Advance Project

Course : ECE-GY-995
Section : BK01
Submission date : 19/12/2020

NAME: Asif Hasnain
NETID: mah1042

NAME: Shruti Singh
NETID: ss13345

Table of Contents

A) Project Background (Abstract / Description of the project)

The background on which our project is developed is stock trading. Two main questions that cover the basis of the project are : what is the stock market ? and How does the stock market work?. The term "stock market" often refers to one of the major stock market indexes, such as the Dow Jones Industrial Average or the S&P 500. Because

it's hard to track every single stock, these indexes include a section of the stock market and their performance is viewed as representative of the entire market.

We often see a news headline that says the stock market has moved lower, or that the stock market closed up or down for the day. Most often, this means stock market indexes have moved up or down, meaning the stocks within the index have either gained or lost value as a whole. Investors who buy and sell stocks hope to turn a profit through this movement in stock prices.

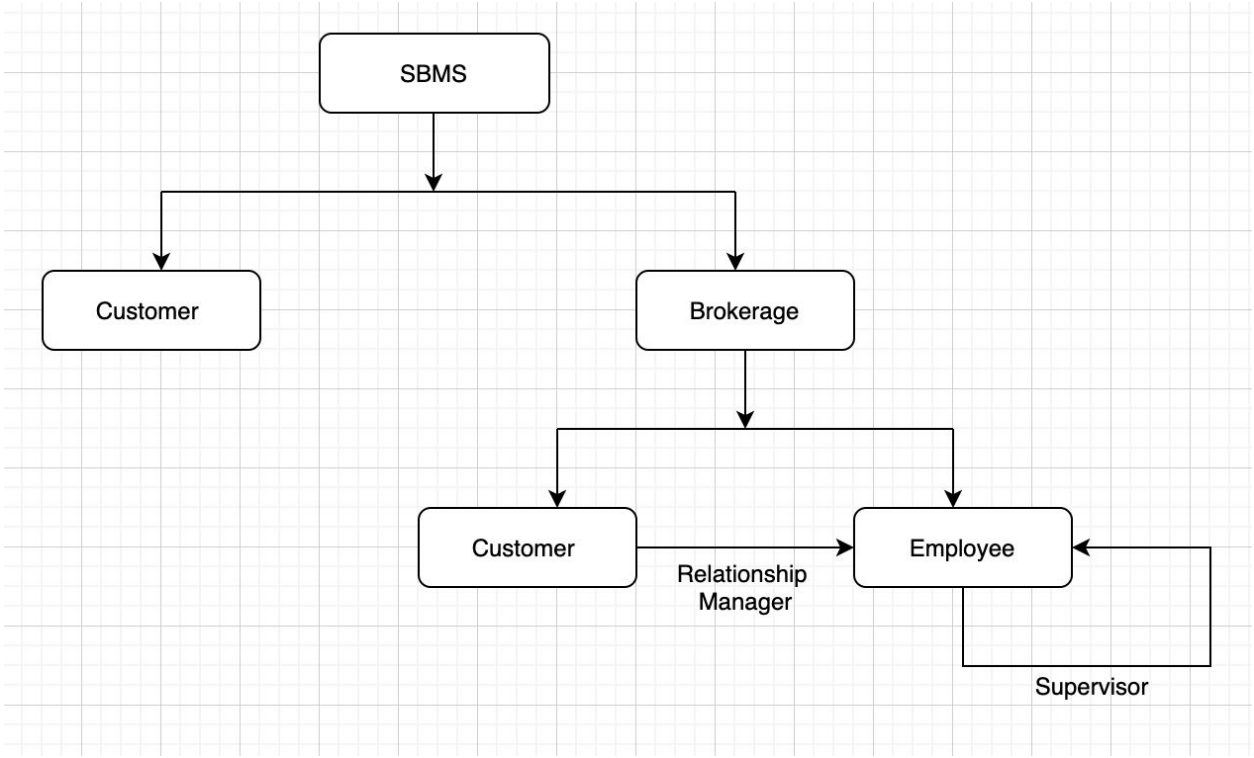
Stock market operates much like an auction house, the stock market enables buyers and sellers to negotiate prices and make trades. The stock market works through a network of exchanges like the New York Stock Exchange or the Nasdaq. Companies list shares of their stock on an exchange through a process called an initial public offering or IPO. Investors purchase those shares, which allows the company to raise money to grow its business. Investors can then buy and sell these stocks among themselves, and the exchange tracks the supply and demand of each listed stock.

That supply and demand help determine the price for each security, or the levels at which stock market participants, investors and traders are willing to buy or sell.
Buyers offer a “bid,” or the highest amount they’re willing to pay, which is usually lower than the amount sellers “ask” for in exchange. This difference is called the bid-ask spread. For a trade to occur, a buyer needs to increase his price or a seller needs to decrease hers.
This all may sound complicated, but computer algorithms generally do most of price-setting calculations. When buying stock, you’ll see the bid, ask, and bid-ask spread on your broker's website, but in many cases, the difference will be pennies, and won’t be of much concern for beginner and long-term investors.

B) Business case (The business case on which you are developing DB systems**)**

The business case for our project is buy and sell of a stock via an application. This project aims at creating a stock management system which will provide the user an option to stock trading. In this application there can be two types of users, individual users and brokerage firms. Both the users have different user experiences although the main goal is stock marketing, in case of an individual user, the user can subscribe to the stocks for which they want to keep track of the trend and in which they want to trade. While in case of brokerage firm users, the admin user has the access to create

employees of their firms and employees and admin both can register individual users who want to do stock trading via there firm. A brokerage firm has to opt for a membership in order to sign up for the application to use. There are three kinds of membership that SBM(stock brokerage management) provides, gold, silver and basic. Gold membership has 500$ as service charge and 1$ of trading fee, Silver membership has 200$ of service charge and 1.5$ of trading fee and the Basic membership has no service charge but the trading fee is high about 2$.



Hierarchy

C) Project Objectives (what is the goal of the project, what the business will gain from the project's result, it should be SMART goal)

The objective of the project is to create a Stock management system as a web based application to meet the current stock management requirements of a brokerage firm and an individual. Our initial objective was to create this application is to recreate the real feel of a stock management system. Such that it can be used by beginners to learn

about the stock market, working of the stock market and how to invest in stocks. This application can be used in finance related courses as a simulation tool where students can learn stock investment by playing around with this application with some dummy credit.

D) Project Scope: (High level list of steps/work is required to achieve project's objective)

In order to achieve the objective of this project, we followed the following steps:

- Understood the working of a stock management system
- Gathered info about API from where we could get the real time information on various stocks
- Created our business model for the stock management system.
- Designed an oltp database to facilitate the the web application
- Created stored procedures to interact with the database.
- Implemented REST APIs to perform CRUD operations on the database.
- Developed UI, so that users can use the features provided by our application.
- We have also created a data warehouse for analytics.

E) Product Scope: (Features of the end results of the product, meaning that features of your OLTP, Web Application, DW system, Analytic/Reporting tool)

- OLTP
    - To be able to perform CRUD operations on data of users, customers, employees, admins, brokerage data, membership information, access information, stock data, subscription of the users etc.
- Web Application
    - A page for users to login into the application
    - A signup page

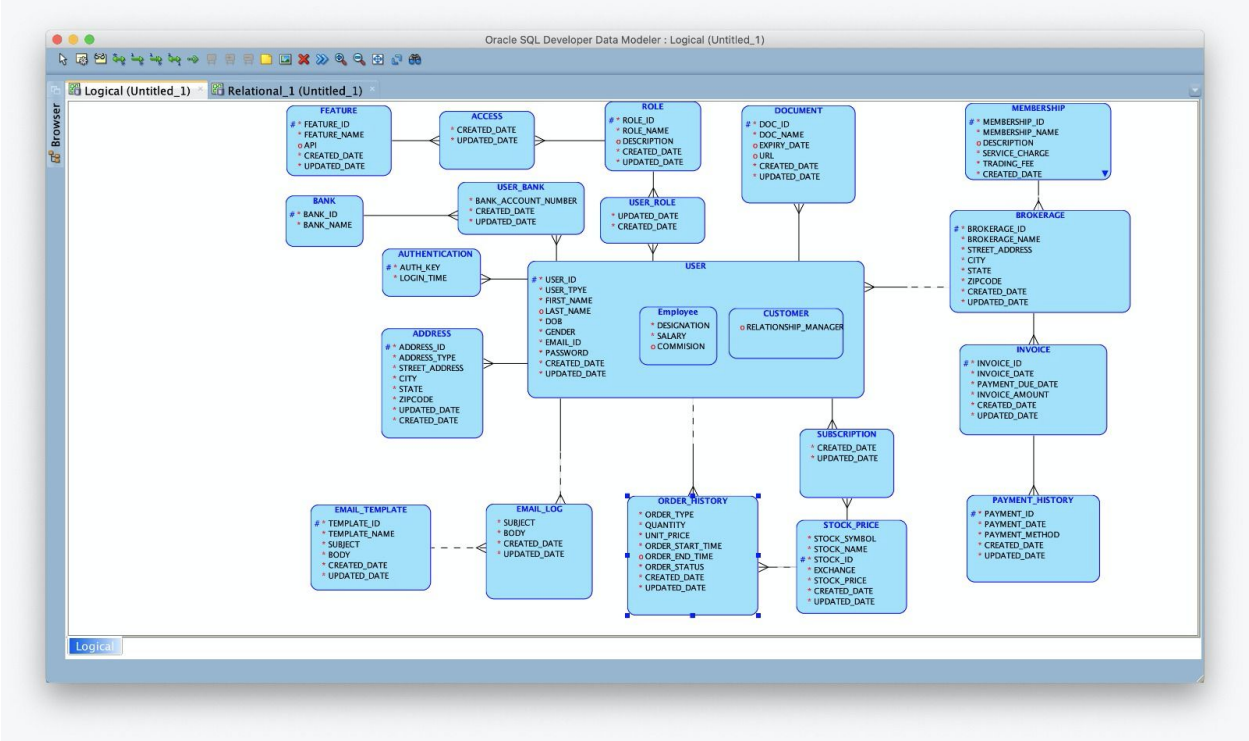
    - Ability to subscribe and unsubscribe to the stocks
    - A service to load and unload credit from their bank account to their SBM account.
    - Admin to be able to create employees and customers for their brokerage firm.
    - Employee to be able to create customers for their brokerage firm.
    - Admin and Employee to be able to imitate customer journey.
- Data Warehouse System
    - To be able to store a huge amount of data of users, subscription, transaction, brokerage, orders and payment.
    - Store denormalized OLTP tables and keep data with similar attributes in one table in order to minimize multi-table joins and get required data for analytics with low cost and fast queries.
- Analytics/Reporting tool
    - Use Data Warehouse System data and create meaning data with certain attributes which can help in business and operation decisions.
    - Various Data tools present in OCI Autonomous Data Warehouse like Apex, Data insight, Business Models are used to get desired analytical information from Data warehouse like average price of a stock.
    - Option to write procedures for business reports. Cron jobs can be used to automate report generation at predetermined intervals.

F) Project Milestones

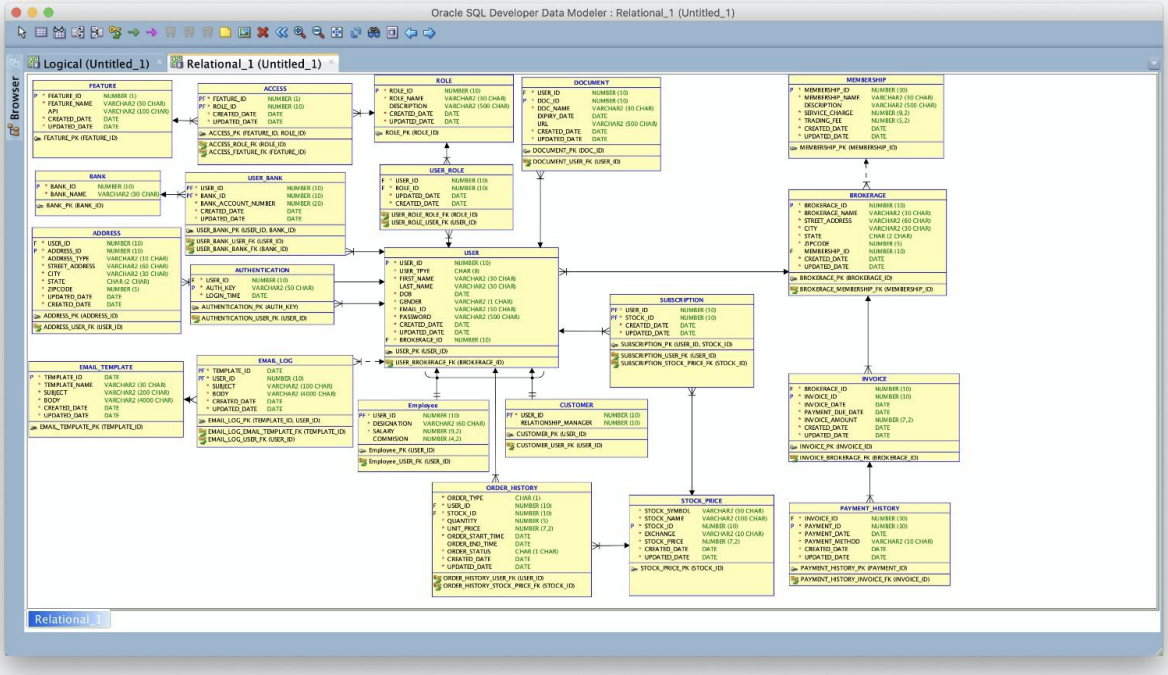
Project Milestones	**Time to complete the milestone(**1 day = 3hours**)**

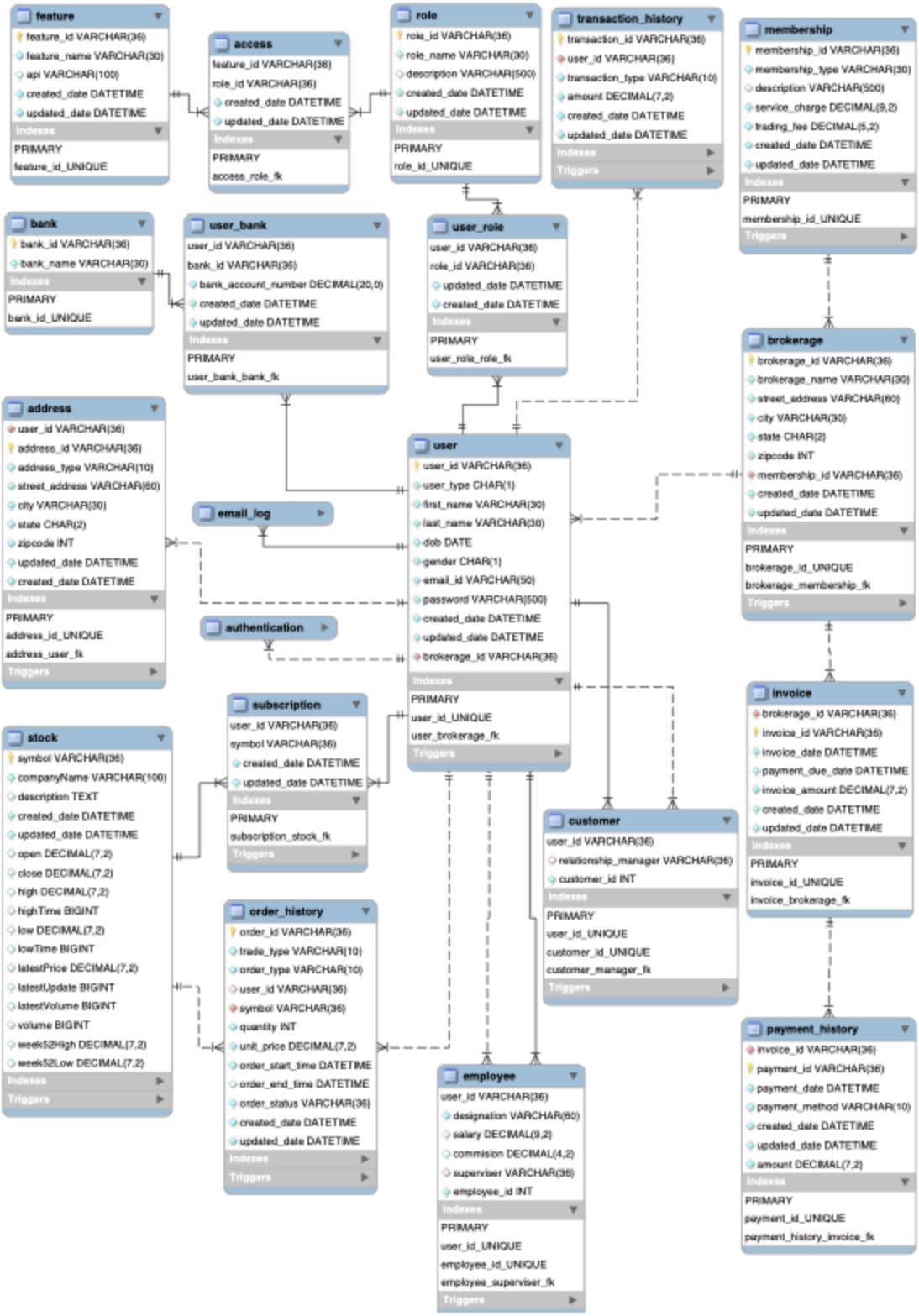
Finalize Business Model	2 week
Finding and understanding stock api	2 days
Infrastructure design	1 week
MySQL OLTP DB design	1 week
OLTP implementation	1 week
History Table	1 hour
Stock api integration	1 day
Login, Sign up, User Homepage	2 week
Account details, subscription	3 days
Buy/sell, Transaction history	1 week
payment history & invoice	3 days
Access Management	3 days
Data warehouse design	1 week
Data warehouse implementation	2 days
ETL	1 week
External table, Partition Table	1 day
Analytics Report	2 days
Testing and Bug Fix	1 week

