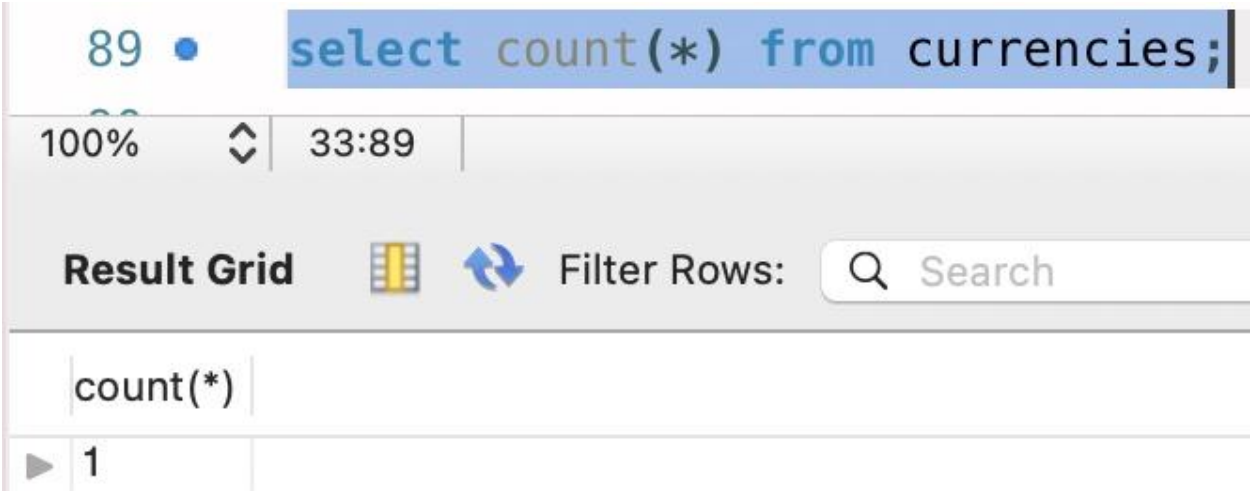


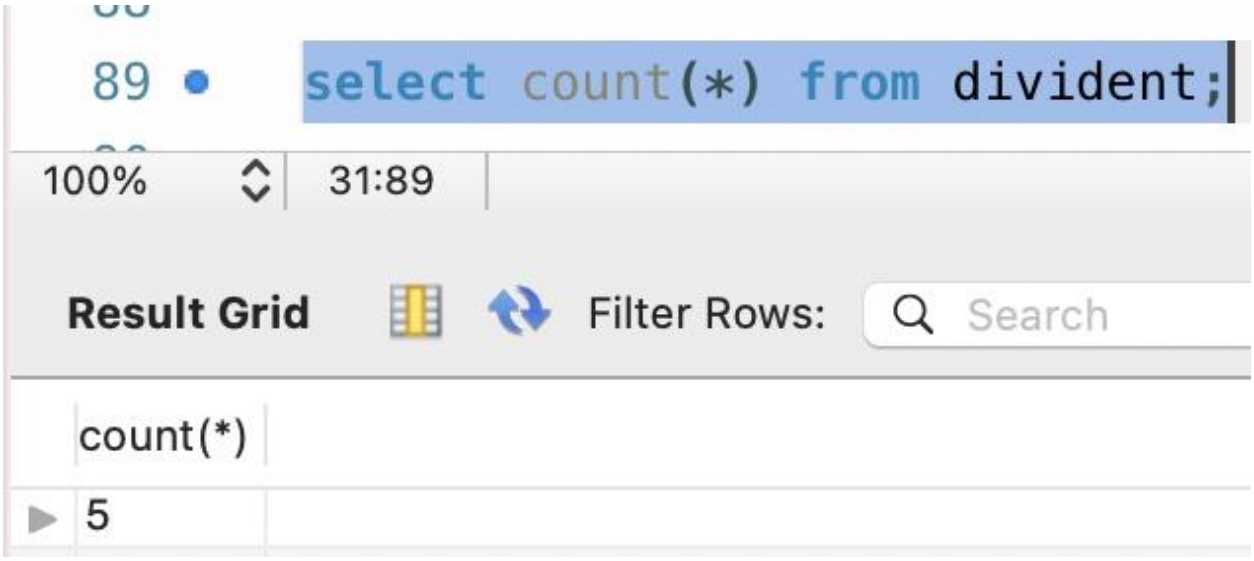
System Design

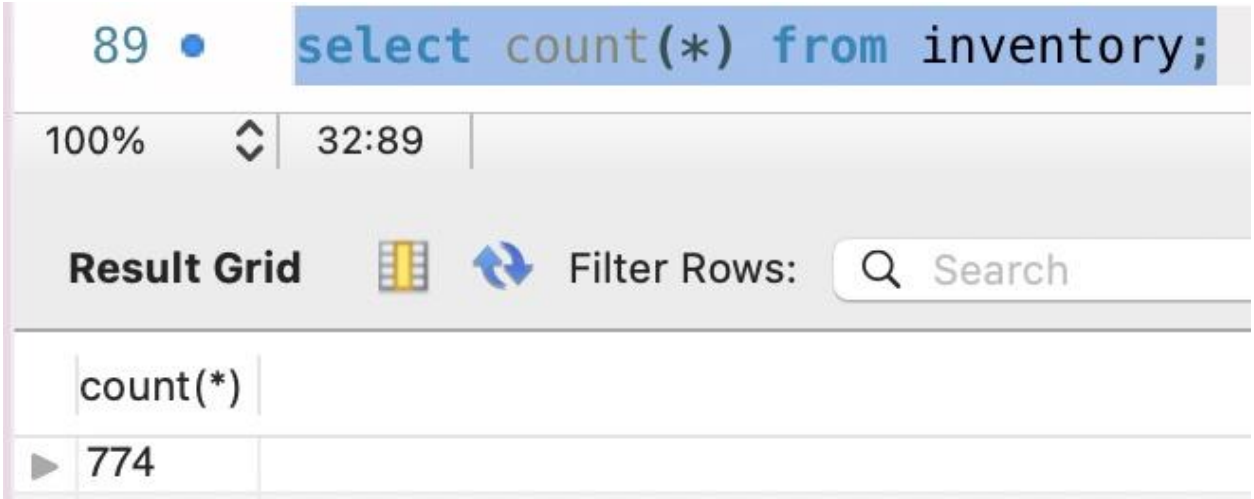
K. Record counts for each OLTP tables

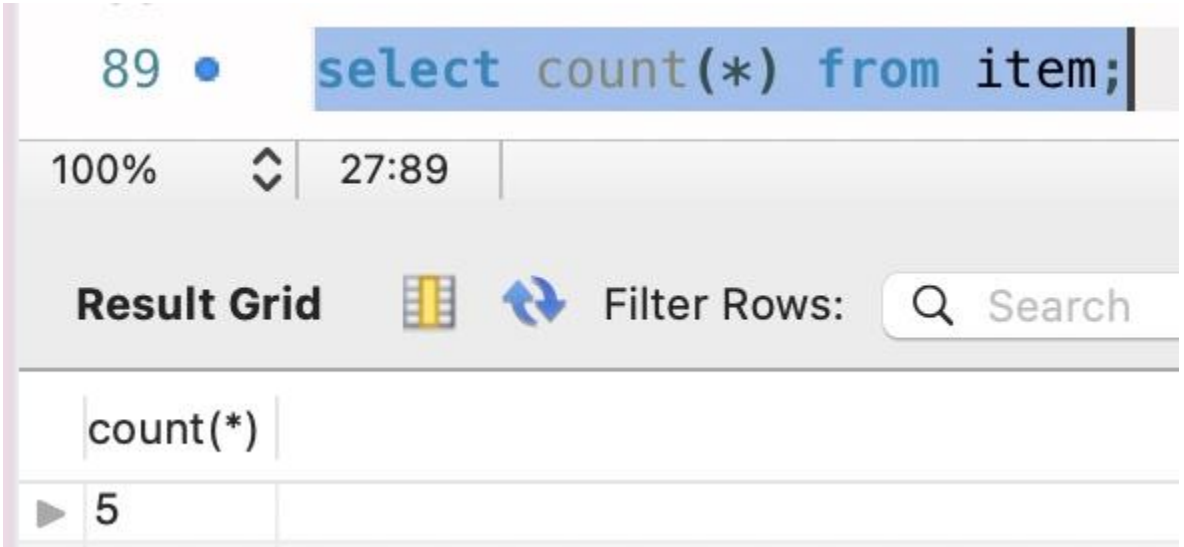

89 select count(*) from address;
100% 30:89
Result Grid Filter Rows: Search
count(*)
757

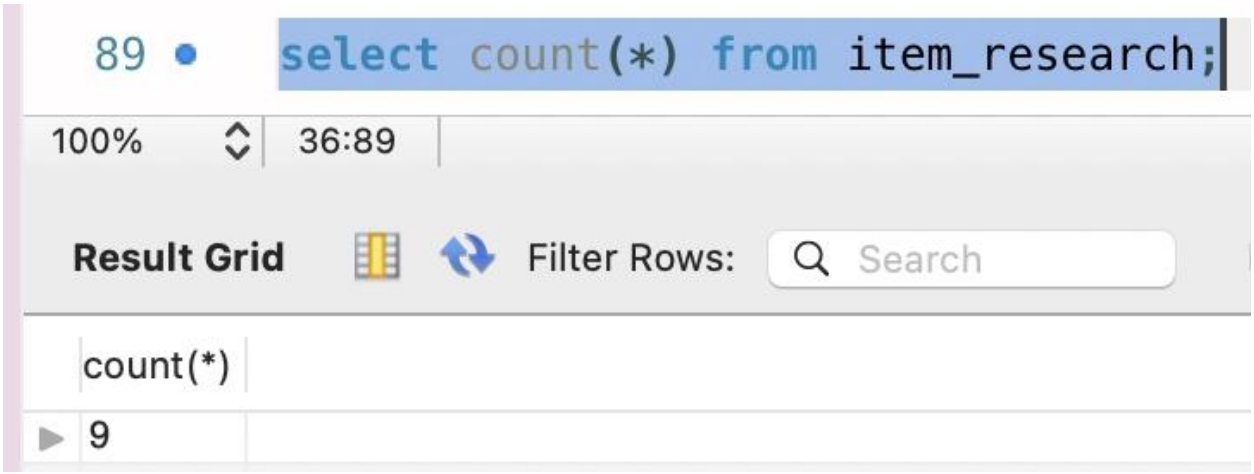

89 select count(*) from bank;
100% 27:89
Result Grid Filter Rows: Search
count(*)
598

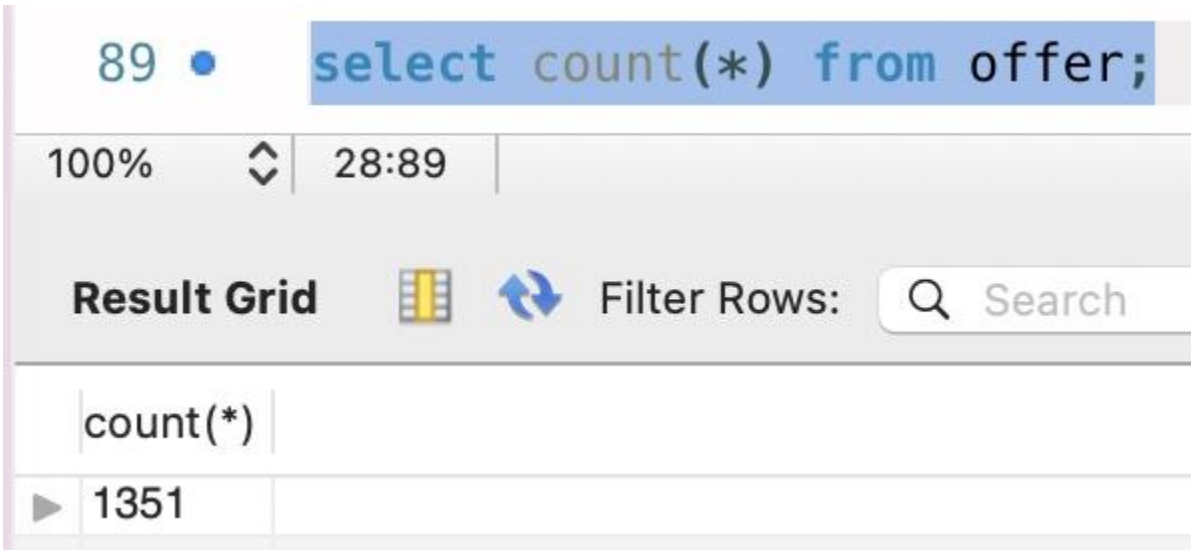

89 select count(*) from currencies;
100% 33:89
Result Grid Filter Rows: Search
count(*)
1

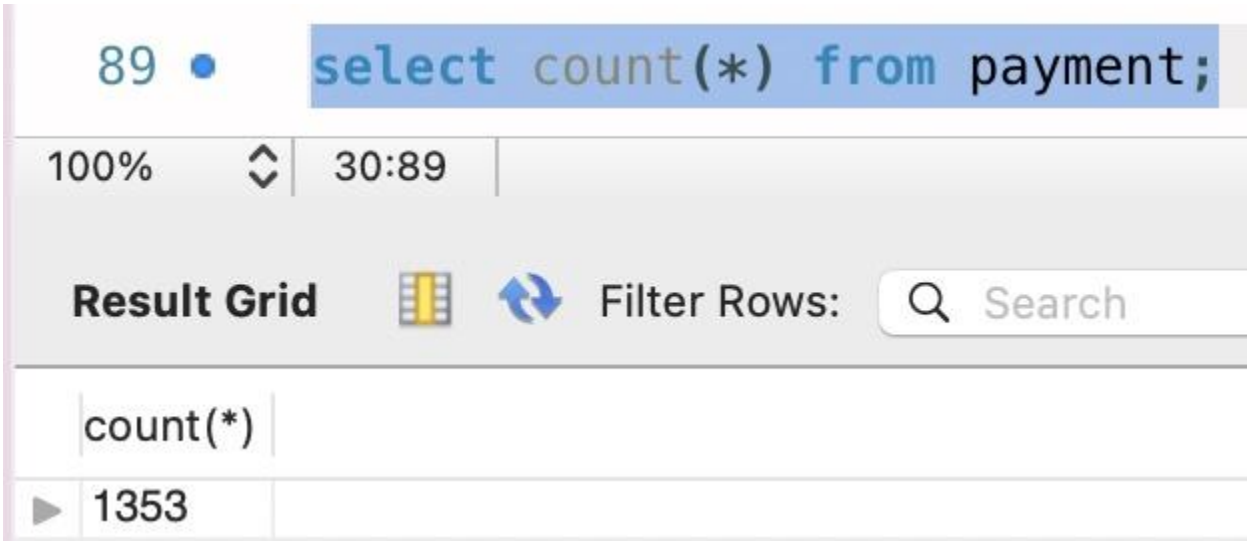

89 select count(*) from divident;
100% 31:89
Result Grid Filter Rows: Search
count(*)
5

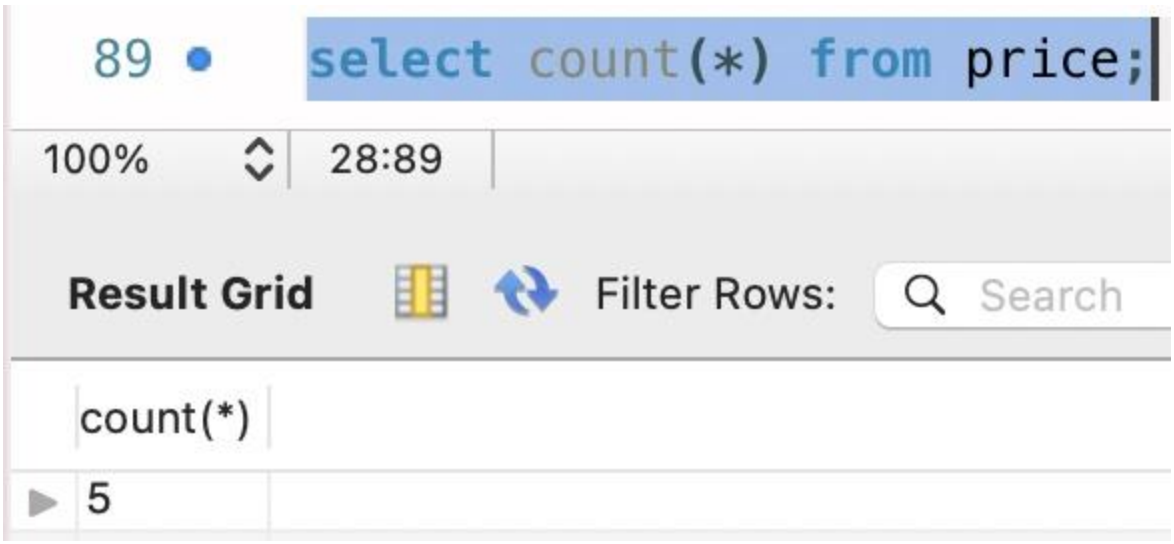

89 select count(*) from inventory;
100% 32:89
Result Grid Filter Rows: Search
count(*)
774

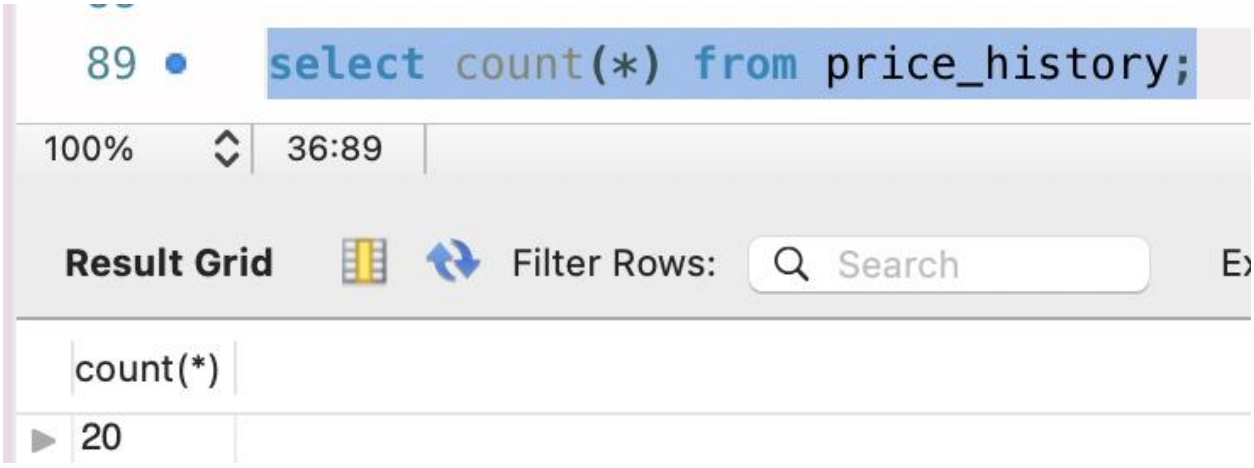

89 select count(*) from item;
100% 27:89
Result Grid Filter Rows: Search
count(*)
5