G) Logical Model (OLTP)



H) Relational Model (OLTP)


Oracle SQL Developer Data Modeler : Relational_1 (Untitled_1)
Logical (Untitled_1)
Relational_1 (Untitled_1)
Browser
FEATURE
ACCESS
ROLE
DOCUMENT
MEMBERSHIP
BANK
USER_BANK
USER_ROLE
BROKERAGE
ADDRESS
AUTHENTICATION
USER
SUBSCRIPTION
EMAIL_TEMPLATE
EMAIL_LOG
Employee
CUSTOMER
INVOICE
ORDER_HISTORY
STOCK_PRICE
PAYMENT_HISTORY
Relational_1


feature
feature_id VARCHAR(36)
feature_name VARCHAR(30)
api VARCHAR(100)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
feature_id_UNIQUE
access
feature_id VARCHAR(36)
role_id VARCHAR(36)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
access_role_fk
role
role_id VARCHAR(36)
role_name VARCHAR(30)
description VARCHAR(500)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
role_id_UNIQUE
transaction_history
transaction_id VARCHAR(36)
user_id VARCHAR(36)
transaction_type VARCHAR(10)
amount DECIMAL(7,2)
created_date DATETIME
updated_date DATETIME
Indexes
Triggers
membership
membership_id VARCHAR(36)
membership_type VARCHAR(30)
description VARCHAR(500)
service_charge DECIMAL(9,2)
trading_fee DECIMAL(5,2)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
membership_id_UNIQUE
Triggers
bank
bank_id VARCHAR(36)
bank_name VARCHAR(30)
Indexes
PRIMARY
bank_id_UNIQUE
user_bank
user_id VARCHAR(36)
bank_id VARCHAR(36)
bank_account_number DECIMAL(20,0)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
user_bank_bank_fk
user_role
user_id VARCHAR(36)
role_id VARCHAR(36)
updated_date DATETIME
created_date DATETIME
Indexes
PRIMARY
user_role_role_fk
brokerage
brokerage_id VARCHAR(36)
brokerage_name VARCHAR(30)
street_address VARCHAR(60)
city VARCHAR(30)
state CHAR(2)
zipcode INT
membership_id VARCHAR(36)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
brokerage_id_UNIQUE
brokerage_membership_fk
Triggers
address
user_id VARCHAR(36)
address_id VARCHAR(36)
address_type VARCHAR(10)
street_address VARCHAR(60)
city VARCHAR(30)
state CHAR(2)
zipcode INT
updated_date DATETIME
created_date DATETIME
Indexes
PRIMARY
address_id_UNIQUE
address_user_fk
Triggers
email_log
user
user_id VARCHAR(36)
user_type CHAR(1)
first_name VARCHAR(30)
last_name VARCHAR(30)
dob DATE
gender CHAR(1)
email_id VARCHAR(50)
password VARCHAR(500)
created_date DATETIME
updated_date DATETIME
brokerage_id VARCHAR(36)
Indexes
PRIMARY
user_id_UNIQUE
user_brokerage_fk
Triggers
authentication
invoice
brokerage_id VARCHAR(36)
invoice_id VARCHAR(36)
invoice_date DATETIME
payment_due_date DATETIME
invoice_amount DECIMAL(7,2)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
invoice_id_UNIQUE
invoice_brokerage_fk
subscription
user_id VARCHAR(36)
symbol VARCHAR(36)
created_date DATETIME
updated_date DATETIME
Indexes
PRIMARY
subscription_stock_fk
Triggers
stock
symbol VARCHAR(36)
companyName VARCHAR(100)
description TEXT
created_date DATETIME
updated_date DATETIME
open DECIMAL(7,2)
close DECIMAL(7,2)
high DECIMAL(7,2)
highTime BIGINT
low DECIMAL(7,2)
lowTime BIGINT
latestPrice DECIMAL(7,2)
latestUpdate BIGINT
latestVolume BIGINT
volume BIGINT
week52High DECIMAL(7,2)
week52Low DECIMAL(7,2)
Indexes
Triggers
customer
user_id VARCHAR(36)
relationship_manager VARCHAR(36)
customer_id INT
Indexes
PRIMARY
user_id_UNIQUE
customer_id_UNIQUE
customer_manager_fk
Triggers
order_history
order_id VARCHAR(36)
trade_type VARCHAR(10)
order_type VARCHAR(10)
user_id VARCHAR(36)
symbol VARCHAR(36)
quantity INT
unit_price DECIMAL(7,2)
order_start_time DATETIME
order_end_time DATETIME
order_status VARCHAR(36)
created_date DATETIME
updated_date DATETIME
Indexes
Triggers
employee
user_id VARCHAR(36)
designation VARCHAR(60)
salary DECIMAL(9,2)
commision DECIMAL(4,2)
superviser VARCHAR(36)
employee_id INT
Indexes
PRIMARY
user_id_UNIQUE
employee_id_UNIQUE
employee_superviser_fk
Triggers
payment_history
invoice_id VARCHAR(36)
payment_id VARCHAR(36)
payment_date DATETIME
payment_method VARCHAR(10)
created_date DATETIME
updated_date DATETIME
amount DECIMAL(7,2)
Indexes
PRIMARY
payment_id_UNIQUE
payment_history_invoice_fk

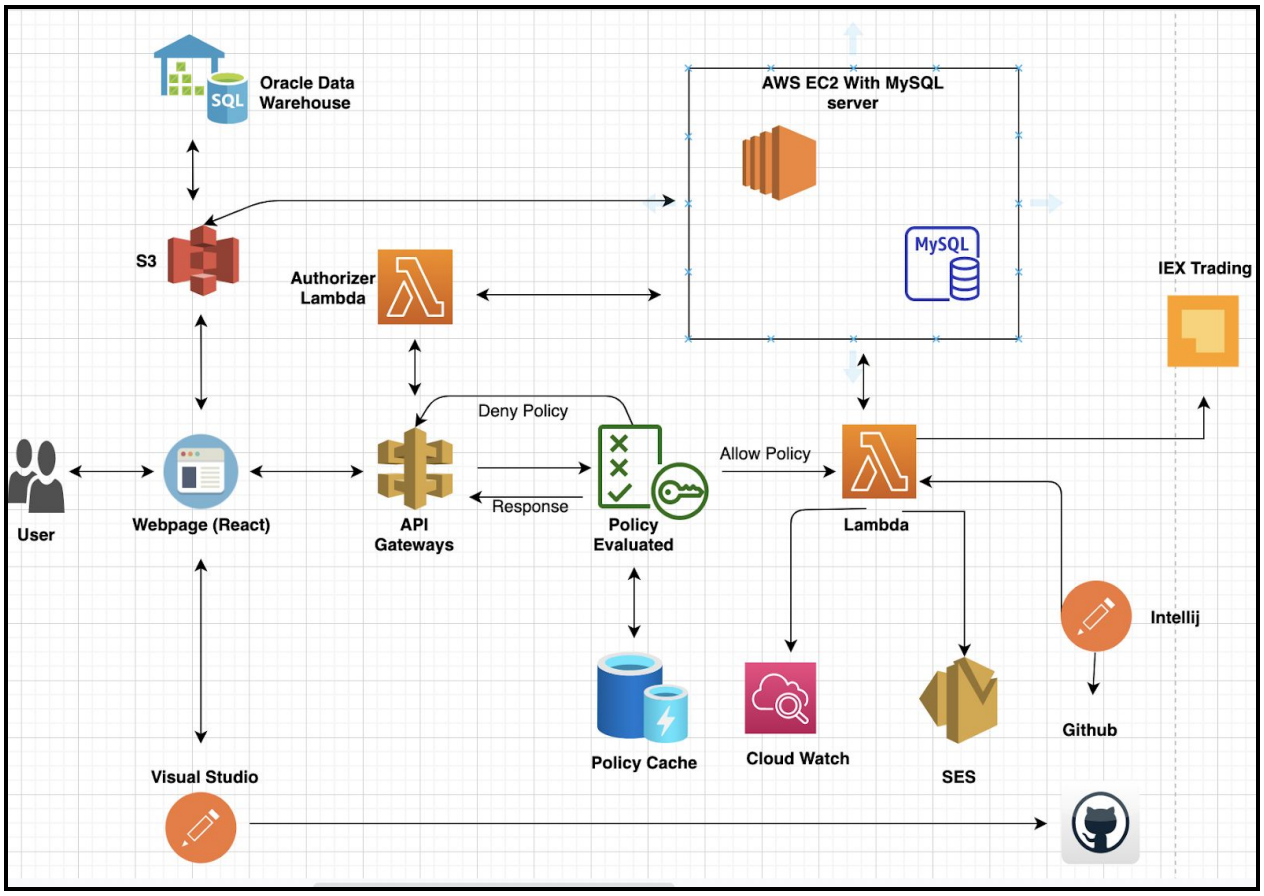
I) Assumptions and Constraints

- A user can have only one bank account.
- Every brokerage customer will have one relationship manager
- An employee may or may not have a supervisor.
- Only one payment can be made against one invoice
- User can have multiple address
- User can have only one bank account
- Every brokerage must have a one and only one membership
- Every brokerage must have root user provided at the time of signup
- A stock can only be sold after purchasing that stock, user can only sell their purchase stocks.
- First name and last name should not be more than 30 characters.
- Gender can be either male or female
- Email id can not be more than 50 characters.
- City can have a maximum of 30 characters.
- Zip Code should be a 5 digit number.
- Bank Account number should be less and equal to 30 digits.
- Bank name, brokerage name, membership type, feature name, role name, Should not be more than 30 characters.
- Company names can have a maximum of 100 characters.
- All intraday stock order will be squared off the same day by closing time of the market

J) Infrastructure

OLTP : MySQL
mysql Ver 8.0.22 for Linux on x86_64 (MySQL Community Server - GPL)
DW: Oracle autonomous database
Reporting Tool: OCI ADW
Front end Programming Language: React.js, JavaScript, HTML, CSS
Back end Programming Language : Java, SQL
Sever : EC2

Other tools : S3, Lamda, API gateway, SES, Couldwatch, GIT, JDBC, IntelliJ, Visual Studio, IEX services



System Design

K) Record counts for each OLTP tables

```
2 •   SELECT COUNT(*) FROM access;
```

100% 1:1

Result Grid Filter Rows: Search Ex

COUNT(*)
30

```
2 •   SELECT COUNT(*) FROM address;
```

100% 29:2

Result Grid Filter Rows: Search

COUNT(*)
6542

```
2 •  SELECT COUNT(*) FROM authentication;
```

100% 37:2

Result Grid Filter Rows: Search Export:

COUNT(*)
78

```
2 •  SELECT COUNT(*) FROM bank;
```

100% 27:2

Result Grid Filter Rows: Search

COUNT(*)
6

```
2 •  SELECT COUNT(*) FROM brokerage;
```

100% 31:2

Result Grid Filter Rows: Search

COUNT(*)
35

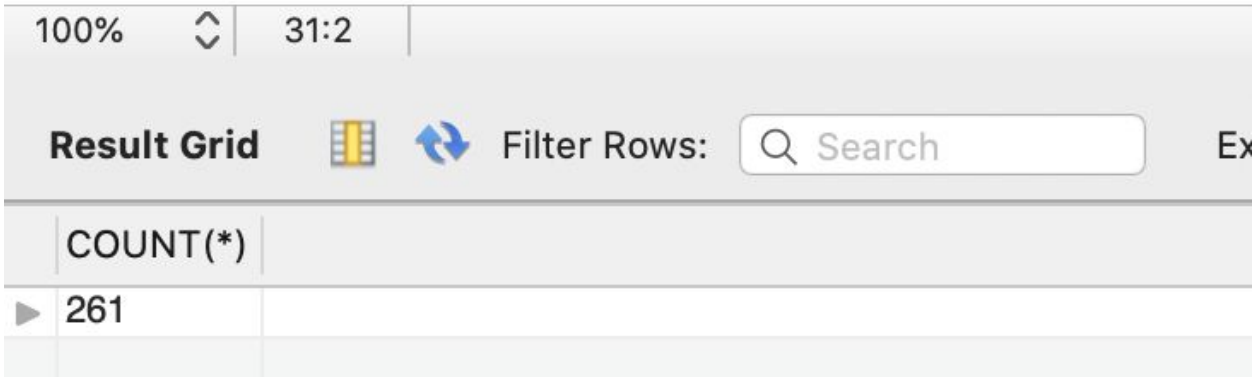
```
2 •  SELECT COUNT(*) FROM customer;
```

100% 31:2

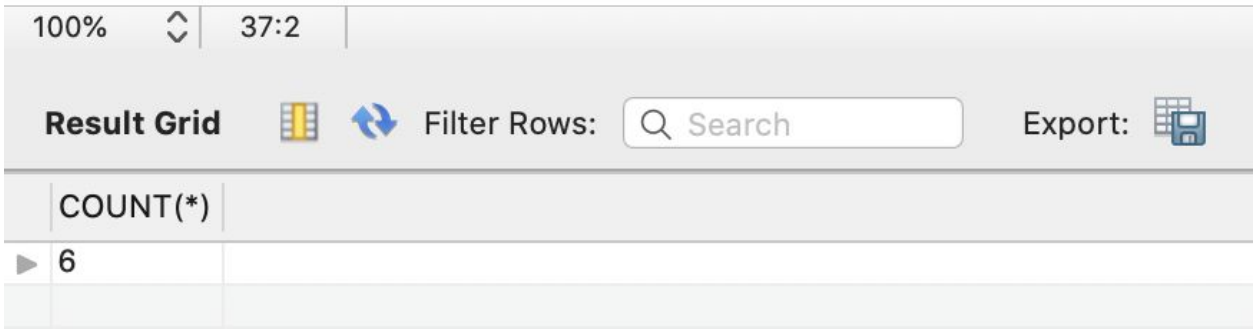
Result Grid Filter Rows: Search Exp

COUNT(*)
3013

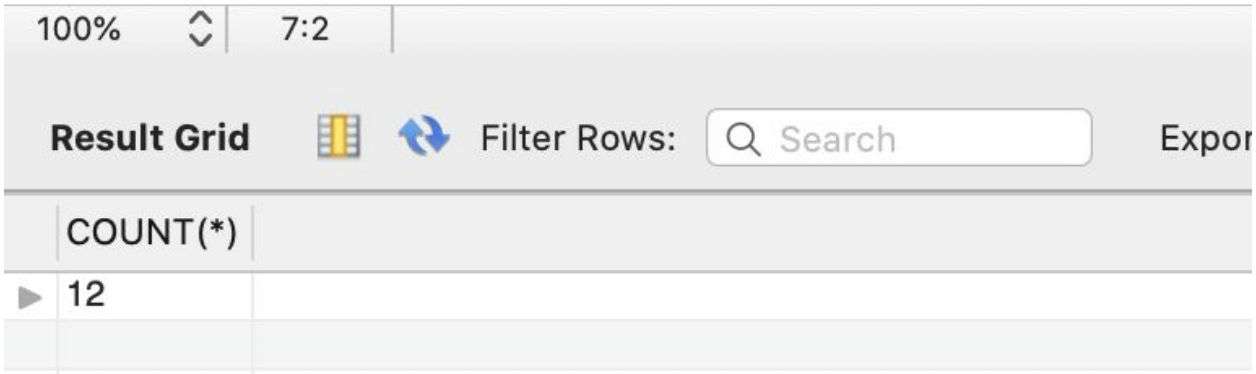
```
2 •  SELECT COUNT(*) FROM employee;
```