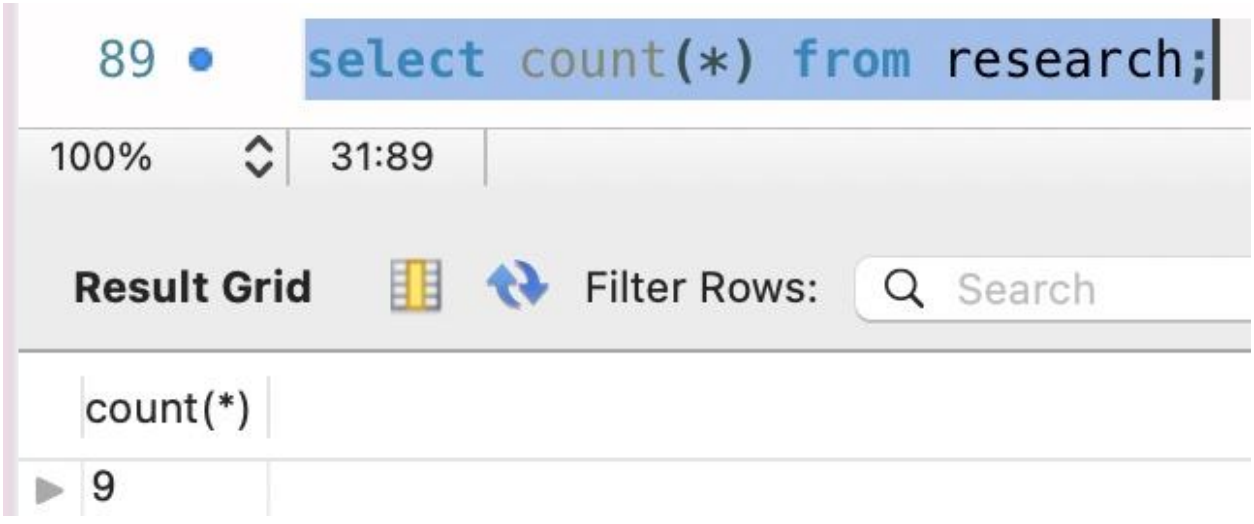

89 select count(*) from item_research;
100% 36:89
Result Grid Filter Rows: Search
count(*)
9

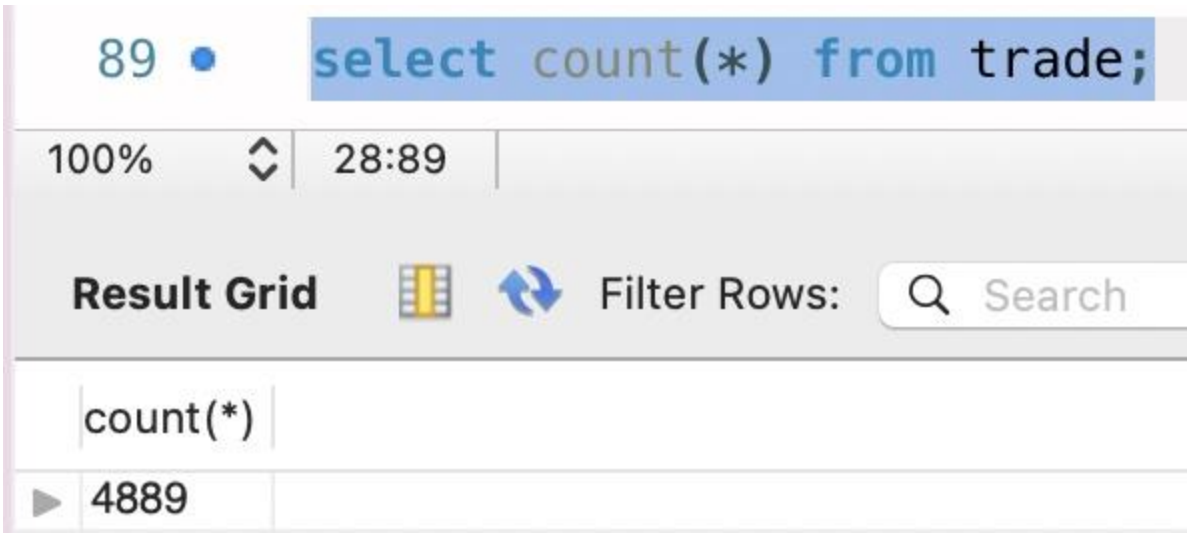

89 select count(*) from offer;
100% 28:89
Result Grid Filter Rows: Search
count(*)
1351

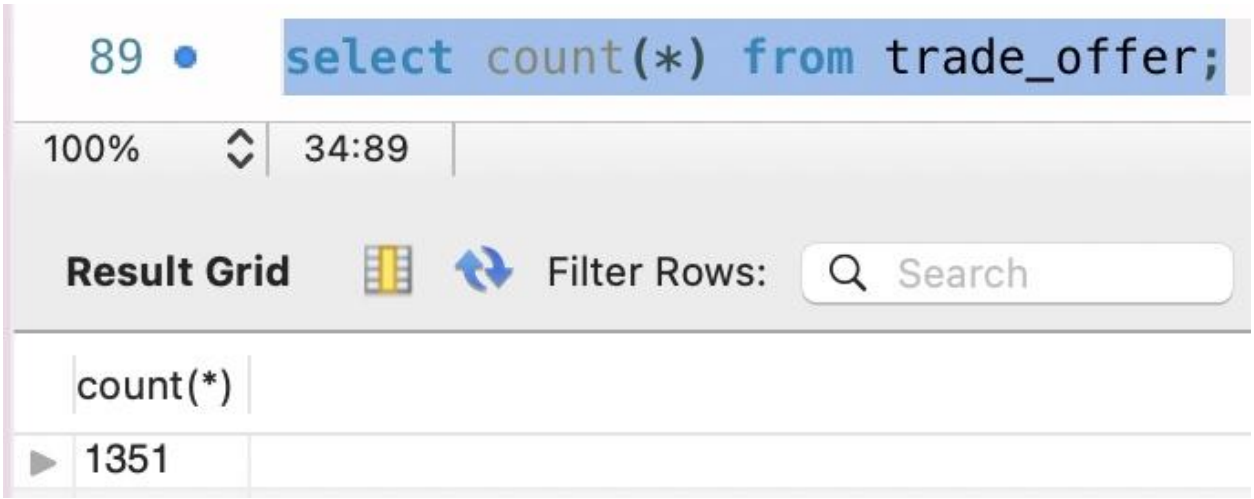

89 select count(*) from payment;
100% 30:89
Result Grid Filter Rows: Search
count(*)
1353

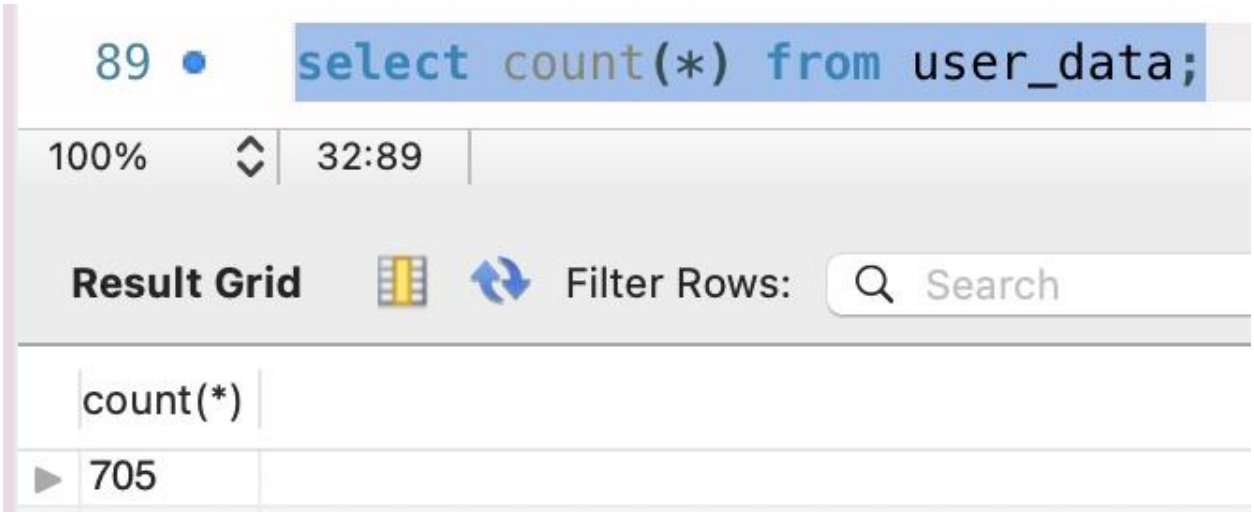

89 select count(*) from price;
100% 28:89
Result Grid Filter Rows: Search
count(*)
5


89 select count(*) from price_history;
100% 36:89
Result Grid Filter Rows: Search
count(*)
20


89 select count(*) from research;
100% 31:89
Result Grid Filter Rows: Search
count(*)
9


89 select count(*) from trade;
100%
28:89
Result Grid
Filter Rows:
Search
count(*)
4889


89 select count(*) from trade_offer;
100%
34:89
Result Grid
Filter Rows:
Search
count(*)
1351


89 select count(*) from user_data;
100%
32:89
Result Grid
Filter Rows:
Search
count(*)
705