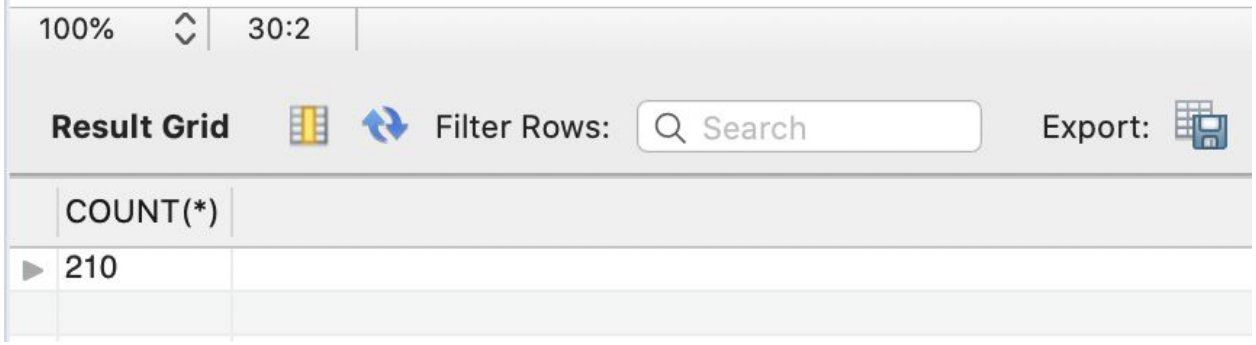

```
2 •  SELECT COUNT(*) FROM extraction_log;
```



```
2 •  SELECT COUNT(*) FROM feature;
```




```
2 •  SELECT COUNT(*) FROM invoice;
```



```
2 •  SELECT COUNT(*) FROM membership;
```

100% 32:2

Result Grid Filter Rows: Search Export:

COUNT(*)
3

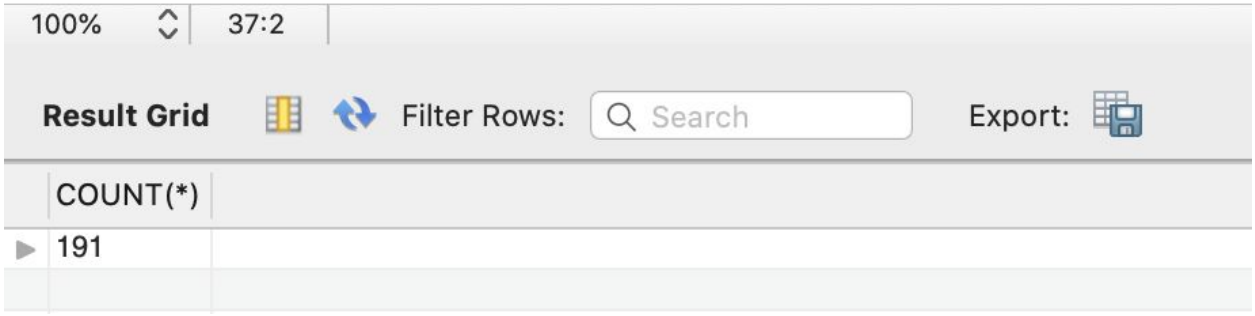
```
2 •  SELECT COUNT(*) FROM order_history;
```

100% 35:2

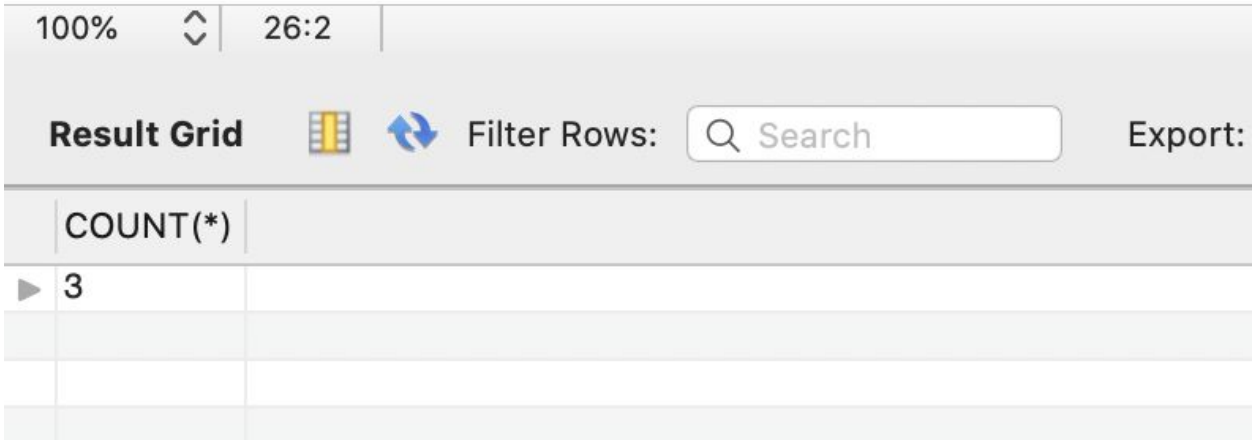
Result Grid Filter Rows: Search Export:

COUNT(*)
66454

```
2 •  SELECT COUNT(*) FROM payment_history;
```




```
2 •  SELECT COUNT(*) FROM role;
```



```
2 •  SELECT COUNT(*) FROM stock;
```

100% 27:2

Result Grid Filter Rows: Search Export:

COUNT(*)
70

```
2 •  SELECT COUNT(*) FROM stock_history;
```

100% 35:2

Result Grid Filter Rows: Search Export:

COUNT(*)
33544

```
2 •  SELECT COUNT(*) FROM subscription;
```

100% 34:2

Result Grid Filter Rows: Search Export:

COUNT(*)
41201

```
2 •  SELECT COUNT(*) FROM transaction_history;
```

100% 41:2

Result Grid Filter Rows: Search Export:

COUNT(*)
2442

```
2 •  SELECT COUNT(*) FROM user;
```

100% 26:2

Result Grid Filter Rows: Search Export:

COUNT(*)
3280

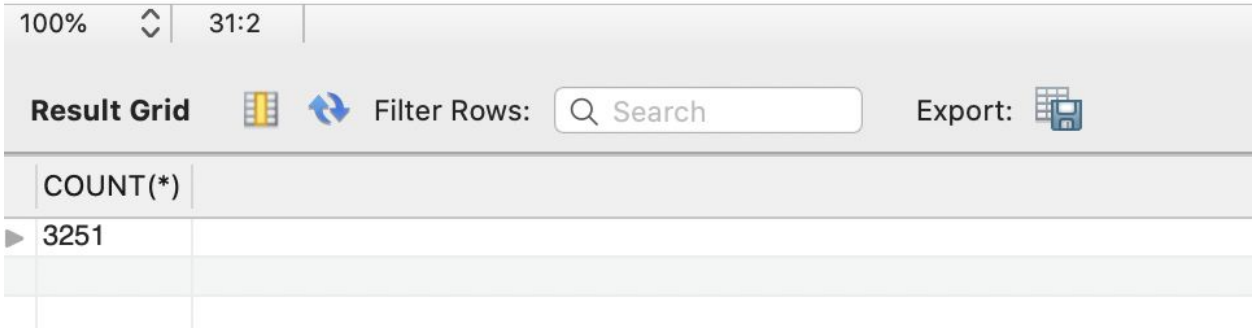
```
2 •  SELECT COUNT(*) FROM user_bank;
```

100% 31:2

Result Grid Filter Rows: Search Export:

COUNT(*)
3034

```
2 •  SELECT COUNT(*) FROM user_role;
```



L) Web application summary (major features, functionality, security measures, user administration etc.)

Application Features and functionality:

For Individual users

- Create new account.
- Login/Logout.
- Reset password
- Change password.
- View profile details.
- Subscribe/Unsubscribe a stock.
- Credit/Debit money from bank.
- All the features are access driven.
- View order list.
- Buy/Sell a stock
    - Interday
    - Intraday

For Brokerage Firms

Admin

- Create New account
- Login/Logout.
- Reset password
- View profile
- Create employees.
- Create Customers
- View list of all employees
- View list of all customers.
- Imitate a customer.
- Administrative access.

Employee

- Login/Logout.
- Reset password
- View profile
- View list of all employees
- View list of all customers.
- Create customers.
- Imitate a customer.
- All the features are access driven.

Security Measures :

- Access to features is provided on basis of roles
- Roles
    - Admin
    - Employee
    - Customer

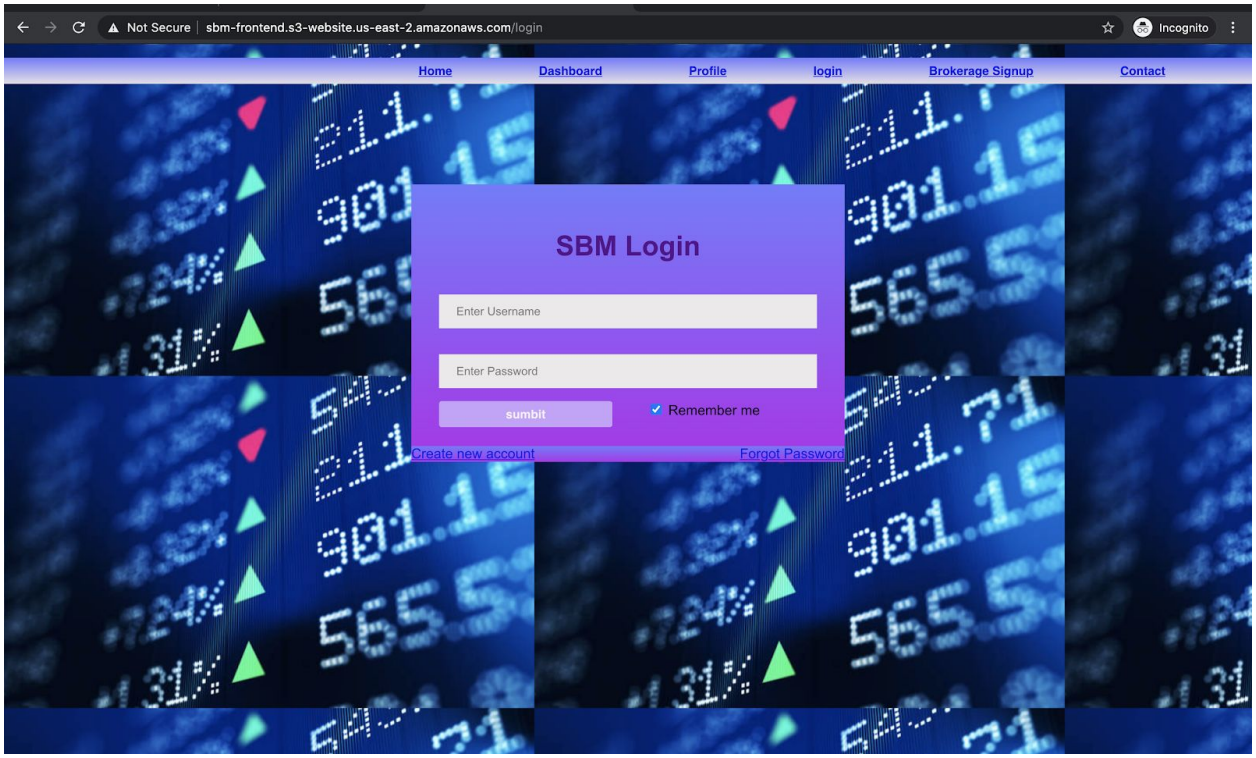
- Features
    - Unsubscribe to a stock
    - Subscribe to a stock
    - Transaction
    - Brokerage invoice details
    - Brokerage Employee
    - Place order
    - Stocks
    - View order list
    - View profile
    - View list of customers
    - Pay for an invoice
    - View subscription

N) Screenshots of the web applications (First page, Login page, some of CRUD page, viewing records etc.)