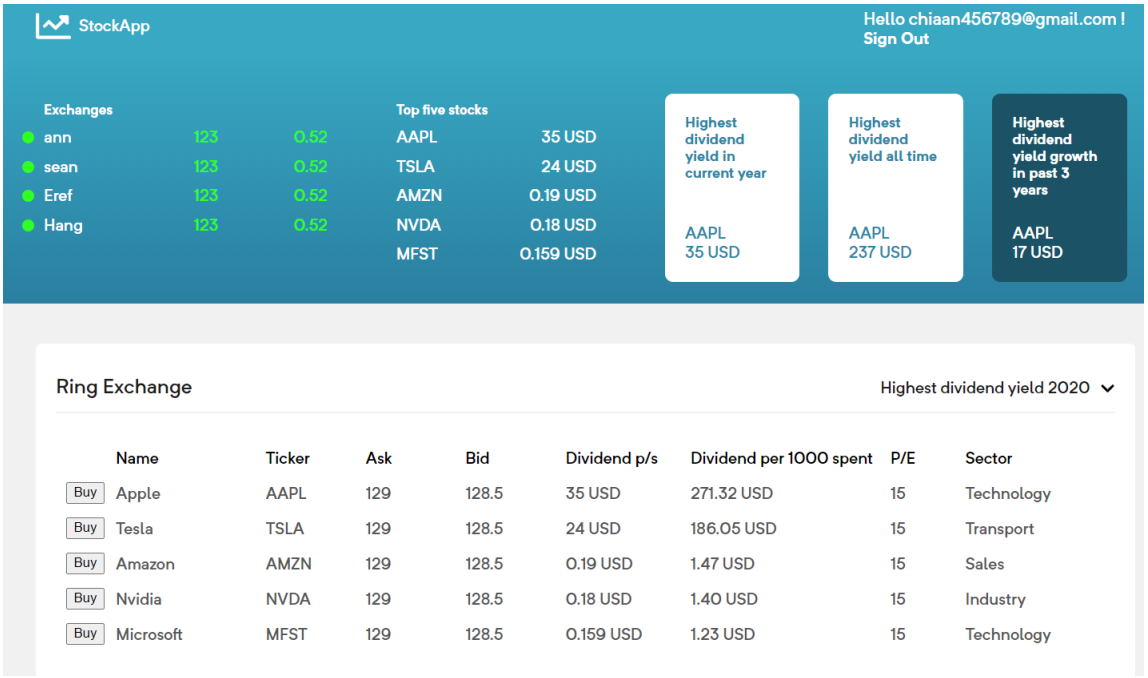
L. Web application summary

Following can be done through web UI,

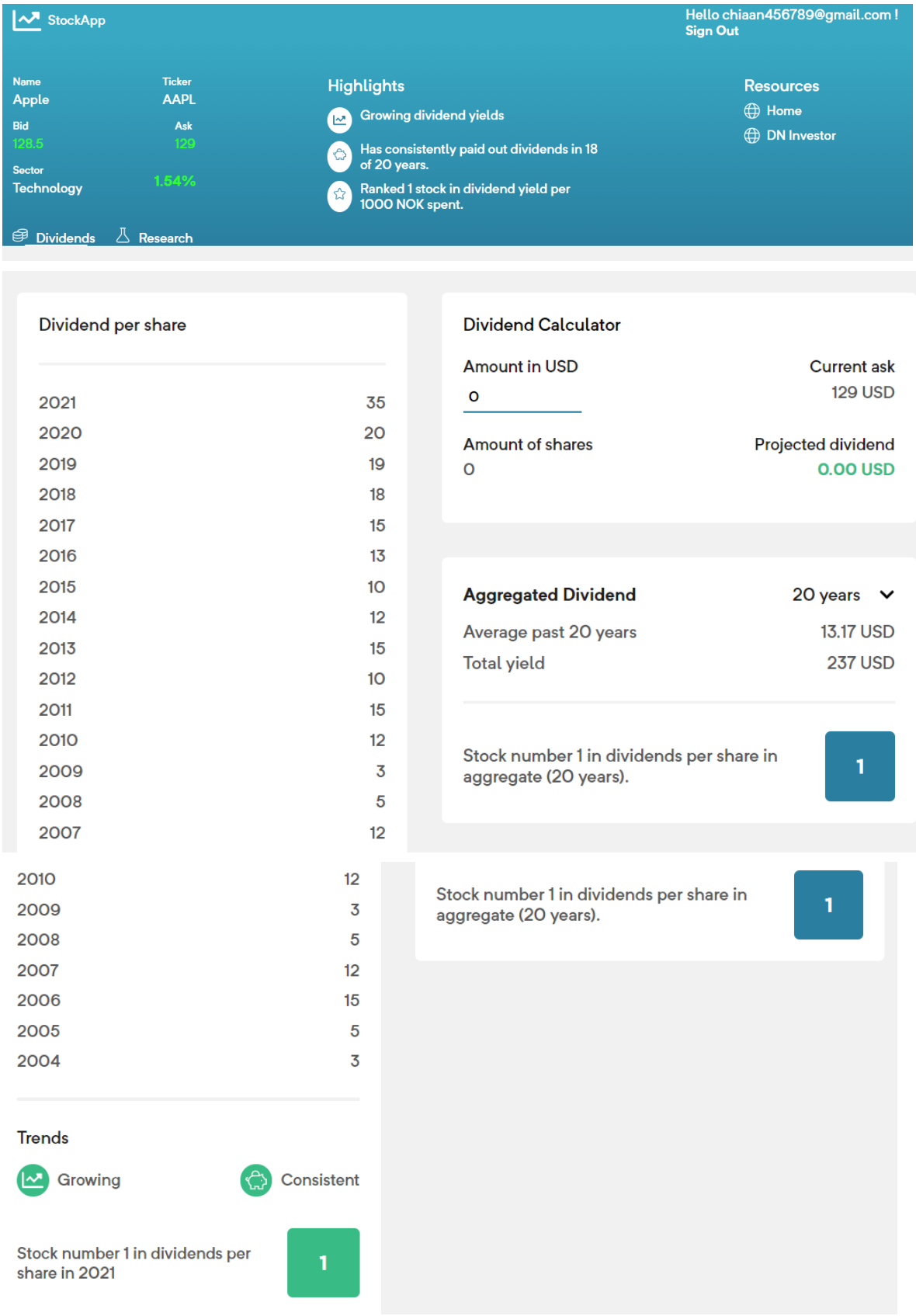
- Create account
- Login/Logout
- View/Edit Profile details.
- Recharge Wallet Balance.
- Buy/Sell Stock.
- See user Portfolio
- See Transaction history
- Add research to a stock.
- Obtain latest stock prices using IEX API, and update price and its history.
- Calculate Dividends.
- Buy/sell stocks according to highest returns.
- Token based user authentication
- User specific access control, using django backend.

M: Few screenshots of the web applications

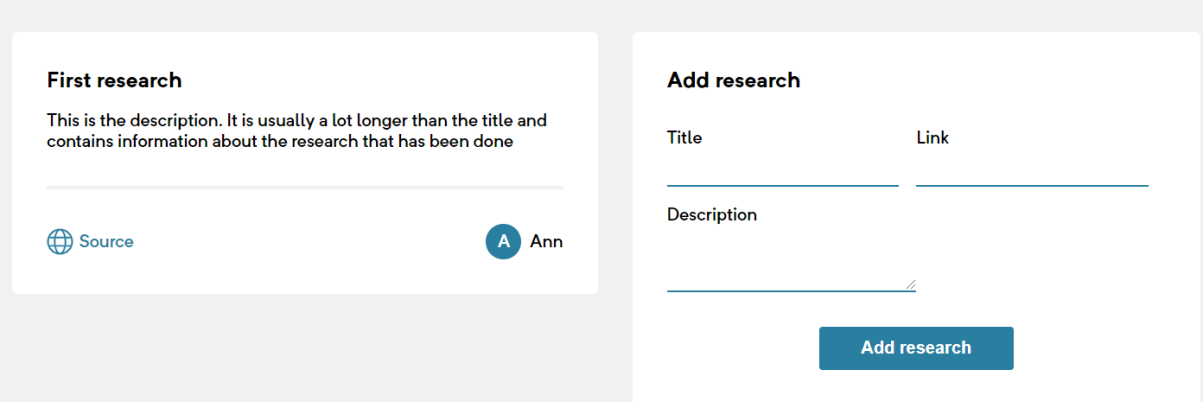
Users could view all the stocks, and use the filter we created to view the stocks in different ways.



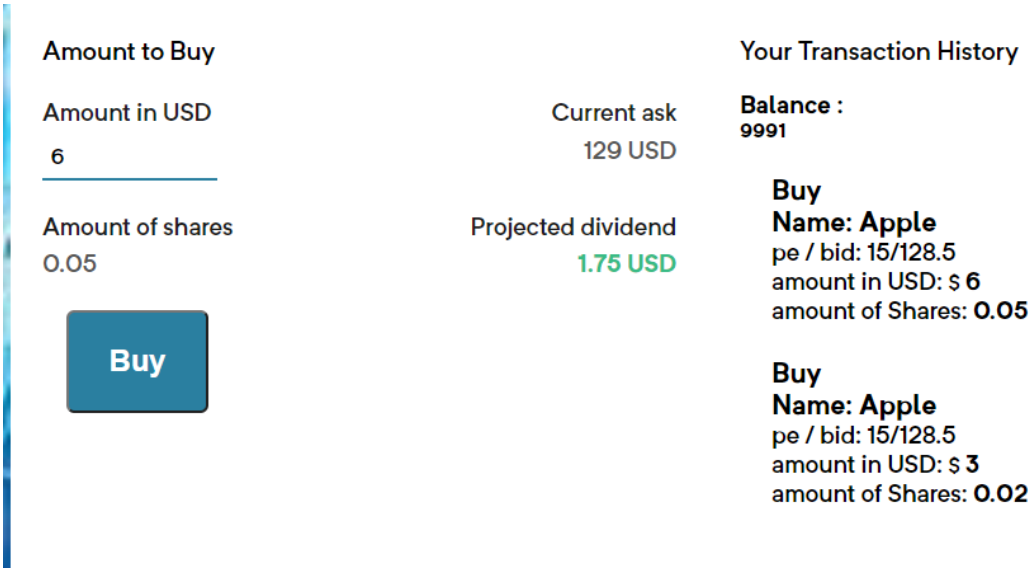
Users could click on specific stock and view details.


StockApp
Hello chiaan456789@gmail.com !
Sign Out
Name
Apple
Ticker
AAPL
Bid
128.5
Ask
129
Sector
Technology
1.54%
Dividends
Research
Highlights
Growing dividend yields
Has consistently paid out dividends in 18 of 20 years.
Ranked 1 stock in dividend yield per 1000 NOK spent.
Resources
Home
DN Investor
Dividend per share
2021 35
2020 20
2019 19
2018 18
2017 15
2016 13
2015 10
2014 12
2013 15
2012 10
2011 15
2010 12
2009 3
2008 5
2007 12
Dividend Calculator
Amount in USD
0
Current ask
129 USD
Amount of shares
0
Projected dividend
0.00 USD
Aggregated Dividend
20 years
Average past 20 years
13.17 USD
Total yield
237 USD
Stock number 1 in dividends per share in aggregate (20 years).
1
2010 12
2009 3
2008 5
2007 12
2006 15
2005 5
2004 3
Trends
Growing
Consistent
Stock number 1 in dividends per share in 2021
1
Stock number 1 in dividends per share in aggregate (20 years).
1

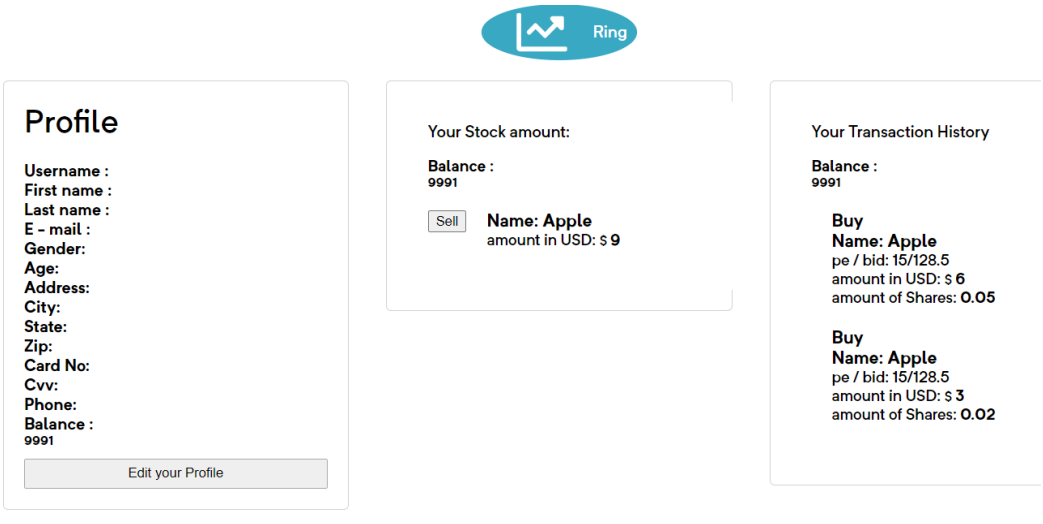
This is the research page for each stock.



User could buy the stock, and the transaction history would be shown on the right.



This is the profile page, users can sell stocks and edit profiles here.



The edit profile page:

Edit Profile

Username :

First name :

Last name :

E - mail :
chiaan0426@gmail.com

Gender:

Age:

Address:

City:

State:

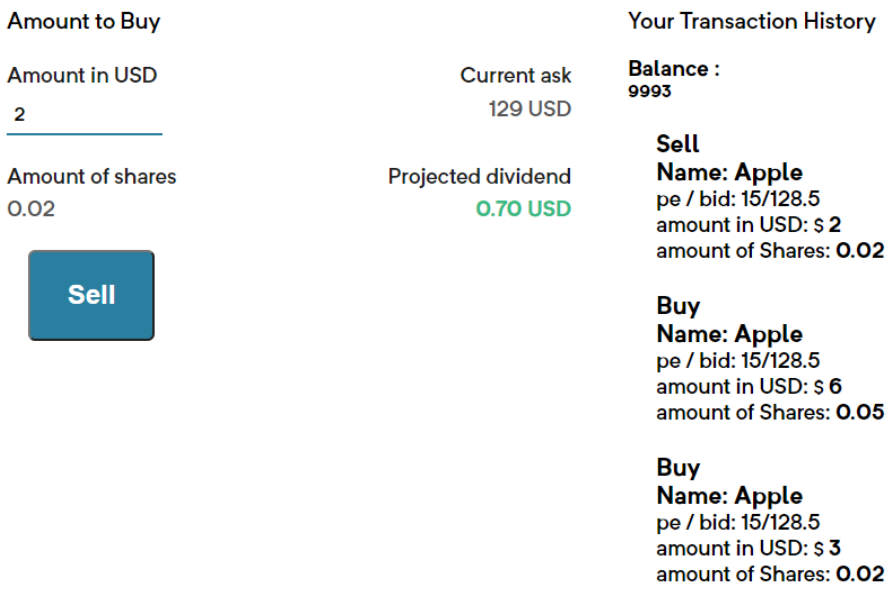
Zipcode:

Card No:

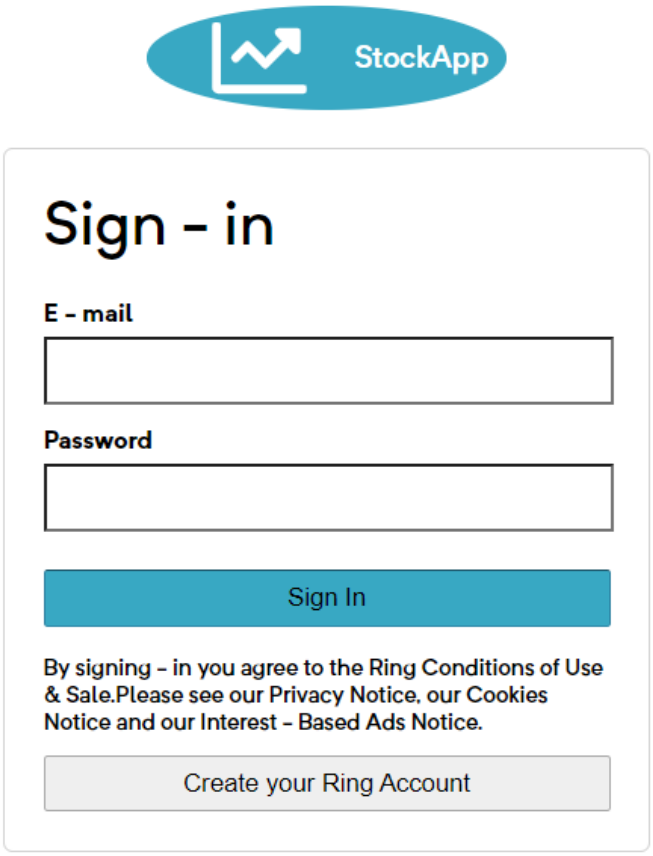
Exp Date:

Cvv:

Phone:



Login page:

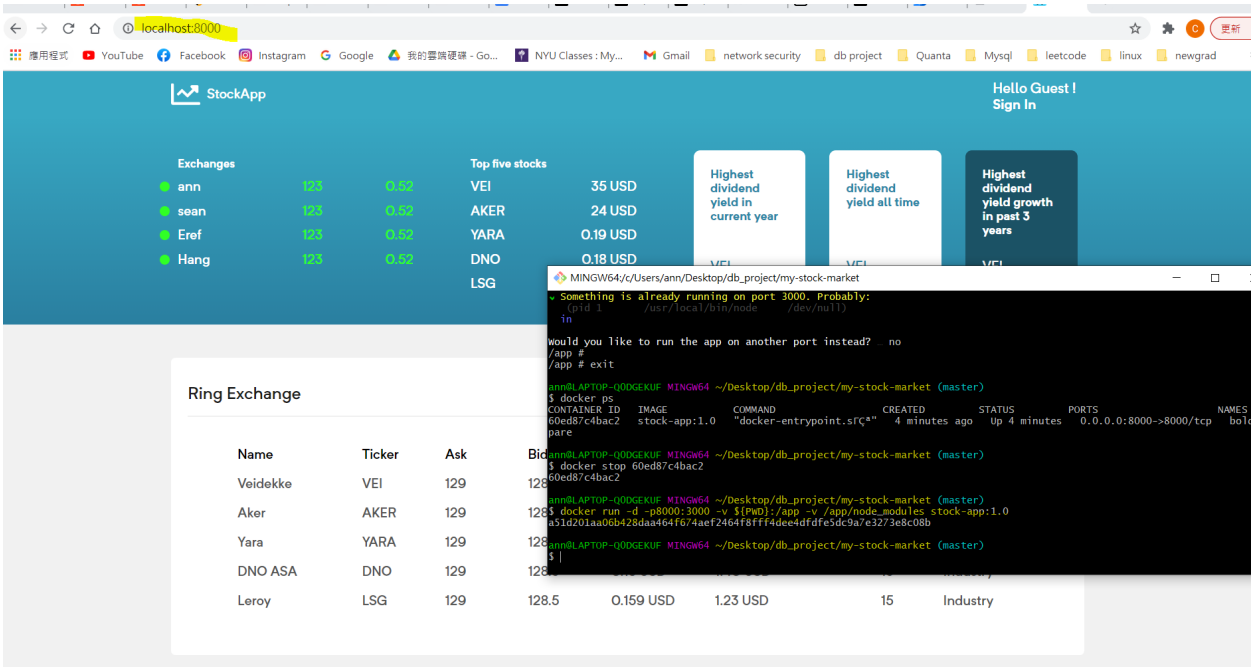


CI/CD with Jenkins pipeline & React Nodejs into K8S:
1.Dockerfile:

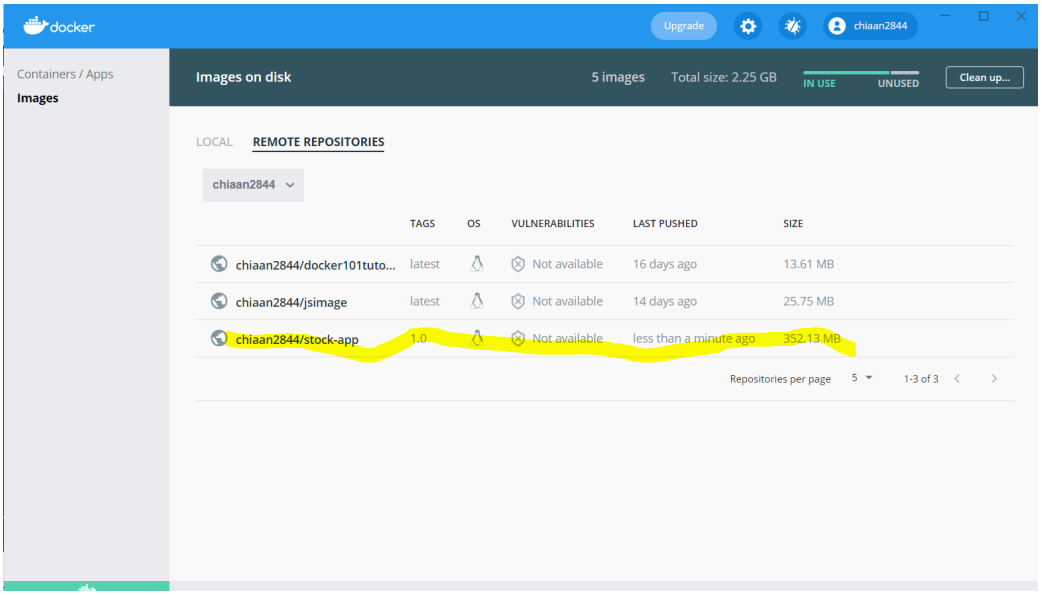
```dockerfile
# pull official base image
FROM node:13.12.0-alpine

# set working directory
WORKDIR /app

# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# install app dependencies
COPY package.json ./
COPY package-lock.json ./
RUN npm install --silent
RUN npm install react-scripts@3.4.1 -g --silent
RUN npm install react-router-dom --silent
RUN npm install classnames --silent
RUN npm install react-util-kit --silent
RUN npm install --save firebase

# add app
COPY . ./

# start app
CMD ["npm", "start"]
```

```
ann@LAPTOP-Q0DGEKUF MINGW64 ~/Desktop/db_project/my-stock-market (master)
$  docker images
REPOSITORY          TAG       IMAGE ID       CREATED              SIZE
stock-app           1.0       d006757bf49d   About a minute ago   1.1GB
mongo               latest    30b3be246e39   5 days ago           449MB
mongo-express       latest    e5a1f58bcef1   6 days ago           128MB
jenkins/jenkins     lts       d457516b229f   6 days ago           571MB

ann@LAPTOP-Q0DGEKUF MINGW64 ~/Desktop/db_project/my-stock-market (master)
$
```

Then verify if the app can really run successfully in the container.