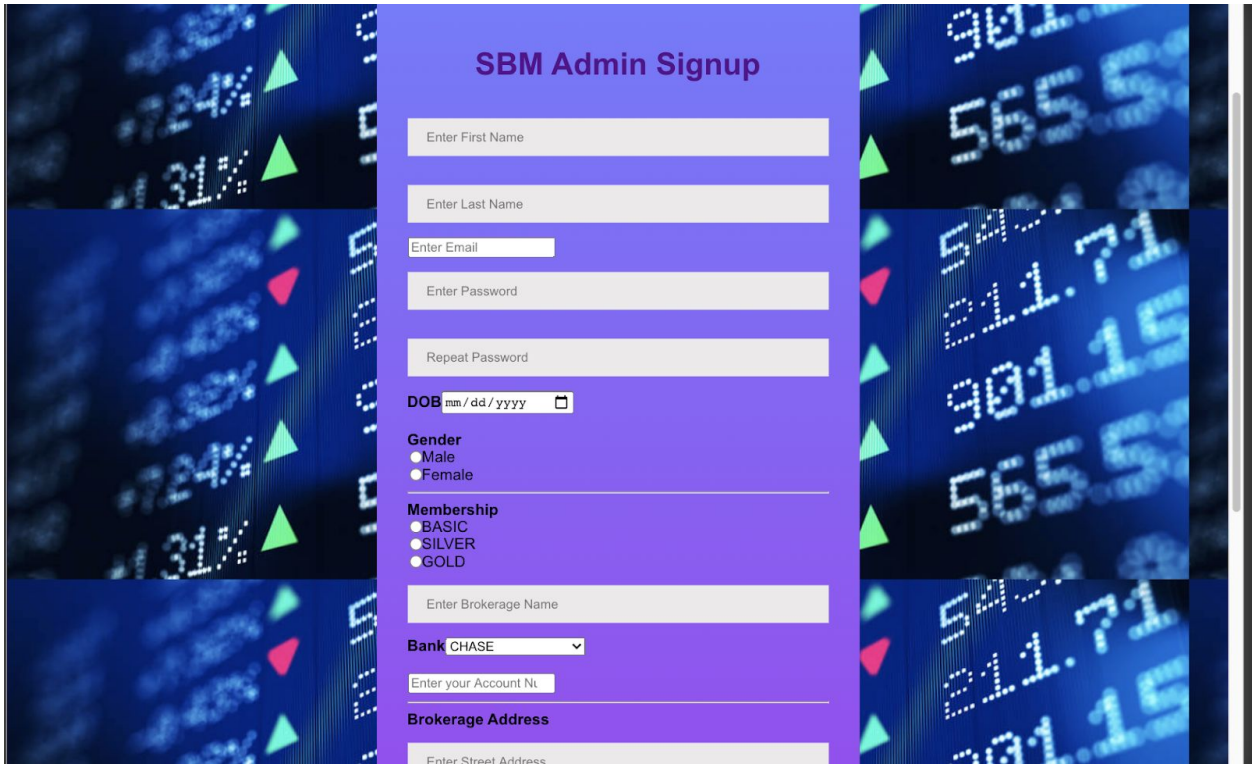
Home Page

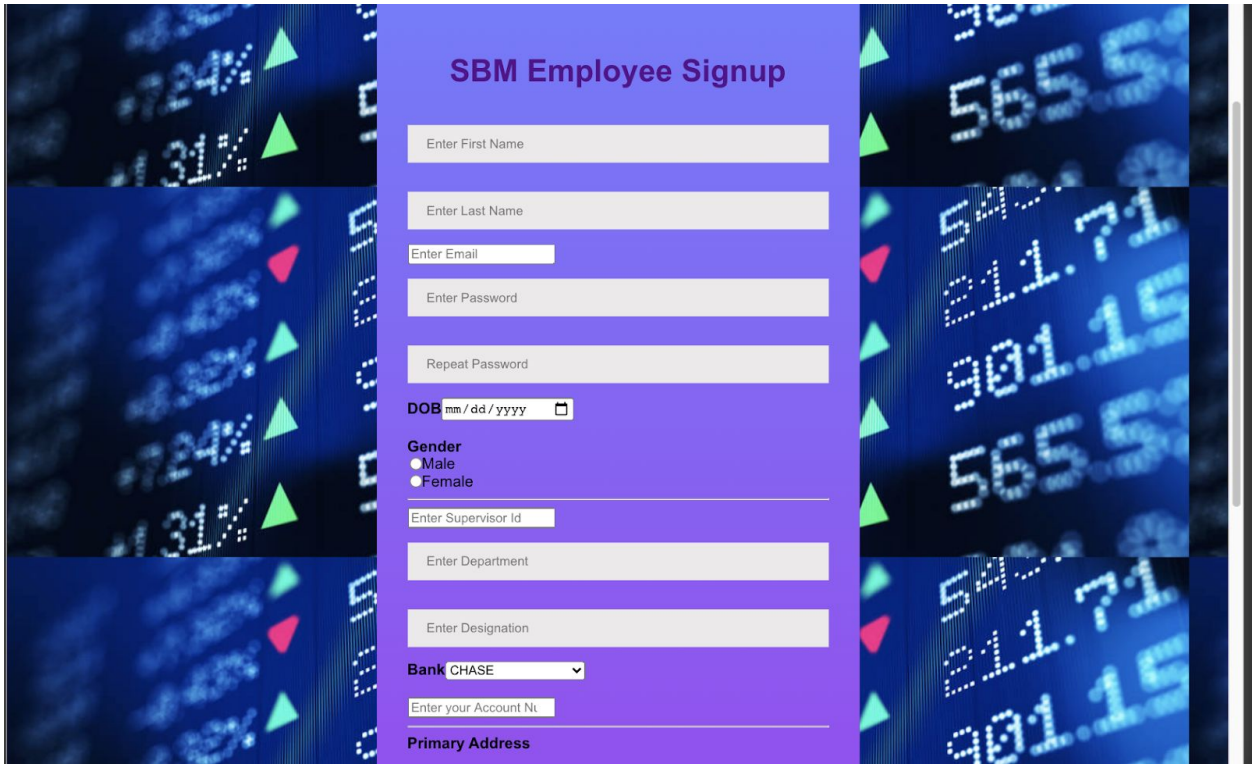

Not Secure | sbm-frontend.s3-website.us-east-2.amazonaws.com/home
Incognito
Home
Dashboard
Profile
login
Brokerage Signup
Contact
Stock Brokerage Management
GOLD
Full access to Customer account ✔
Full access to Admin account ✔
Service Charge : 500$
Trading Fee: 1$
SILVER
Full access to Customer account ✔
Full access to Admin account ✔
Service Charge : 200$
Trading Fee: 1.50$
BASIC
Full access to Customer account ✔
Service Charge : 0$
Trading Fee: 2$
Stock Information
Why is the stock market up today? Why is the market down? What's the outlook for stock market futures? What are the best stocks to buy right now? Is it time to sell your stocks?
The SnP 500 ended a four-session losing streak to climb 1.3%, and the Dow also advanced by more than 1%. Both the Nasdaq Composite and small-cap Russell 2000 ended Tuesday's session at record closing highs.
"The odds of a fiscal deal before year's end have been improving. At this point, we think it is slightly more likely than not that Congress will pass this week a package similar to the recent $748bn bipartisan proposal, which would be close to our standing assumption of a $700bn (3.3% of GDP) package," Goldman Sachs economists led by Jan

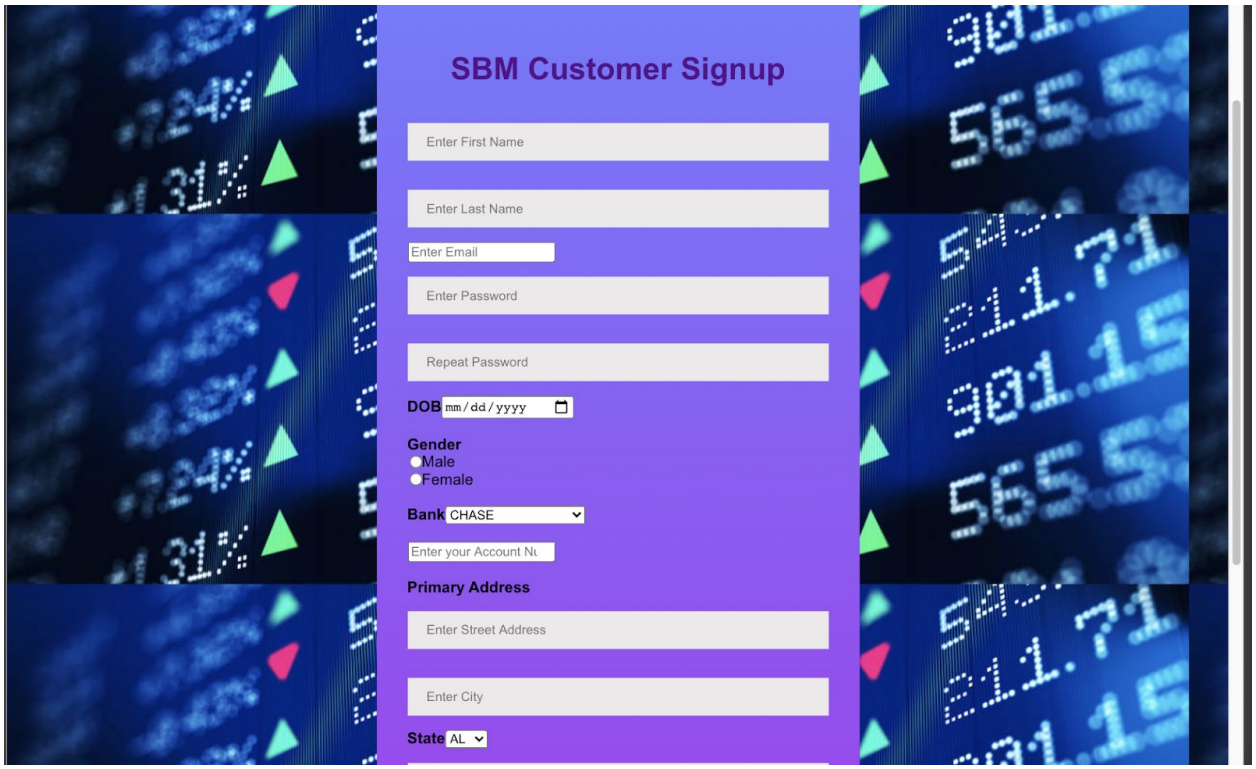
Login Page


Not Secure | sbm-frontend.s3-website.us-east-2.amazonaws.com/login
Incognito
Home
Dashboard
Profile
login
Brokerage Signup
Contact
SBM Login
Enter Username
Enter Password
sumbit
Remember me
Create new account
Forgot Password

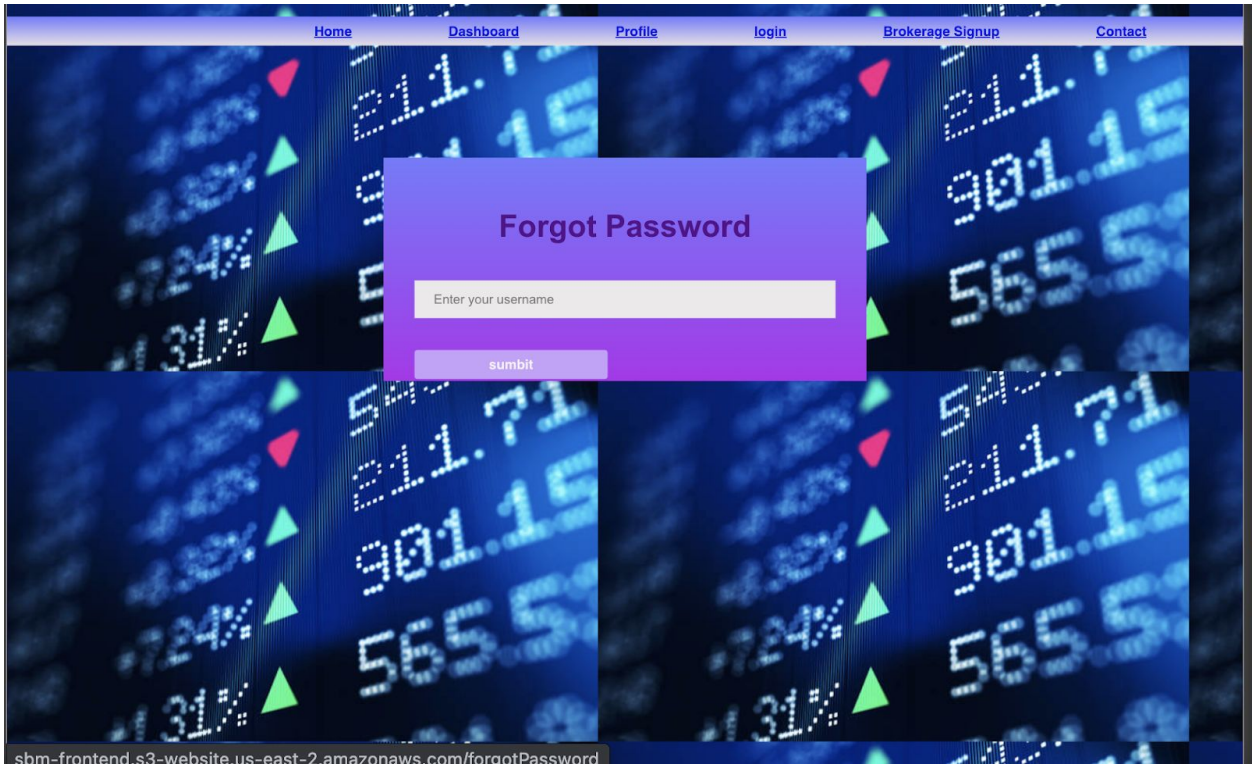
Sign Ups


SBM Admin Signup
Enter First Name
Enter Last Name
Enter Email
Enter Password
Repeat Password
DOB
mm/dd/yyyy
Gender
Male
Female
Membership
BASIC
SILVER
GOLD
Enter Brokerage Name
Bank
CHASE
Enter your Account Nu
Brokerage Address
Enter Street Address

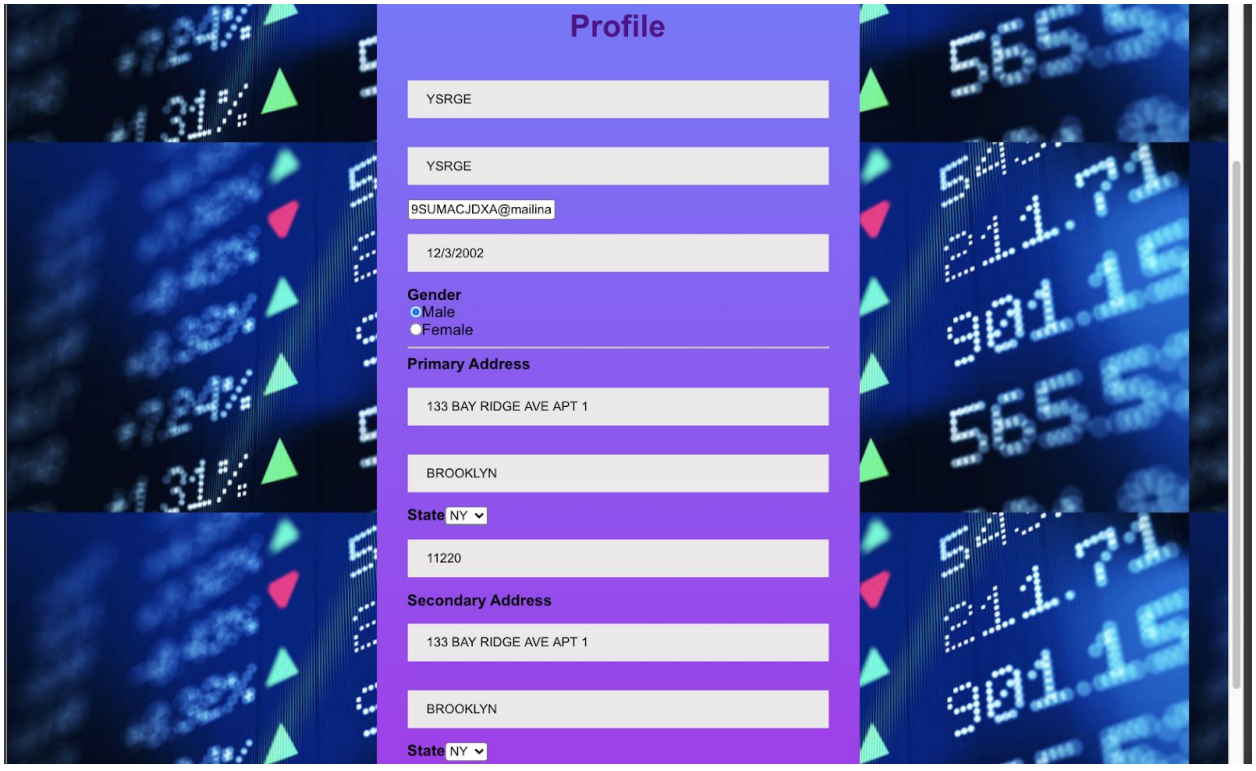

SBM Employee Signup
Enter First Name
Enter Last Name
Enter Email
Enter Password
Repeat Password
DOB mm / dd / yyyy
Gender
Male
Female
Enter Supervisor Id
Enter Department
Enter Designation
Bank CHASE
Enter your Account Nu
Primary Address


SBM Customer Signup
Enter First Name
Enter Last Name
Enter Email
Enter Password
Repeat Password
DOB mm / dd / yyyy
Gender
Male
Female
Bank CHASE
Enter your Account Nu
Primary Address
Enter Street Address
Enter City
State AL