2. Push the image to a private cloud image registry.

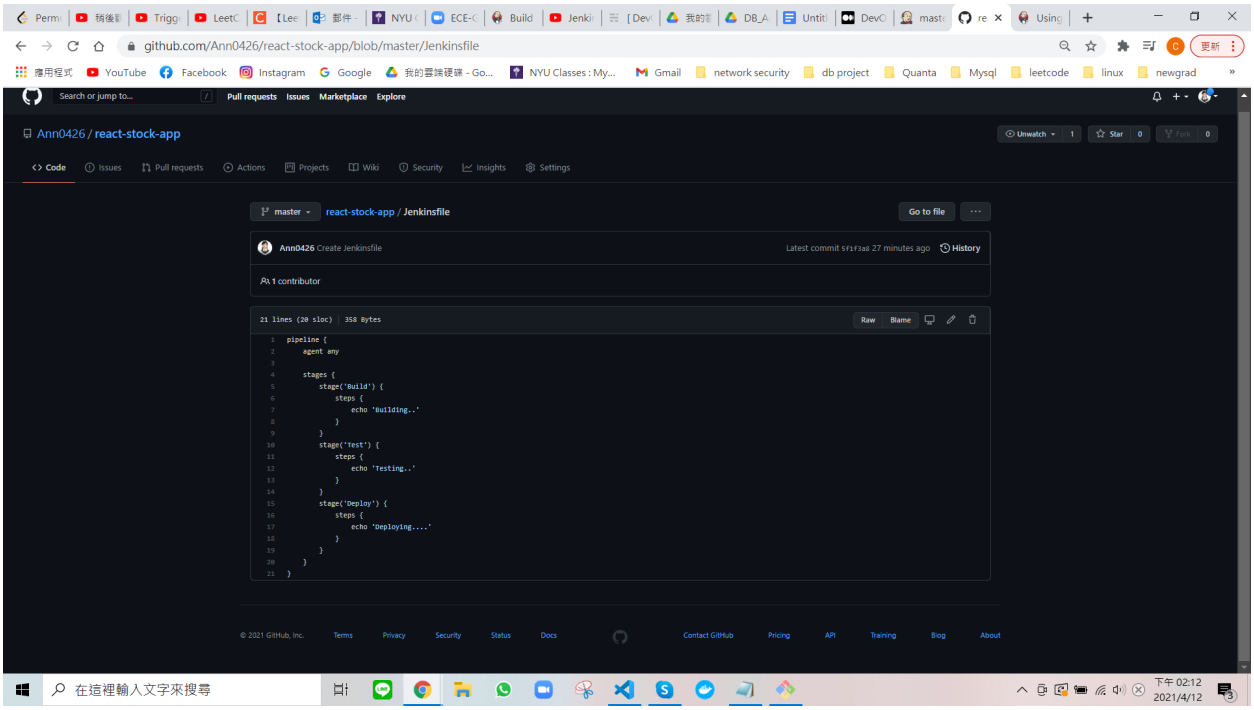


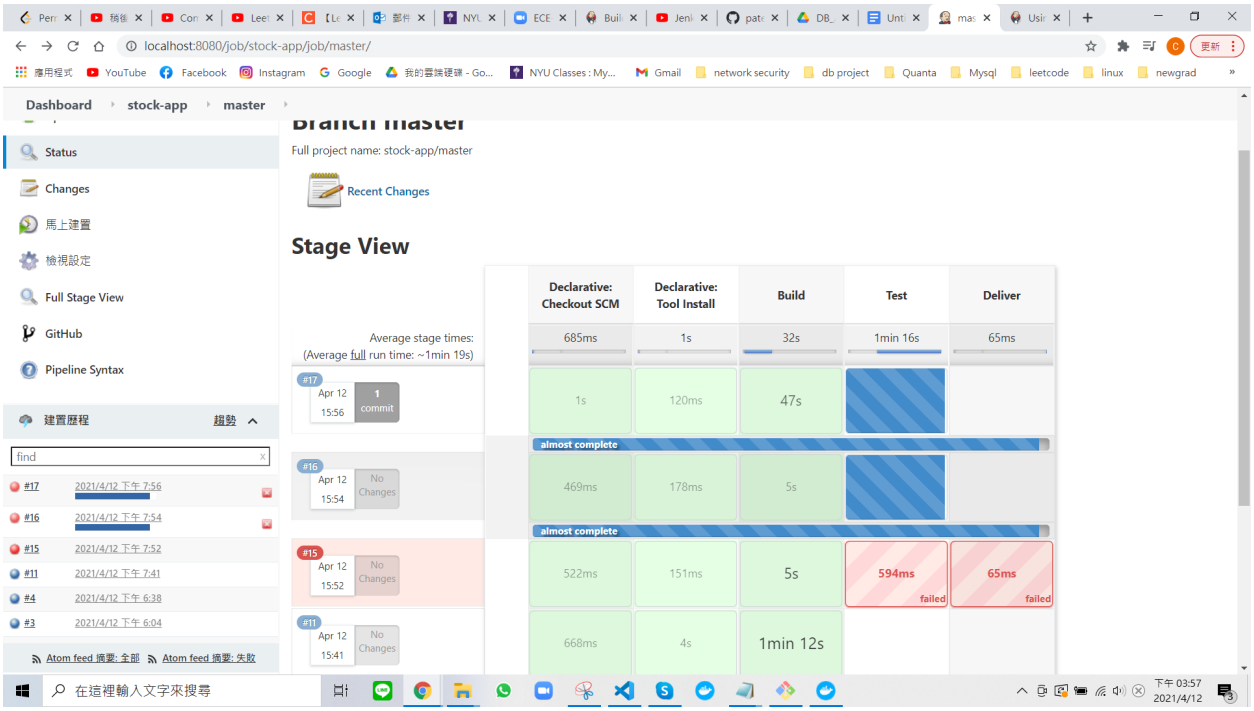
3. Create a deployment file and deploy with k8s.
Specify the replicas, service and image:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stock-app
  labels:
    app: stock-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: stock-app
  template:
    metadata:
      labels:
        app: stock-app
    spec:
      containers:
      - name: stock-app
        image: chiaan2844/stock-app
        ports:
        - containerPort: 8000
---
apiVersion: v1
kind: Service
metadata:
  name: stock-app-service
  labels:
    app: stock-app
spec:
  type: NodePort
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 8000
    nodePort: 32121
  selector:
    app: stock-app
```

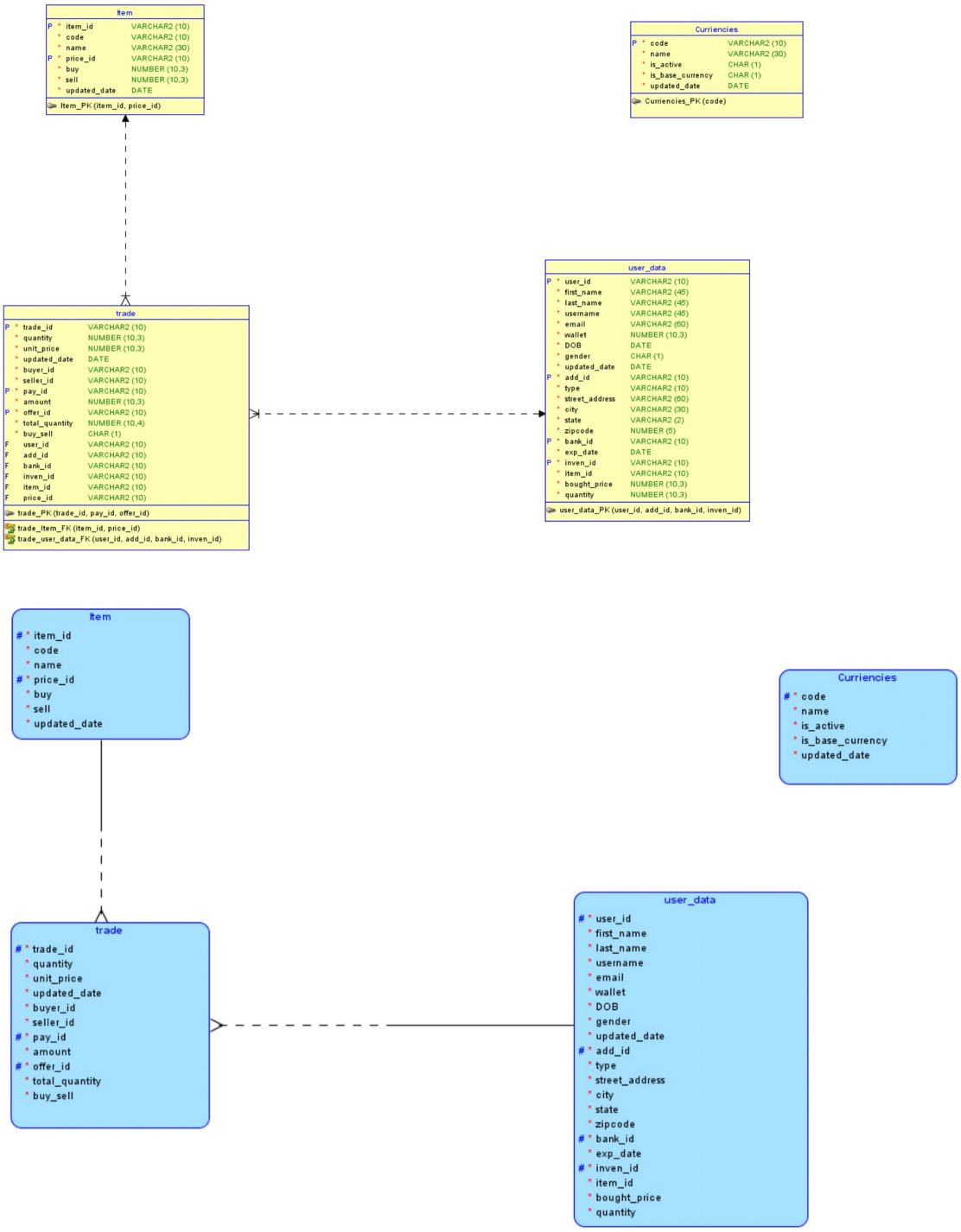
```
ann@LAPTOP-QODGEKUF MINGW64 ~/Desktop/db_project/my-stock-market (master)
$ kubectl get deployment
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
stock-app   0/2     2            0           4m20s

ann@LAPTOP-QODGEKUF MINGW64 ~/Desktop/db_project/my-stock-market (master)
$ kubectl get pods
NAME                         READY   STATUS             RESTARTS   AGE
stock-app-c78d849cb-6qw99    0/1     ImagePullBackOff   0          4m26s
stock-app-c78d849cb-lcfcm    0/1     ImagePullBackOff   0          4m26s

ann@LAPTOP-QODGEKUF MINGW64 ~/Desktop/db_project/my-stock-market (master)
$ kubectl get service
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP          5m49s
stock-app-service   NodePort    10.99.96.186    <none>        8000:32121/TCP   4m31s

ann@LAPTOP-QODGEKUF MINGW64 ~/Desktop/db_project/my-stock-market (master)
$ kubectl get nodes
NAME       STATUS   ROLES                  AGE    VERSION
minikube   Ready    control-plane,master   6m3s   v1.20.2
```

The Ci/CD pipeline set up with Jenkins:




localhost:8080/job/stock-app/job/master/
Dashboard
stock-app
master
Status
Changes
馬上建置
檢視設定
Full Stage View
GitHub
Pipeline Syntax
建置歷程
趨勢
Full project name: stock-app/master
Recent Changes
Stage View
Declarative: Checkout SCM
Declarative: Tool Install
Build
Test
Deliver
Average stage times:
(Average full run time: ~1min 19s)
685ms
1s
32s
1min 16s
65ms
#17
Apr 12
15:56
1 commit
1s
120ms
47s
almost complete
#16
Apr 12
15:54
No Changes
469ms
178ms
5s
almost complete
#15
Apr 12
15:52
No Changes
522ms
151ms
5s
594ms
failed
65ms
failed
#11
Apr 12
15:41
No Changes
668ms
4s
1min 12s
Atom feed 摘要: 全部
Atom feed 摘要: 失敗
在這裡輸入文字來搜尋

N. DW logical and Relational model



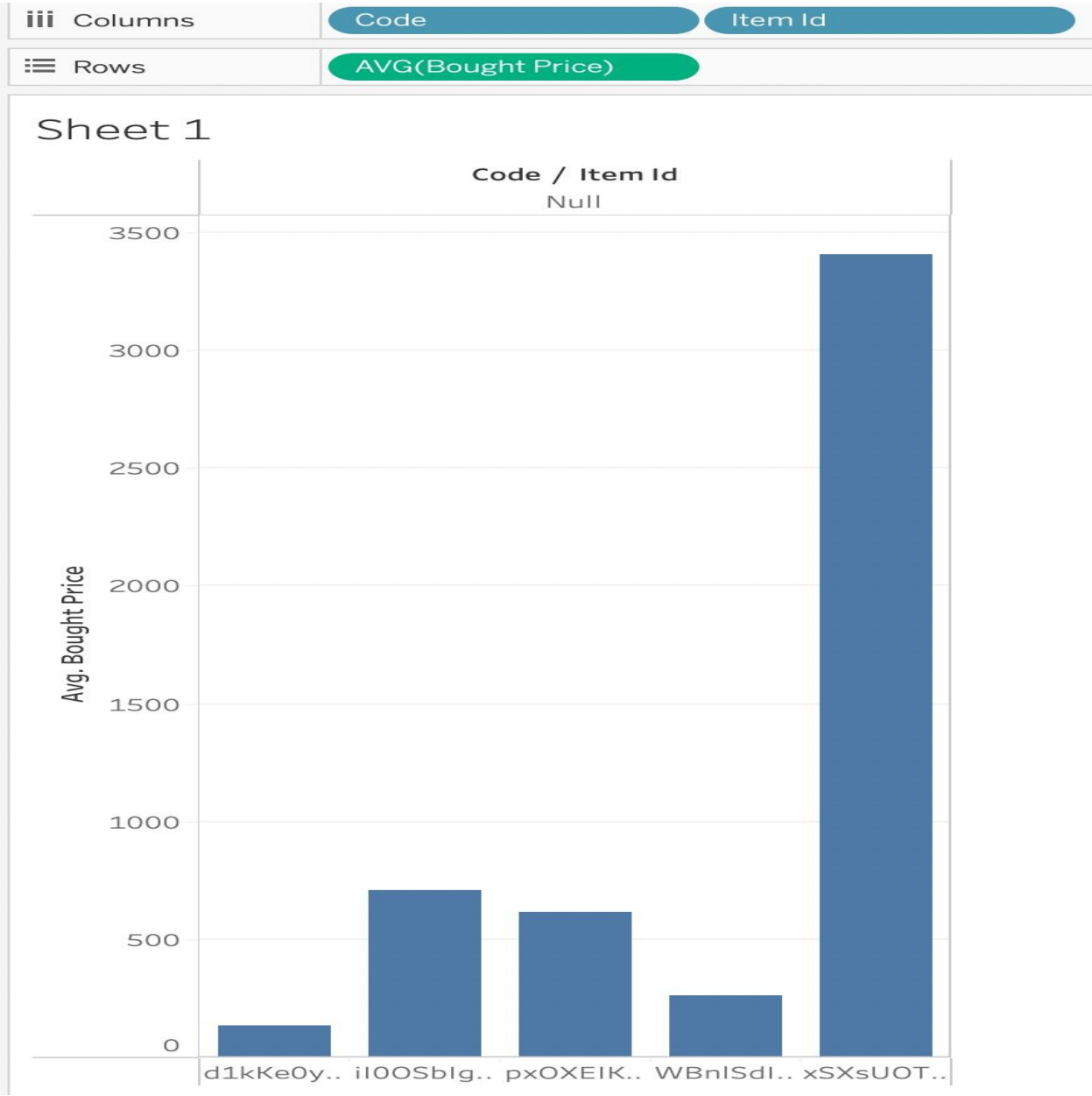
Relational model and Logical Model

O. Brief summary of ETL approach, about half page / one page

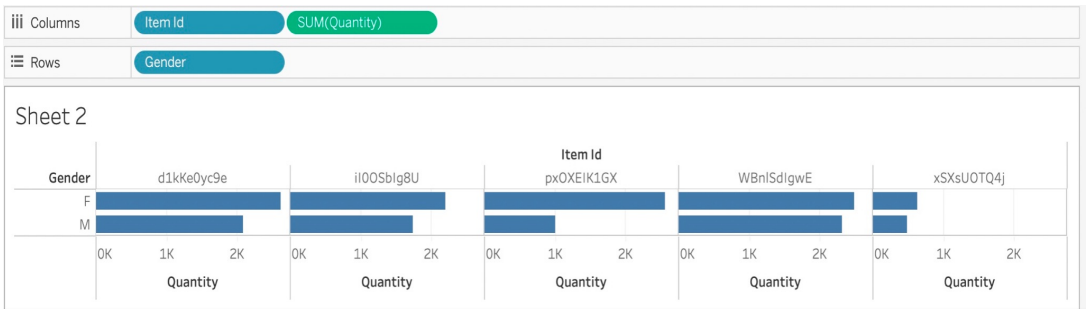
For creating the data warehouse and loading the data into the warehouse we did the following:-

- Extract the columns in the same order as the DW tables by using joins on OLTP and save it as a csv file.
- Copy the CSV file to Amazon S3.
- Truncate the External tables.
- Load the CSV file to the Staging table in Redshift using COPY command.
- For initial load, load the whole data into DW tables.
- For incremental load, run a Procedure to check with the already present rows and, update them if they are modified or, insert the new rows.

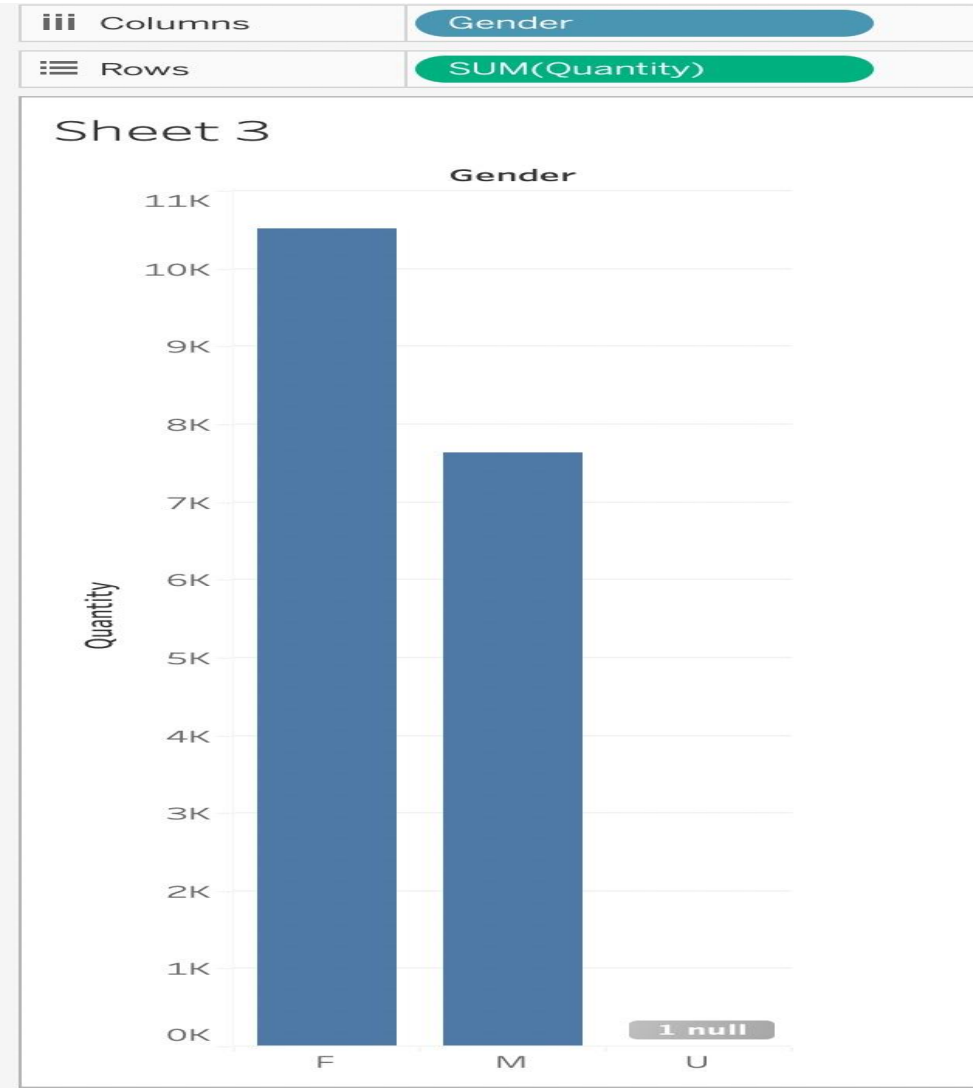
P. SQL and data analysis from DW systems, Reports/Charts



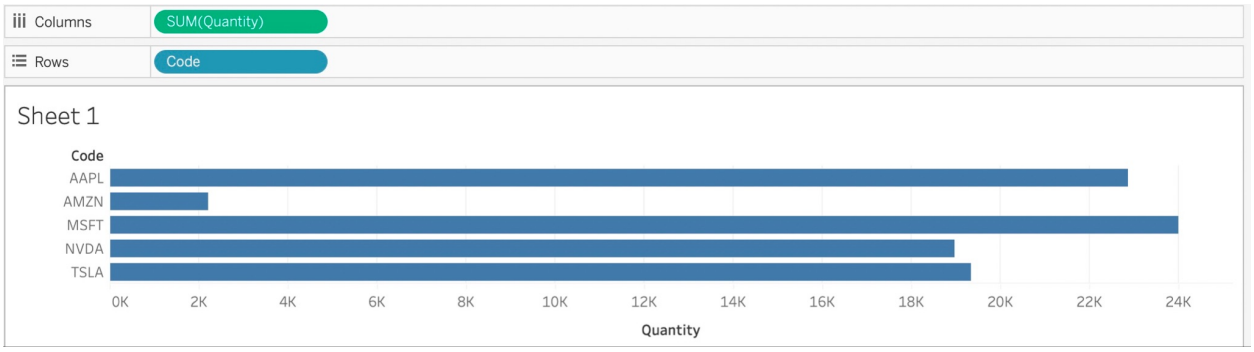
Average Buying Price for each Item



Gender vs item and Quantity



Quantity vs Gender



Stock vs Quantity

Q: Lesson learned

- Learned a lot about Stock Trading.
- Implementation and design of Data Warehouse.
- Creation of History, External and Partition Tables.
- Implementing ETL, and Visualising using BI tools.
- AWS - RDS, S3, IAM Roles, Redshift.
- Technologies like React, Firebase, Docker, Kubernetes, Jenkins (CI/CD), Django, Tableau, DBeaver, etc.
- Learned a lot about cross platform communication using APIs.
- Last but not the least, how to manage time, work with teammates and importance of deadlines.

Things that didn't go well:-

- We spent a lot of time choosing the right DB for the use case, as we spent almost ¾ th of the time working on Mongodb as a backend.

R: Appendix

OLTP DDL code (Tables, Constraints, Trigger, History tables, any function/ procedure created etc.)