Forgot Password

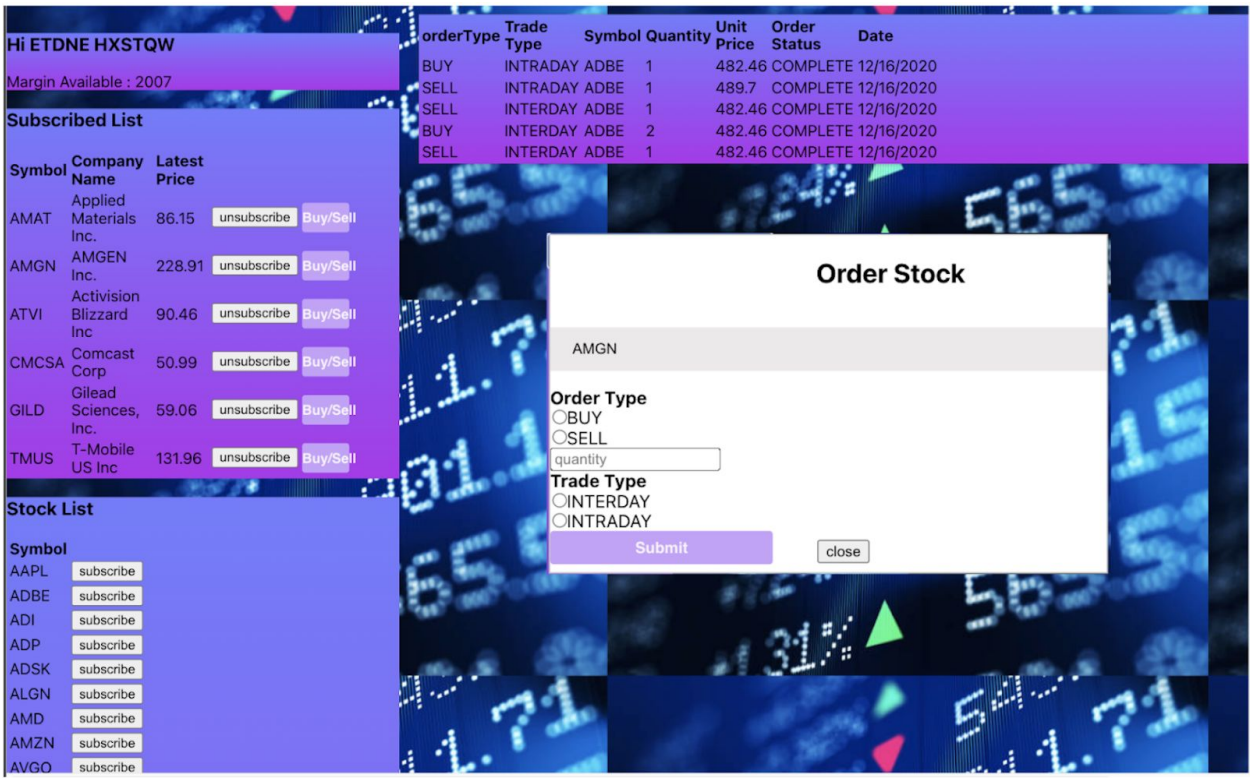


Profile

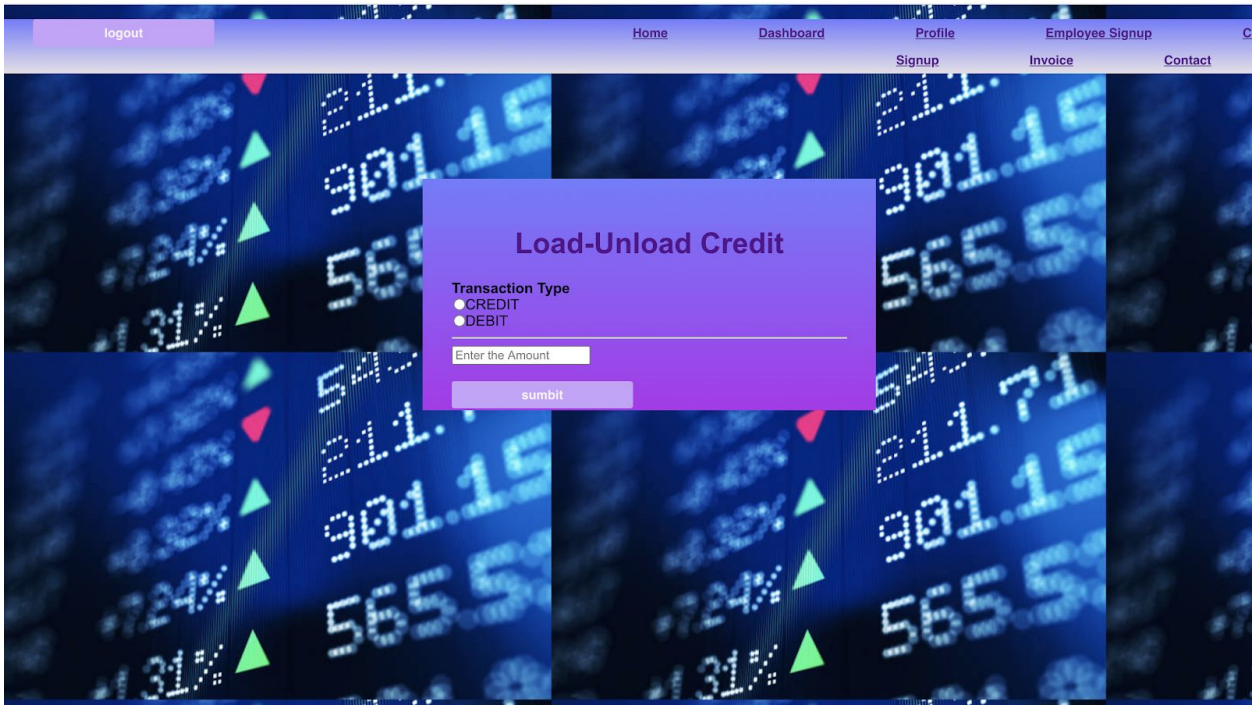


Dashboard

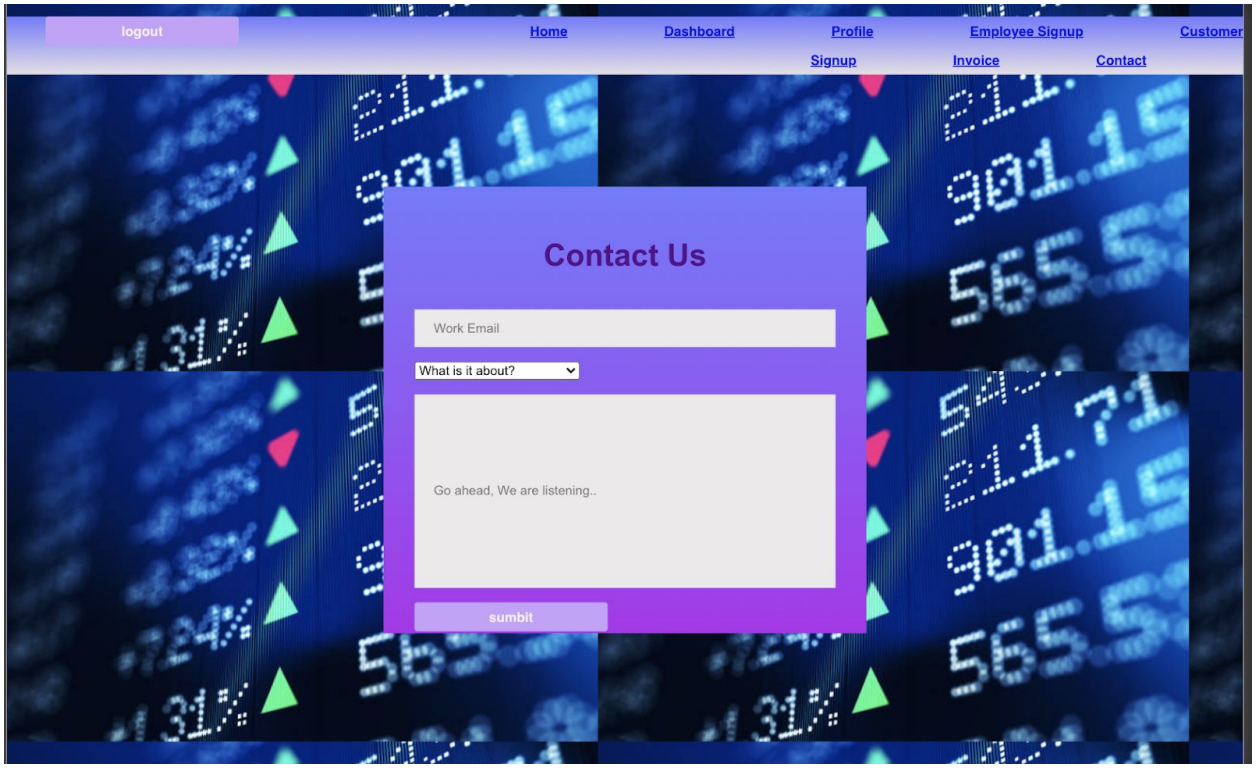
Order Modal



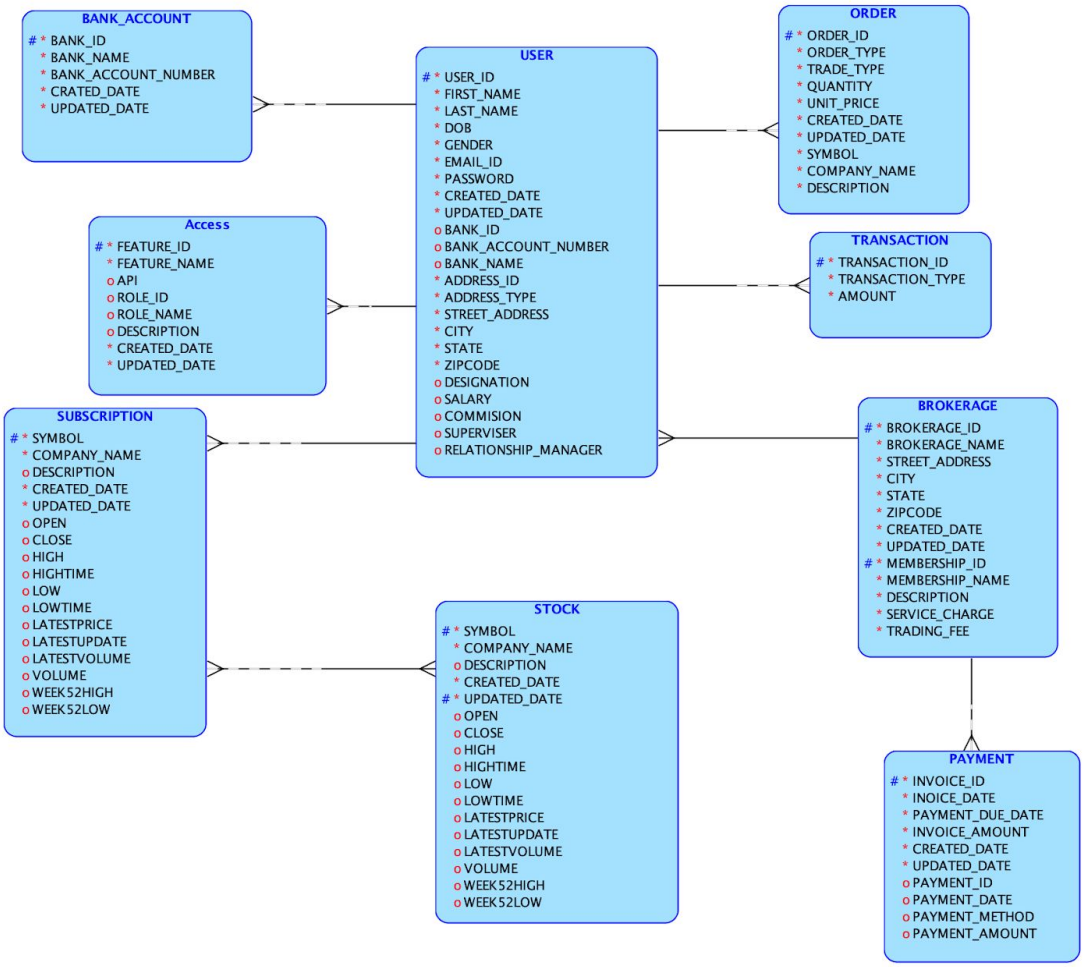
Transaction Page



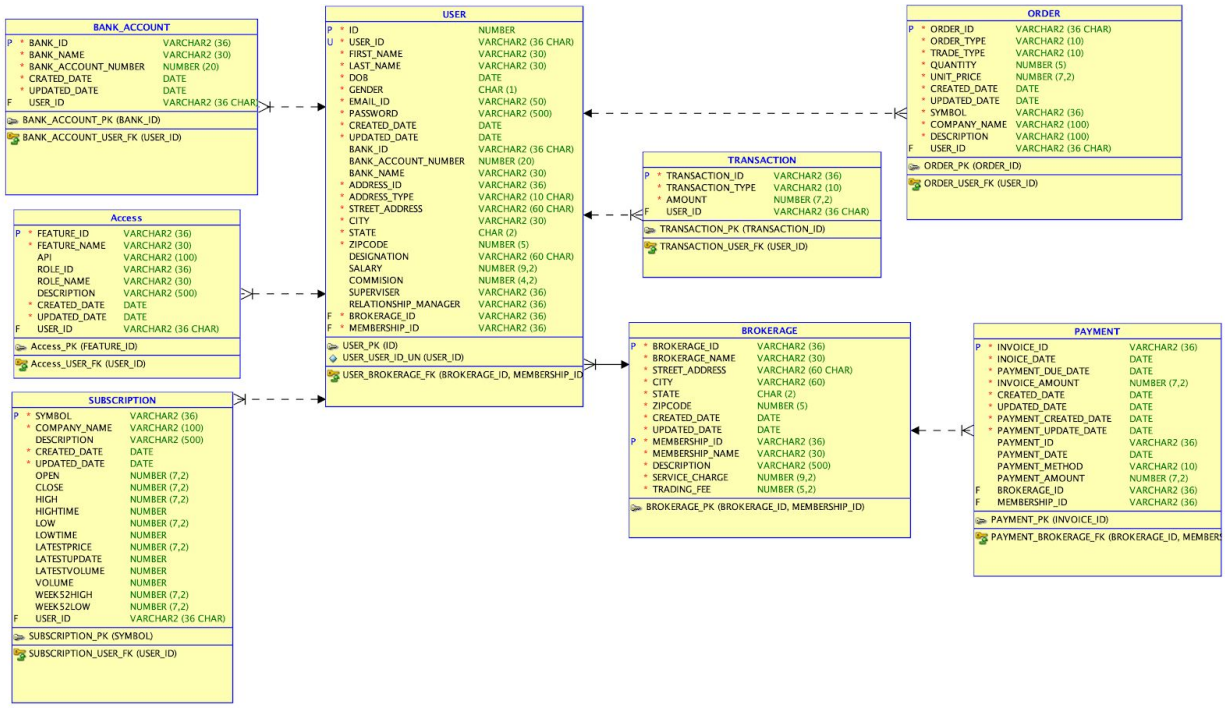
Contact Us



N) DW logical and Relational model



Logical Model



Relational Model

O) Brief summary of ETL approach

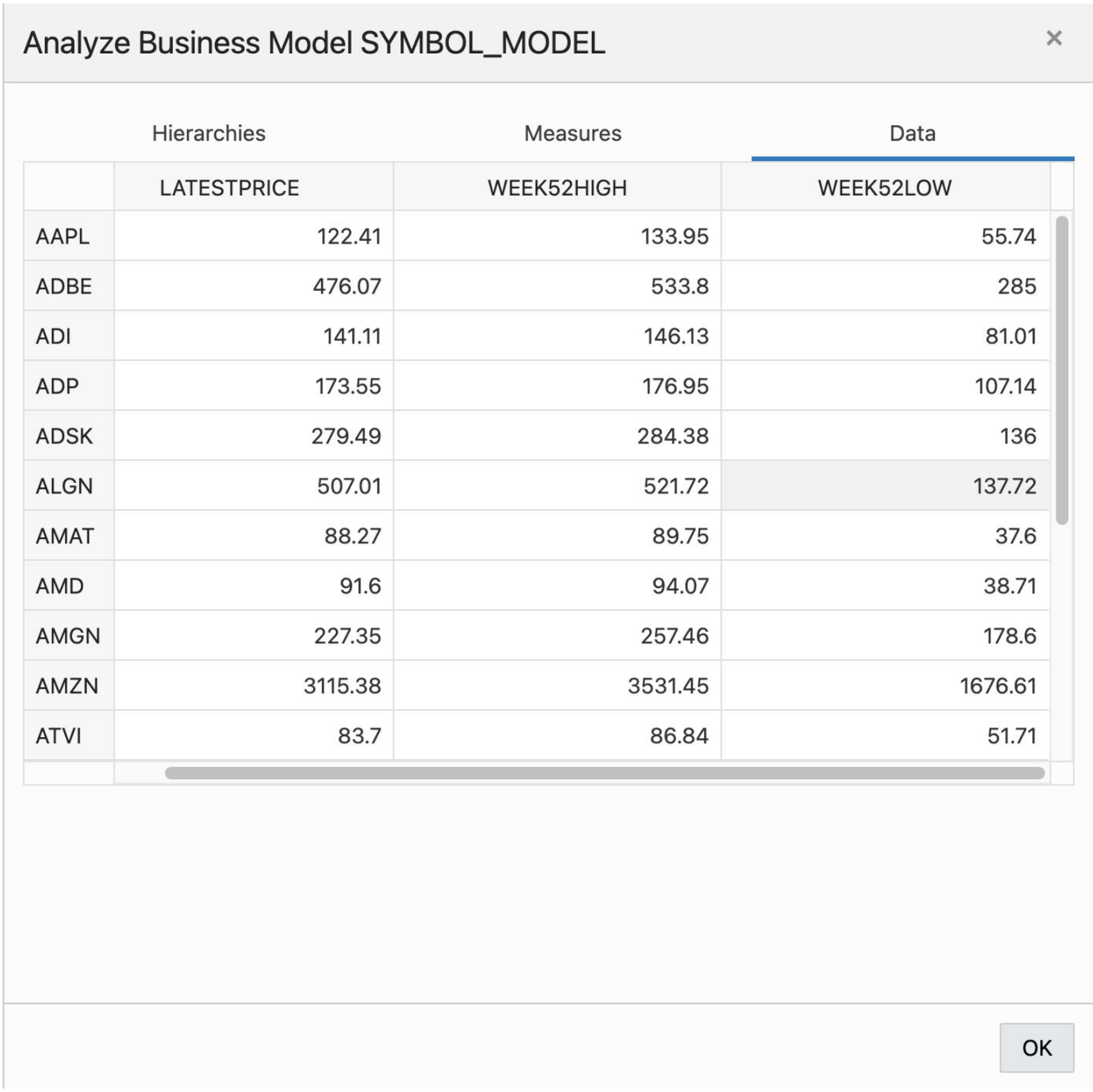
For creating the data warehouse and loading the data into the warehouse we did the following-

- We have used an Oracle autonomous database for data warehouse.
- To extract the data from OLTP we created three procedures, each for extraction of users, orders and subscription. For extracting data for the data warehouse users table we have joined user, user_bank, bank, address, employee, customer and brokerage table. For extracting data for the data warehouse order table we have joined order_history and stock table. extracting data for the data warehouse subscription table we have joined the subscription and stock table.
- The extracted files are stored in the var/lib/mysql-files folder of EC2.
- We have used a few scripts to copy the extracted files from EC2 to s3.
  aws s3 sync /var/lib/mysql-files s3://ec2-mysql-data-dump
  rm /var/lib/mysql-files/*.csv
- After files are stored in the bucket in s3, we downloaded those files.
- These files are used to load data into a stage table at oracle autonomous database where our data warehouse is located.
- In OAD we have created the stage table for corresponding data warehouse tables and we load the data from CSVs into staging tables, OAD provides a drag

and drop facility for loading the data from local system to database and then we have to map the fields of CSV to columns of tables.

- Once the data is available in the stage table, we run the procedures which we created for load from stage table to warehouse table.

P) SQL and data analysis from DW systems, Reports/Charts [screenshot, with brief explanation about purpose of query and reports



	LATESTPRICE	WEEK52HIGH	WEEK52LOW
AAPL	122.41	133.95	55.74
ADBE	476.07	533.8	285
ADI	141.11	146.13	81.01
ADP	173.55	176.95	107.14
ADSK	279.49	284.38	136
ALGN	507.01	521.72	137.72
AMAT	88.27	89.75	37.6
AMD	91.6	94.07	38.71
AMGN	227.35	257.46	178.6
AMZN	3115.38	3531.45	1676.61
ATVI	83.7	86.84	51.71

Here we are analyzing average price, maximum week52high, minimum week52low for all the stocks.

Q) Lesson learned (summary of what have you learned from the project, what went well, what did not go well and why)

- We learned about stocks and stock trading
- Learned how to design Data Warehouse and implement data warehouse
- Create history table, partition table and external table

- All the processes involved in ETL, extraction of data from OLTP database, transforming that data and then loading it into data warehouse tables.
- We learned about Database Servers, how to instantiate a server and create the schema
- AWS
- We learned about Oracle autonomous database for creating our data warehouse and explored other features of it for creating reports and analytics
- During this project we also got to explore React.js and implement our UI in react.
- Project Management
- Most of it all we learned how to manage our time and resources, coordinate with team members to work towards a common goal and achieve our aim.

What went well and what did not

- Everything went well except for one part while extracting the csv from OLTP database. We didn't have enough idea about the access system of EC2 and the scripts used to access certain locations on the server and copy data from there. And as a result of which our OLTP db crashed and we had to create the database again from scratch.

R) Appendix

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

```
CREATE DATABASE sbm;
use sbm;
CREATE TABLE user (
    user_id             VARCHAR(36) NOT NULL,
    user_type                       CHAR(1) NOT NULL,
    first_name          VARCHAR(30) NOT NULL,
    last_name           VARCHAR(30) NOT NULL,
    dob                 DATE NOT NULL,
    gender              CHAR(1) NOT NULL,
    email_id            VARCHAR(50) NOT NULL,
    password            VARCHAR(500) NOT NULL,
    created_date        DATETIME NOT NULL,
```

```
    updated_date          DATETIME NOT NULL,
    brokerage_id          VARCHAR(36) NOT NULL
);
CREATE TABLE bank (
          bank_id             VARCHAR(36) NOT NULL,
    bank_name             VARCHAR(30) NOT NULL,
          created_date        DATETIME NOT NULL,
    updated_date          DATETIME NOT NULL
);
CREATE TABLE user_bank (
          user_id             VARCHAR(36) NOT NULL,
          bank_id             VARCHAR(36) NOT NULL,
    bank_account_number   DECIMAL(20,0) NOT NULL,
          created_date        DATETIME NOT NULL,
    updated_date          DATETIME NOT NULL
);
CREATE TABLE address (
          user_id             VARCHAR(36) NOT NULL,
          address_id          VARCHAR(36) NOT NULL,
    address_type          VARCHAR(10) NOT NULL,
    street_address        VARCHAR(60) NOT NULL,
    city                  VARCHAR(30) NOT NULL,
    state                 CHAR(2) NOT NULL,
    zipcode               INT NOT NULL,
    created_date          DATETIME NOT NULL,
    updated_date          DATETIME NOT NULL
);
CREATE TABLE employee (
          user_id             VARCHAR(36) NOT NULL,
    designation           VARCHAR(60) NOT NULL,
    salary                DECIMAL(9, 2),
    commision             DECIMAL(4, 2),
    superviser            VARCHAR(36),
    employee_id INT
);

CREATE TABLE customer (
user_id VARCHAR(36), relationship_manager VARCHAR(36), customer_id INT
);
```

```
CREATE TABLE stock(
symbol VARCHAR(36), companyName VARCHAR(100), description TEXT,
open DECIMAL(7,2),
close DECIMAL(7,2),
high DECIMAL(7,2),
highTime BIGINT,
low DECIMAL(7,2),
lowTime BIGINT,
latestPrice DECIMAL(7,2), latestUpdate BIGINT, latestVolume BIGINT,
volume BIGINT,
week52High DECIMAL(7,2), week52Low DECIMAL(7,2),
created_date DATETIME, updated_date DATETIME
);
CREATE TABLE subscription (
user_id VARCHAR(36), symbol VARCHAR(36),
created_date DATETIME,
updated_date DATETIME
);

CREATE TABLE  order_history(
order_id VARCHAR(36), trade_type VARCHAR(10), order_type VARCHAR(10), user_id
VARCHAR(36), symbol VARCHAR(36), quantity INT,
unit_price DECIMAL(7,2), order_start_time DATETIME, order_end_time DATETIME,
order_status VARCHAR(36), created_date DATETIME, updated_date DATETIME
);
CREATE TABLE  brokerage (
brokerage_id VARCHAR(36), brokerage_name VARCHAR(30), street_address
VARCHAR(60), city VARCHAR(30),
state CHAR(2),
zipcode INT,
membership_id VARCHAR(36), created_date DATETIME, updated_date DATETIME
);
CREATE TABLE  membership(
membership_id VARCHAR(36), membership_type VARCHAR(30), description
VARCHAR(500), service_charge DECIMAL(9,2), trading_fee DECIMAL(5,2),
created_date DATETIME, updated_date DATETIME
);
CREATE TABLE  invoice(
```

```
brokerage_id VARCHAR(36), invoice_id VARCHAR(36), invoice_date DATETIME,
payment_due_date DATETIME, invoice_amount DECIMAL(7,2), created_date
DATETIME, updated_date DATETIME
);
CREATE TABLE  payment_history(
invoice_id VARCHAR(36), payment_id VARCHAR(36), payment_date DATETIME,
payment_method VARCHAR(10), created_date DATETIME, updated_date DATETIME,
amount DECIMAL(7,2)
);
CREATE TABLE  feature(
feature_id VARCHAR(36), feature_name VARCHAR(30), api VARCHAR(100),
created_date DATETIME, updated_date DATETIME
);
CREATE TABLE  access(
feature_id VARCHAR(36), role_id VARCHAR(36),
created_date DATETIME,
updated_date DATETIME
);
CREATE TABLE  email_log (
        log_id VARCHAR(36),
   to_email VARCHAR(50),
   subject VARCHAR(100),
   body TEXT,
   created_date DATETIME,
        updated_date DATETIME
);
CREATE TABLE  role(
role_id VARCHAR(36), role_name VARCHAR(30), description VARCHAR(500),
created_date DATETIME, updated_date DATETIME
);
CREATE TABLE  user_role(
user_id VARCHAR(36), role_id VARCHAR(36),
updated_date DATETIME,
created_date DATETIME
);
CREATE TABLE  transaction_history(
transaction_id VARCHAR(36), user_id VARCHAR(36), transaction_type
VARCHAR(10), amount DECIMAL(7,2), created_date DATETIME, updated_date
DATETIME
);
```

```
CREATE TABLE authentication (
        authentication_id VARCHAR(36),
        user_id VARCHAR(36),
    updated_date DATETIME,
        created_date DATETIME
);



ALTER TABLE `sbm`.`user`
CHANGE COLUMN `brokerage_id` `brokerage_id` VARCHAR(36) NOT NULL ;
ALTER TABLE `sbm`.`user`
CHANGE COLUMN `user_id` `user_id` VARCHAR(36) NOT NULL ;

ALTER TABLE `sbm`.`document`
DROP FOREIGN KEY `document_user_fk`;
ALTER TABLE `sbm`.`email_log`
DROP FOREIGN KEY `email_log_user_fk`;
ALTER TABLE `sbm`.`employee`
DROP FOREIGN KEY `employee_user_fk`;
ALTER TABLE `sbm`.`invoice`
DROP FOREIGN KEY `invoice_brokerage_fk`;
ALTER TABLE `sbm`.`order_history`
DROP FOREIGN KEY `order_history_stock_price_fk`,
DROP FOREIGN KEY `order_history_user_fk`;
ALTER TABLE `sbm`.`payment_history`
DROP FOREIGN KEY `payment_history_invoice_fk`;
ALTER TABLE `sbm`.`subscription`
DROP FOREIGN KEY `subscription_stock_price_fk`,
DROP FOREIGN KEY `subscription_user_fk`;
ALTER TABLE `sbm`.`user_bank`
DROP FOREIGN KEY `user_bank_bank_fk`,
DROP FOREIGN KEY `user_bank_user_fk`;
ALTER TABLE `sbm`.`user_role`
DROP FOREIGN KEY `user_role_role_fk`,
DROP FOREIGN KEY `user_role_user_fk`;
ALTER TABLE `sbm`.`access`
DROP FOREIGN KEY `access_feature_fk`,
DROP FOREIGN KEY `access_role_fk`;
ALTER TABLE `sbm`.`address`
```

```
DROP FOREIGN KEY `address_user_fk`;
ALTER TABLE `sbm`.`brokerage`
DROP FOREIGN KEY `brokerage_membership_fk`;
ALTER TABLE `sbm`.`authentication`
DROP FOREIGN KEY `authentication_user_fk`;
ALTER TABLE `sbm`.`customer`
DROP FOREIGN KEY `customer_user_fk`;
ALTER TABLE `sbm`.`user`
DROP FOREIGN KEY `user_brokerage_fk`;


ALTER TABLE `sbm`.`access`
ADD CONSTRAINT `access_feature_fk`
  FOREIGN KEY (`feature_id`)
  REFERENCES `sbm`.`feature` (`feature_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
ADD CONSTRAINT `access_role_fk`
  FOREIGN KEY (`role_id`)
  REFERENCES `sbm`.`role` (`role_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;

ALTER TABLE `sbm`.`user`
ADD CONSTRAINT `user_brokerage_fk`
  FOREIGN KEY (`brokerage_id`)
  REFERENCES `sbm`.`brokerage` (`brokerage_id`)
        ON DELETE RESTRICT
        ON UPDATE RESTRICT;

ALTER TABLE `sbm`.`address`
ADD CONSTRAINT `address_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
        ON DELETE CASCADE
        ON UPDATE CASCADE;


ALTER TABLE `sbm`.`brokerage`
ADD CONSTRAINT `brokerage_membership_fk`
```

```
  FOREIGN KEY (`membership_id`)
  REFERENCES `sbm`.`membership` (`membership_id`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT;


ALTER TABLE `sbm`.`authentication`
ADD CONSTRAINT `authentication_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;


ALTER TABLE `sbm`.`customer`
ADD CONSTRAINT `customer_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;


ALTER TABLE `sbm`.`document`
ADD CONSTRAINT `document_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;


ALTER TABLE `sbm`.`email_log`
ADD CONSTRAINT `email_log_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
ADD CONSTRAINT `email_log_email_template_fk`
  FOREIGN KEY (`template_id`)
  REFERENCES `sbm`.`email_template` (`template_id`)
  ON DELETE RESTRICT
```

```
  ON UPDATE RESTRICT;


ALTER TABLE `sbm`.`employee`
CHANGE COLUMN `user_id` `user_id` VARCHAR(36) NOT NULL ;
ALTER TABLE `sbm`.`employee`
ADD CONSTRAINT `employee_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;


ALTER TABLE `sbm`.`invoice`
ADD CONSTRAINT `invoice_brokerage_fk`
  FOREIGN KEY (`brokerage_id`)
  REFERENCES `sbm`.`brokerage` (`brokerage_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;



ALTER TABLE `sbm`.`order_history`
ADD CONSTRAINT `order_history_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
ALTER TABLE `sbm`.`order_history`
ADD CONSTRAINT `order_history_stock_fk`
  FOREIGN KEY (`symbol`)
  REFERENCES `sbm`.`stock` (`symbol`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT;

ALTER TABLE `sbm`.`payment_history`
ADD CONSTRAINT `payment_history_invoice_fk`
  FOREIGN KEY (`invoice_id`)
  REFERENCES `sbm`.`invoice` (`invoice_id`)
  ON DELETE CASCADE
```

```
  ON UPDATE CASCADE;


ALTER TABLE `sbm`.`subscription`
ADD CONSTRAINT `subscription_stock_fk`
  FOREIGN KEY (`symbol`)
  REFERENCES `sbm`.`stock` (`symbol`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
ADD CONSTRAINT `subscription_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;


ALTER TABLE `sbm`.`user_bank`
ADD CONSTRAINT `user_bank_bank_fk`
  FOREIGN KEY (`bank_id`)
  REFERENCES `sbm`.`bank` (`bank_id`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT,
ADD CONSTRAINT `user_bank_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;


ALTER TABLE `sbm`.`transaction_history`
ADD CONSTRAINT `transaction_history_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;

ALTER TABLE `sbm`.`employee`
ADD CONSTRAINT `employee_superviser_fk`
  FOREIGN KEY (`superviser`)
  REFERENCES `sbm`.`user` (`user_id`)
```

```
  ON DELETE SET NULL
  ON UPDATE SET NULL;

ALTER TABLE `sbm`.`employee`
DROP FOREIGN KEY `employee_superviser_fk`;


ALTER TABLE `sbm`.`customer`
ADD CONSTRAINT `customer_manager_fk`
  FOREIGN KEY (`relationship_manager`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE SET NULL
  ON UPDATE SET NULL;

ALTER TABLE `sbm`.`user_role`
ADD CONSTRAINT `user_role_role_fk`
  FOREIGN KEY (`role_id`)
  REFERENCES `sbm`.`role` (`role_id`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT,
ADD CONSTRAINT `user_role_user_fk`
  FOREIGN KEY (`user_id`)
  REFERENCES `sbm`.`user` (`user_id`)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
```