```
CREATE TABLE address (
    add_id          BIGINT NOT NULL,
    type            VARCHAR(10) NOT NULL,
```

Data Insertions:-

```
import mysql.connector
import random
import string
import hashlib
import datetime as d
import sys
from api import result
#establishing the connection
conn = mysql.connector.connect(
   user='root', password='rootroot',
host='ring.cpqiozbjlzz9.us-east-1.rds.amazonaws.com',
database='ringer',auth_plugin='mysql_native_password')

#Creating a cursor object using the cursor() method
cursor = conn.cursor()


cursor.execute("set foreign_key_checks=0;")
conn.commit()

# For initial Volume
cursor.execute("SELECT * FROM user_data WHERE user_id='1';")
on = cursor.fetchall()
if len(on) == 0:
     cursor.execute(f""INSERT INTO user_data VALUES
('1','xxxxxx','xxxxxx','xxxxxxxxxxxx','xxxxxxxx','xxxx@xxxx.xxx
','{d.datetime.now().strftime('%Y-%m-%d %H:%M:
%S')}',10000,'{d.date(random.randint(1990,2010),
random.randint(1,12),random.randint(1,28)).strftime("%y-%m-
%d")}','M','{d.datetime.now().strftime('%Y-%m-%d %H:%M:
%S')}');""")
cursor.execute("SELECT * FROM payment WHERE pay_id='1';")
on = cursor.fetchall()
if len(on) == 0:
     cursor.execute(f""INSERT INTO payment VALUES
('1','1','{d.datetime.now().strftime('%Y-%m-%d %H:%M:
```

DW Tables (Similar code for Staging Tables) :-

```
CREATE TABLE curriencies (
    code              VARCHAR(10) NOT NULL,
    name              VARCHAR(30) NOT NULL,
    is_active         CHAR(1) NOT NULL,
    is_base_currency  CHAR(1) NOT NULL,
    updated_date      DATE NOT NULL
);

ALTER TABLE curriencies ADD /* CONSTRAINT curriencies_pk */
PRIMARY KEY ( code );

CREATE TABLE itemDW (
    item_id       VARCHAR(10) NOT NULL,
    code          VARCHAR(10) NOT NULL,
    name          VARCHAR(30) NOT NULL,
    price_id      VARCHAR(10) NOT NULL,
    buy           DECIMAL(10, 3) NOT NULL,
    sell          DECIMAL(10, 3) NOT NULL,
    updated_date  DATE NOT NULL
);

ALTER TABLE itemDW ADD /* CONSTRAINT itemDW_pk */ PRIMARY KEY (
item_id,

price_id );

CREATE TABLE tradeDW (
    trade_id        VARCHAR(10) NOT NULL,
    quantity        DECIMAL(10, 3) NOT NULL,
    unit_price      DECIMAL(10, 3) NOT NULL,
    updated_date    DATE NOT NULL,
    buyer_id        VARCHAR(10) NOT NULL,
    seller_id       VARCHAR(10) NOT NULL,
    pay_id          VARCHAR(10) NOT NULL,
    amount          DECIMAL(10, 3) NOT NULL,
    offer_id        VARCHAR(10) NOT NULL,
```

ETL Code Used:-

```
export PATH=$PATH:/Applications/MySQLWorkbench.app/Contents/
MacOS

Staging-Initial -

Staging-User-

mysql -u root -p --database=ringer --host=ring.cpqiozbjlzz9.us-
east-1.rds.amazonaws.com --port=3306 --batch -e "select
u.user_id,u.first_name,u.last_name,u.username,u.email,u.wallet,
DATE(u.DOB),u.gender,DATE(u.updated_date),a.add_id,a.type,a.str
eet_address,a.city,a.state,a.zipcode,b.bank_id,DATE(b.exp_date)
,i.inven_id,i.item_id,i.bought_price,i.quantity from user_data
u join bank b on u.user_id = b.user_id join address a on
u.user_id = a.user_id join inventory i on u.user_id = i.user_id
WHERE u.updated_date > date_sub(curdate(),interval 30 day) or
b.updated_date > date_sub(curdate(),interval 30 day) or
a.updated_date > date_sub(curdate(),interval 30 day) or
i.updated_date > date_sub(curdate(),interval 30 day);" | sed
's/\t/","/g;s/^/"/;s/$/"/;s/\n//g' > user_bank_add_inven.csv

Staging-Trade-

mysql -u root -p --database=ringer --host=ring.cpqiozbjlzz9.us-
east-1.rds.amazonaws.com --port=3306 --batch -e "select
t.trade_id,t.quantity,t.unit_price,DATE(t.updated_date),t.buyer
_id,t.seller_id,p.pay_id,p.amount,o.offer_id,ROUND(o.quantity,3
) as "totalquantity",o.buy_sell,o.price,u.user_id,
a.add_id,b.bank_id,i.inven_id,t.item_id,it.price_id from trade
t,trade_offer tto,offer o,payment p ,user_data u, address a,
bank b, inventory i, item it WHERE (t.pay_id = p.pay_id and
tto.trade_id = t.trade_id and tto.offer_id = o.offer_id and
(u.user_id=t.buyer_id or u.user_id=t.seller_id) and
(a.user_id=t.buyer_id or a.user_id=t.seller_id) and
(b.user_id=t.buyer_id or b.user_id=t.seller_id) and
(i.user_id=t.buyer_id or i.user_id=t.seller_id) and
```

Note: There is still a lot of database, Backend, DW and Frontend code, which can be seen in this github link.