Triggers

```
CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`address_BEFORE_INSERT` BEFORE INSERT ON `address` FOR EACH ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`address_BEFORE_UPDATE` BEFORE UPDATE ON `address` FOR EACH
ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;
```

```
CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`access_BEFORE_INSERT`
BEFORE INSERT ON `access_` FOR EACH ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`access__BEFORE_UPDATE` BEFORE UPDATE ON `access_` FOR EACH
ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`feature_BEFORE_INSERT`
BEFORE INSERT ON `feature` FOR EACH ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`feature_BEFORE_UPDATE` BEFORE UPDATE ON `feature` FOR EACH ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`role_BEFORE_INSERT`
BEFORE INSERT ON `role` FOR EACH ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`role_BEFORE_UPDATE`
BEFORE UPDATE ON `role` FOR EACH ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`user_role_BEFORE_INSERT` BEFORE INSERT ON `user_role` FOR EACH
ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`user_role_BEFORE_UPDATE` BEFORE UPDATE ON `user_role` FOR EACH
ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;
```

```
CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`transaction_history_BEFORE_INSERT` BEFORE INSERT ON
`transaction_history` FOR EACH ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`transaction_history_BEFORE_UPDATE` BEFORE UPDATE ON
`transaction_history` FOR EACH ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`bank_BEFORE_INSERT`
BEFORE INSERT ON `bank` FOR EACH ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`bank_BEFORE_UPDATE`
BEFORE UPDATE ON `bank` FOR EACH ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`brokerage_BEFORE_INSERT` BEFORE INSERT ON `brokerage` FOR EACH
ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`brokerage_BEFORE_UPDATE` BEFORE UPDATE ON `brokerage` FOR EACH
ROW
        SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`user_BEFORE_INSERT`
BEFORE INSERT ON `user` FOR EACH ROW
        SET NEW.created_date = CURRENT_TIMESTAMP,
        NEW.updated_date = CURRENT_TIMESTAMP;
```

```
CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`user_BEFORE_UPDATE`
BEFORE UPDATE ON `user` FOR EACH ROW
    SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`email_log_BEFORE_INSERT` BEFORE INSERT ON `email_log` FOR EACH
ROW
    SET NEW.created_date = CURRENT_TIMESTAMP,
    NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`email_log_BEFORE_UPDATE` BEFORE UPDATE ON `email_log` FOR EACH
ROW
    SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`authentication_BEFORE_INSERT` BEFORE INSERT ON `authentication` FOR
EACH ROW
    SET NEW.created_date = CURRENT_TIMESTAMP,
    NEW.updated_date = CURRENT_TIMESTAMP;

CREATE DEFINER = CURRENT_USER TRIGGER
`sbm`.`authentication_BEFORE_UPDATE` BEFORE UPDATE ON `authentication`
FOR EACH ROW
    SET NEW.updated_date = CURRENT_TIMESTAMP;


CREATE DEFINER = CURRENT_USER TRIGGER `sbm`.`stock_BEFORE_INSERT`
BEFORE INSERT ON `stock` FOR EACH ROW
    SET NEW.created_date = CURRENT_TIMESTAMP,
    NEW.updated_date = CURRENT_TIMESTAMP;
```

Events

```
CREATE EVENT IF NOT EXISTS `Clean_Older_Than_24_Hours`
ON SCHEDULE
  EVERY 1 HOUR
```

```
  COMMENT 'Clean up expired sessions'
  DO
    DELETE FROM authentication
    WHERE login_time < TIMESTAMP(DATE_SUB(NOW(), INTERVAL 24 hour));

CREATE EVENT IF NOT EXISTS `Delete_OTP_In_5_Min`
ON SCHEDULE
  EVERY 5 MINUTE
  COMMENT 'Clean up expired OTP'
  DO
    DELETE FROM otp
    WHERE updated_date < TIMESTAMP(DATE_SUB(NOW(), INTERVAL 15 minute));
```

History Table

```
CREATE TABLE stock_history AS SELECT * FROM stock WHERE 1=2;
        select * from order_history where trade_type = 'INTRADAY' and
DATE_FORMAT(order_end_time, '%Y-%m-%d') =
DATE_FORMAT(CURRENT_TIMESTAMP, '%Y-%m-%d');
```

Stored Procedures

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_authentication_details`(IN
authentication_id_in VARCHAR(36),IN user_id_in VARCHAR(36))
BEGIN
        INSERT INTO authentication(authentication_id,user_id)
values(authentication_id_in,user_id_in);
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_email_log`(IN to_email_in
VARCHAR(50), IN subject_in VARCHAR(100), IN body_in TEXT, IN user_id_in
VARCHAR(36))
BEGIN
        insert into email_log(to_email,subject,body,user_id)
    VALUES(to_email_in,subject_in,body_in,user_id_in);
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`add_intraday_squareoff_order`(IN order_id_in VARCHAR(36), IN order_type_in
VARCHAR(10),
IN user_id_in VARCHAR(36), IN symbol_in VARCHAR(36), IN quantity_in INT, IN
order_status_in VARCHAR(36))
BEGIN
        INSERT INTO
order_history(order_id,trade_type,order_type,user_id,symbol,quantity,unit_price,
   order_start_time,order_end_time,order_status)
   VALUE(order_id_in, "INTRADAY", order_type_in, user_id_in, symbol_in, quantity_in,
   (SELECT latestPrice FROM stock WHERE symbol = symbol_in),
CURRENT_TIMESTAMP, CURRENT_TIMESTAMP,order_status_in);
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_new_user`(IN user_id_in
VARCHAR(36), IN user_type_in CHAR(1), IN first_name_in VARCHAR(30),
IN last_name_in VARCHAR(30), IN dob_in DATE, IN gender_in CHAR(1), IN
email_id_in VARCHAR(50), IN password_in VARCHAR(500),
IN brokerage_id_in VARCHAR(36), IN brokerage_name_in VARCHAR(30), IN
membership_type_in VARCHAR(30), IN relationship_manager_in INT,
IN designation_in VARCHAR(60), IN salary_in DECIMAL(9,2), IN commision_in
DECIMAL(4,2), IN superviser_in INT, IN address_id_in1 VARCHAR(36),
 IN street_address_in1 VARCHAR(60), IN city_in1 VARCHAR(30), IN state_in1
CHAR(2), IN zipcode_in1 INT, IN address_id_in2 VARCHAR(36),
 IN street_address_in2 VARCHAR(60), IN city_in2 VARCHAR(30), IN state_in2
CHAR(2), IN zipcode_in2 INT, IN bank_id_in VARCHAR(36), IN bank_account_number
DECIMAL(20,0))
BEGIN
        DECLARE exit handler for sqlexception
                BEGIN
                        ROLLBACK;
                        RESIGNAL;
                END;
        START TRANSACTION;
```

```
        IF user_type_in = 'A' THEN
                BEGIN
                        INSERT INTO
brokerage(brokerage_id,brokerage_name,membership_id,street_address,city,state,zipc
ode)
        VALUES(brokerage_id_in,brokerage_name_in,(SELECT m.membership_id
FROM membership m WHERE m.membership_type = membership_type_in),
        street_address_in1,city_in1,state_in1,zipcode_in1);
                        INSERT INTO
user(user_id,user_type,first_name,last_name,dob,gender,email_id,password,brokerage
_id)

VALUES(user_id_in,user_type_in,first_name_in,last_name_in,dob_in,gender_in,email_i
d_in,password_in,brokerage_id_in);
        INSERT INTO employee(user_id,designation,salary,commision)
        VALUES(user_id_in,"ADMIN",salary_in,commision_in);
                        INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in1,"PRIMARY",street_address_in1,city_in1,state_in1,
zipcode_in1);
                        INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in2,"SECONDARY",street_address_in2,city_in2,state_i
n2,zipcode_in2);
        INSERT INTO user_bank(user_id,bank_id,bank_account_number)
        VALUES(user_id_in,bank_id_in,bank_account_number);
        INSERT INTO user_role(user_id,role_id)
        VALUES(user_id_in,'4480ac51-9428-40af-882d-e0cdfd549a1a');
      END;
   ELSEIF user_type_in = 'E' THEN
                BEGIN
                        INSERT INTO
user(user_id,user_type,first_name,last_name,dob,gender,email_id,password,brokerage
_id)

VALUES(user_id_in,user_type_in,first_name_in,last_name_in,dob_in,gender_in,email_i
d_in,password_in,brokerage_id_in);
        INSERT INTO employee(user_id,designation,salary,commision,superviser)
```

```
        VALUES(user_id_in,designation_in,salary_in,commision_in,(SELECT e.user_id
from employee e where e.employee_id = superviser_in));
                        INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in1,"PRIMARY",street_address_in1,city_in1,state_in1,
zipcode_in1);
                        INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in2,"SECONDARY",street_address_in2,city_in2,state_i
n2,zipcode_in2);
                        INSERT INTO user_role(user_id,role_id)
        VALUES(user_id_in,'07615f51-9a9c-4ab8-bbcb-d10730017521');
      END;
        ELSEIF user_type_in = 'C' THEN
              BEGIN
                        IF relationship_manager_in = 0 THEN
                              BEGIN
                                    INSERT INTO
user(user_id,user_type,first_name,last_name,dob,gender,email_id,password,brokerage
_id)

VALUES(user_id_in,user_type_in,first_name_in,last_name_in,dob_in,gender_in,email_i
d_in,password_in,"676ecd2a-0098-4380-a430-9f510e548144");
                                    INSERT INTO customer(user_id)
                                    VALUES(user_id_in);
                                    INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in1,"PRIMARY",street_address_in1,city_in1,state_in1,
zipcode_in1);
                                    INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in2,"SECONDARY",street_address_in2,city_in2,state_i
n2,zipcode_in2);
                                    INSERT INTO
user_bank(user_id,bank_id,bank_account_number)
```

```
VALUES(user_id_in,bank_id_in,bank_account_number);
                                        INSERT INTO user_role(user_id,role_id)

VALUES(user_id_in,'958c3e84-a01b-46c9-93bf-2e314f2d8186');
            END;
                        ELSE
                                BEGIN
                                        INSERT INTO
user(user_id,user_type,first_name,last_name,dob,gender,email_id,password,brokerage
_id)

VALUES(user_id_in,user_type_in,first_name_in,last_name_in,dob_in,gender_in,email_i
d_in,password_in,brokerage_id_in);
                                        INSERT INTO
customer(user_id,relationship_manager)
                                        VALUES(user_id_in,(SELECT user_id from employee
where employee_id = relationship_manager_in));
                                        INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in1,"PRIMARY",street_address_in1,city_in1,state_in1,
zipcode_in1);
                                        INSERT INTO
address(user_id,address_id,address_type,street_address,city,state,zipcode)

VALUES(user_id_in,address_id_in2,"SECONDARY",street_address_in2,city_in2,state_i
n2,zipcode_in2);
                                        INSERT INTO
user_bank(user_id,bank_id,bank_account_number)

VALUES(user_id_in,bank_id_in,bank_account_number);
                                        INSERT INTO user_role(user_id,role_id)

VALUES(user_id_in,'958c3e84-a01b-46c9-93bf-2e314f2d8186');
            END;
                        END IF;
                END;
    END IF;
    COMMIT;
```

```
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `add_order`(IN order_id_in
VARCHAR(36), IN trade_type_in VARCHAR(10), IN order_type_in VARCHAR(10),
IN user_id_in VARCHAR(36), IN symbol_in VARCHAR(36), IN quantity_in INT, IN
unit_price_in DECIMAL(7,2), IN order_status_in VARCHAR(36))
BEGIN
	INSERT INTO
order_history(order_id,trade_type,order_type,user_id,symbol,quantity,unit_price,
   order_start_time,order_end_time,order_status)
   VALUE(order_id_in, trade_type_in, order_type_in, user_id_in, symbol_in, quantity_in,
   unit_price_in, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP,order_status_in);
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `add_otp`(IN email_id_in
VARCHAR(50), IN otp_in INT)
BEGIN
	INSERT INTO otp(email_id, otp) VALUES(email_id_in,otp_in)
   ON DUPLICATE KEY UPDATE otp = otp_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `add_stock_data`(IN symbol_in
VARCHAR(36), IN companyName_in VARCHAR(100), IN open_in DECIMAL(7,2),
IN close_in DECIMAL(7,2), IN high_in DECIMAL(7,2), IN highTime_in BIGINT, IN
low_in DECIMAL(7,2), IN lowTime_in BIGINT,
IN latestPrice_in DECIMAL(7,2), IN latestUpdate_in BIGINT, IN latestVolume_in
BIGINT, IN volume_in BIGINT,
IN week52High_in DECIMAL(7,2),IN week52Low_in DECIMAL(7,2))
BEGIN
	INSERT INTO
stock(symbol,companyName,open,close,high,highTime,low,lowTime,latestPrice,
   latestUpdate,latestVolume, volume,week52High,week52Low)

VALUES(symbol_in,companyName_in,open_in,close_in,high_in,highTime_in,low_in,lo
wTime_in,latestPrice_in,
   latestUpdate_in,latestVolume_in,volume_in,week52High_in,week52Low_in)
   ON DUPLICATE KEY UPDATE
```

```
    open = open_in,
    close = close_in,
    high = high_in,
    highTime = highTime_in,
    low = low_in,
    lowTime = lowTime_in,
    latestPrice = latestPrice_in,
    latestUpdate = latestUpdate_in,
    latestVolume = latestVolume_in,
    volume = volume_in,
    week52High = week52High_in,
    week52Low = week52Low_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `add_transaction`(IN
transaction_id_in VARCHAR(36),IN user_id_in VARCHAR(36),IN transaction_type_in
VARCHAR(10),
        IN amount_in DECIMAL(7,2))
BEGIN
        INSERT INTO
transaction_history(transaction_id,user_id,transaction_type,amount)
    VALUE(transaction_id_in,user_id_in,transaction_type_in,amount_in);
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_address_by_userid`(IN
user_id_in VARCHAR(36))
BEGIN
        SELECT * FROM address WHERE user_id = user_id_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE
`get_all_customer_in_brokerage`(IN brokerage_id_in VARCHAR(36))
BEGIN
        SELECT u.user_id as user_id, u.first_name as first_name, u.last_name as
last_name, u.dob as dob,
    u.gender as gender, u.email_id as email_id, e.employee_id as
relationship_manager_id, c.customer_id as customer_id
```

```
    FROM user u
    JOIN customer c ON u.user_id = c.user_id
    LEFT JOIN employee e ON c.relationship_manager = e.user_id
    WHERE brokerage_id = brokerage_id_in AND user_type = 'C';
END


CREATE DEFINER=`root`@`localhost` PROCEDURE
`get_all_employee_in_brokerage`(IN brokerage_id_in VARCHAR(36))
BEGIN
        SELECT u.user_id as user_id, u.first_name as first_name, u.last_name as
last_name, e.designation as designation,
    u.gender as gender, u.email_id as email_id, s.employee_id as superviser_id,
e.employee_id as employee_id
    FROM user u
        JOIN employee e ON u.user_id = e.user_id
    LEFT JOIN employee s ON e.superviser = s.user_id
    WHERE brokerage_id = brokerage_id_in AND (user_type = 'E' OR user_type = 'A');
END



CREATE DEFINER=`root`@`localhost` PROCEDURE `get_all_intraday_orders`(IN
date_in DATETIME)
BEGIN
        select * from order_history where trade_type = 'INTRADAY' and
DATE_FORMAT(order_end_time, '%Y-%m-%d') = DATE_FORMAT(date_in,
'%Y-%m-%d');
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_authentication_details`(IN
authKey VARCHAR(36))
BEGIN
        SELECT * FROM authentication WHERE authentication_id = authKey;
    UPDATE authentication SET updated_date = CURRENT_TIMESTAMP
    WHERE authentication_id = authKey;
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_brokerage_data`(IN
brokerage_id_in VARCHAR(36))
BEGIN
	SELECT * FROM brokerage b
   LEFT JOIN membership m ON b.membership_id = m.membership_id
   WHERE brokerage_id = brokerage_id_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE
`get_completed_order_history`(IN user_id_in VARCHAR(36))
BEGIN
	SELECT * FROM order_history
   WHERE user_id = user_id_in AND order_status = 'COMPLETE';
END


CREATE DEFINER=`root`@`localhost` PROCEDURE
`get_completed_order_history_with_user_id_and_symbol`(IN user_id_in
VARCHAR(36),IN symbol_in VARCHAR(36))
BEGIN
	SELECT * FROM order_history
   WHERE user_id = user_id_in AND  symbol = symbol_in AND order_status =
'COMPLETE';
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_customer_by_user_id`(IN
user_id_in VARCHAR(36))
BEGIN
	SELECT * FROM customer WHERE user_id = user_id_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_employee_by_user_id`(IN
user_id_in VARCHAR(36))
BEGIN
	SELECT * FROM employee WHERE user_id = user_id_in;
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_invoice`(IN invoice_id_in
VARCHAR(36))
BEGIN
	SELECT * FROM invoice
   LEFT JOIN payment_history ON invoice.invoice_id = payment_history.invoice_id
   WHERE invoice.invoice_id = invoice_id_in;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `get_invoice_list`(IN
brokerage_id_in VARCHAR(36))
BEGIN
	SELECT * FROM invoice
   LEFT JOIN payment_history ON invoice.invoice_id = payment_history.invoice_id
   WHERE invoice.brokerage_id = brokerage_id_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_order_history`(IN user_id_in
VARCHAR(36))
BEGIN
	SELECT * FROM order_history WHERE user_id = user_id_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_otp`(IN email_id_in
VARCHAR(36))
BEGIN
	SELECT * FROM otp WHERE email_id = email_id_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_stock_data`(IN symbol_in
VARCHAR(36))
BEGIN
	SELECT stock.symbol as symbol, stock.companyName as companyName,
stock.latestPrice as latestPrice
	FROM stock
   WHERE symbol = symbol_in;
```

```
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_subscription_list`(IN
user_id_in VARCHAR(36))
BEGIN
	SELECT stock.symbol as symbol, stock.companyName as companyName,
stock.latestPrice as latestPrice ,
	case when subscription.symbol IS NULL THEN false else true end as subscribed
	FROM stock
	LEFT Join subscription on stock.symbol = subscription.symbol and
subscription.user_id = user_id_in AND delete_flag = FALSE;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_transaction_history`(IN
user_id_in VARCHAR(36))
BEGIN
	SELECT * FROM transaction_history WHERE user_id = user_id_in;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `get_user_by_email_id`(IN
email_id_in VARCHAR(50))
BEGIN
	SELECT * FROM user where email_id = email_id_in;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `get_user_by_employee_id`(IN
employee_id_in INT)
BEGIN
	SELECT * FROM employee e
   JOIN user u on e.user_id = u.user_id
   where e.employee_id = employee_id_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `get_user_details`(IN user_id_in
VARCHAR(36))
BEGIN
	SELECT * FROM user WHERE user_id = user_id_in;
```

```
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `loop`()
BEGIN
        declare max int unsigned default 10 + FLOOR(RAND() * 10);
        declare count int unsigned default 0;
   declare empcount int unsigned default 0;
   declare supervisor int unsigned default 0;
   declare RELMANAGER int unsigned default 0;
   declare adminid varchar(36) default uuid();
   declare EMPID varchar(36) default uuid();
   declare userid varchar(36) default uuid();
   declare invoiceid varchar(36) default uuid();
   declare monthn int unsigned default 12;
   declare num int unsigned default 0;
   declare brokrage_id  varchar(36) default uuid();
        DECLARE exit handler for sqlexception
                BEGIN
                        ROLLBACK;
                        RESIGNAL;
                END;
        START TRANSACTION;
   CALL add_new_user(adminid,'A',RandString(5, false),RandString(6,
false),'2002-12-04','M',CONCAT(RandString(10, true),'@mailinator.com'),

'4d63f211c7a9a9497d6504fa9599346626a1b06731fa7463f71b8a68804328c2',brokrage
_id, 'Test Corp', 'SILVER',
   0,'ADMIN',0.0,0.0,0,uuid(),'133 BAY RIDGE AVE APT
1','BROOKLYN','NY',11220,uuid(),'133 BAY RIDGE AVE APT 1','BROOKLYN','NY'
   ,11220,'1',5648756);
   CALL mutiple_subscription(adminid,15);
        while num< monthn do
      set invoiceid = uuid();
                INSERT INTO
invoice(invoice_id,brokerage_id,invoice_date,payment_due_date,invoice_amount)
                SELECT invoiceid,b.brokerage_id, DATE_SUB(CURRENT_TIMESTAMP,
INTERVAL num MONTH), LAST_DAY(DATE_SUB(CURRENT_TIMESTAMP,
INTERVAL num MONTH)), m.service_charge
                FROM brokerage b
```

```
                JOIN membership m ON b.membership_id = m.membership_id
                WHERE m.membership_type <> 'BASIC' AND b.brokerage_id =
brokrage_id;
      IF num > 0 THEN
                        INSERT INTO
payment_history(invoice_id,payment_id,payment_method,amount,payment_date)
                        VALUES(invoiceid, uuid(),'BANK',(SELECT invoice_amount from
invoice where invoice_id =
invoiceid),LAST_DAY(DATE_SUB(CURRENT_TIMESTAMP, INTERVAL num
MONTH)));
      END IF;
      set num = num+1;
   end while;
   set num = 0;
   select employee_id into supervisor from employee where user_id = adminid;
   while empcount < max do
                set empcount=empcount+1;
                set count =0;
      SET EMPID = UUID();
      CALL add_new_user(EMPID,'E',RandString(5, false),RandString(6,
false),'2002-12-04','M',CONCAT(RandString(10, true),'@mailinator.com'),

'4d63f211c7a9a9497d6504fa9599346626a1b06731fa7463f71b8a68804328c2',brokrage
_id,null,null,
      0,'MANAGER',0.0,0.0,supervisor,uuid(),'133 BAY RIDGE AVE APT
1','BROOKLYN','NY',11220,uuid(),'133 BAY RIDGE AVE APT 1','BROOKLYN','NY'
      ,11220,NULL,0);
      CALL mutiple_subscription(EMPID,15);
      select employee_id into RELMANAGER from employee where user_id = EMPID;
                while count < max do
                        set count=count+1;
         set userid = uuid();
                        CALL add_new_user(userid,'C',RandString(5, false),RandString(6,
false),'2002-12-04','M',CONCAT(RandString(10, true),'@mailinator.com'),

'4d63f211c7a9a9497d6504fa9599346626a1b06731fa7463f71b8a68804328c2',brokrage
_id,null,null,
                        RELMANAGER,null,0.0,0.0,0,uuid(),'133 BAY RIDGE AVE APT
1','BROOKLYN','NY',11220,uuid(),'133 BAY RIDGE AVE APT 1','BROOKLYN','NY'
                        ,11220,'1',5648756);
```

```
        CALL mutiple_subscription(userid,15);
        CALL add_transaction(uuid(), userid,'CREDIT',20000 + FLOOR(RAND() *
20000));
        CALL multiple_orders(userid);
       end while;
   end while;
   COMMIT;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `multiple_orders`(IN user_id_in
VARCHAR(36))
BEGIN
       declare length int unsigned default FLOOR(5 + RAND() * 10);
   declare buy int unsigned default FLOOR(1 + RAND() * 5);
   declare sell int unsigned default FLOOR(1 + RAND() * (buy-1));
   declare count int unsigned default 0;
   declare smbl varchar(36) default 'AAPL';
   declare price DECIMAL(7,2) default 121.55;
   while count < length do
               set smbl = (SELECT symbol FROM stock ORDER BY RAND() LIMIT 1);
      set price = (SELECT latestPrice FROM stock s where s.symbol = smbl);
      set buy = FLOOR(1 + RAND() * 5);
      set sell = FLOOR(1 + RAND() * (buy-1));
               call add_order(uuid(), 'INTERDAY', 'BUY', user_id_in, smbl, buy, price,
'COMPLETE');
      set price = cast((price + (RAND() * 10)) as decimal(7,2));
      call add_order(uuid(), 'INTERDAY', 'SELL', user_id_in, smbl, sell, price ,
'COMPLETE');
               set count = count+1;
   end while;
   set count = 0;
   set length = FLOOR(RAND() * 10);
   set buy = FLOOR(2 + RAND() * 5);
   set sell = FLOOR(1 + RAND() * (buy-1));
   while count < length do
               set smbl = (SELECT symbol FROM stock ORDER BY RAND() LIMIT 1);
      set price = (SELECT latestPrice FROM stock s where s.symbol = smbl);
      set buy = FLOOR(1 + RAND() * 5);
      set sell = FLOOR(1 + RAND() * (buy-1));
```

```
                call add_order(uuid(), 'INTRADAY', 'BUY', user_id_in, smbl, buy, price,
'COMPLETE');
        set price = cast((price - (RAND() * 10)) as decimal(7,2));
        call add_order(uuid(), 'INTRADAY', 'SELL', user_id_in, smbl, sell, price,
'COMPLETE');
                set count = count+1;
    end while;
END
CREATE DEFINER=`root`@`localhost` PROCEDURE `mutiple_subscription`(IN
user_id_in VARCHAR(36),IN length SMALLINT(3))
BEGIN
          declare count int unsigned default 0;
    while count < length do
                set count=count+1;
        INSERT IGNORE INTO subscription(user_id, symbol)
VALUES(user_id_in,(SELECT symbol FROM stock ORDER BY RAND() LIMIT 1));
    end while;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `pay_invoice`(IN invoice_id_in
VARCHAR(36), IN payment_id_in VARCHAR(36), IN payment_method_in
VARCHAR(10),
IN amount_in DECIMAL(7,2))
BEGIN
          INSERT INTO
payment_history(invoice_id,payment_id,payment_method,amount,payment_date)
    VALUES(invoice_id_in,
payment_id_in,payment_method_in,amount_in,CURRENT_TIMESTAMP);
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `subscribe`(IN user_id_in
VARCHAR(36), IN symbol_in VARCHAR(36))
BEGIN
          INSERT INTO subscription (user_id, symbol)
    VALUES(user_id_in, symbol_in);
END
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `unsubscribe`(IN user_id_in
VARCHAR(36), IN symbol_in VARCHAR(36))
BEGIN
        UPDATE subscription
   SET delete_flag = TRUE
   WHERE user_id = user_id_in AND symbol = symbol_in;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `update_password`(IN
email_id_in VARCHAR(50), IN password_in VARCHAR(500))
BEGIN
        UPDATE user
        SET password = password_in
        WHERE email_id = email_id_in;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `verify_access`(IN user_id_in
VARCHAR(36), IN api_in VARCHAR(100))
BEGIN
        SELECT * FROM user_role u
   JOIN access a ON u.role_id = a.role_id
   JOIN feature f ON a.feature_id = f.feature_id AND api_in RLIKE f.api
   WHERE u.user_id = user_id_in;
END
```

Function

```
CREATE DEFINER=`root`@`localhost` FUNCTION `RandString`(length SMALLINT(3),
digit boolean) RETURNS varchar(100) CHARSET utf8
   NO SQL
BEGIN
   SET @returnStr = '';
   SET @allowedChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
   IF digit THEN SET @allowedChars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
   END IF;
   SET @i = 0;
```

```
    WHILE (@i < length) DO
        SET @returnStr = CONCAT(@returnStr, substring(@allowedChars,
FLOOR(RAND() * LENGTH(@allowedChars) + 1), 1));
        SET @i = @i + 1;
    END WHILE;

    RETURN @returnStr;
END
```

2. OLTP DML code

We have created procedures and function to insert data into OLTP

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `loop`()
BEGIN
        declare max int unsigned default 10 + FLOOR(RAND() * 10);
        declare count int unsigned default 0;
    declare empcount int unsigned default 0;
    declare supervisor int unsigned default 0;
    declare RELMANAGER int unsigned default 0;
    declare adminid varchar(36) default uuid();
    declare EMPID varchar(36) default uuid();
    declare userid varchar(36) default uuid();
    declare invoiceid varchar(36) default uuid();
    declare monthn int unsigned default 12;
    declare num int unsigned default 0;
    declare brokrage_id  varchar(36) default uuid();
        DECLARE exit handler for sqlexception
                BEGIN
                        ROLLBACK;
                        RESIGNAL;
                END;
        START TRANSACTION;
    CALL add_new_user(adminid,'A',RandString(5, false),RandString(6,
false),'2002-12-04','M',CONCAT(RandString(10, true),'@mailinator.com'),

'4d63f211c7a9a9497d6504fa9599346626a1b06731fa7463f71b8a68804328c2',brokrage
_id, 'Test Corp', 'SILVER',
```

```
   0,'ADMIN',0.0,0.0,0,uuid(),'133 BAY RIDGE AVE APT
1','BROOKLYN','NY',11220,uuid(),'133 BAY RIDGE AVE APT 1','BROOKLYN','NY'
   ,11220,'1',5648756);
   CALL mutiple_subscription(adminid,15);
        while num< monthn do
      set invoiceid = uuid();
                INSERT INTO
invoice(invoice_id,brokerage_id,invoice_date,payment_due_date,invoice_amount)
                SELECT invoiceid,b.brokerage_id, DATE_SUB(CURRENT_TIMESTAMP,
INTERVAL num MONTH), LAST_DAY(DATE_SUB(CURRENT_TIMESTAMP,
INTERVAL num MONTH)), m.service_charge
                FROM brokerage b
                JOIN membership m ON b.membership_id = m.membership_id
                WHERE m.membership_type <> 'BASIC' AND b.brokerage_id =
brokrage_id;
      IF num > 0 THEN
                        INSERT INTO
payment_history(invoice_id,payment_id,payment_method,amount,payment_date)
                        VALUES(invoiceid, uuid(),'BANK',(SELECT invoice_amount from
invoice where invoice_id =
invoiceid),LAST_DAY(DATE_SUB(CURRENT_TIMESTAMP, INTERVAL num
MONTH)));
      END IF;
      set num = num+1;
   end while;
   set num = 0;
   select employee_id into supervisor from employee where user_id = adminid;
   while empcount < max do
                set empcount=empcount+1;
                set count =0;
      SET EMPID = UUID();
      CALL add_new_user(EMPID,'E',RandString(5, false),RandString(6,
false),'2002-12-04','M',CONCAT(RandString(10, true),'@mailinator.com'),

'4d63f211c7a9a9497d6504fa9599346626a1b06731fa7463f71b8a68804328c2',brokrage
_id,null,null,
      0,'MANAGER',0.0,0.0,supervisor,uuid(),'133 BAY RIDGE AVE APT
1','BROOKLYN','NY',11220,uuid(),'133 BAY RIDGE AVE APT 1','BROOKLYN','NY'
      ,11220,NULL,0);
      CALL mutiple_subscription(EMPID,15);
```

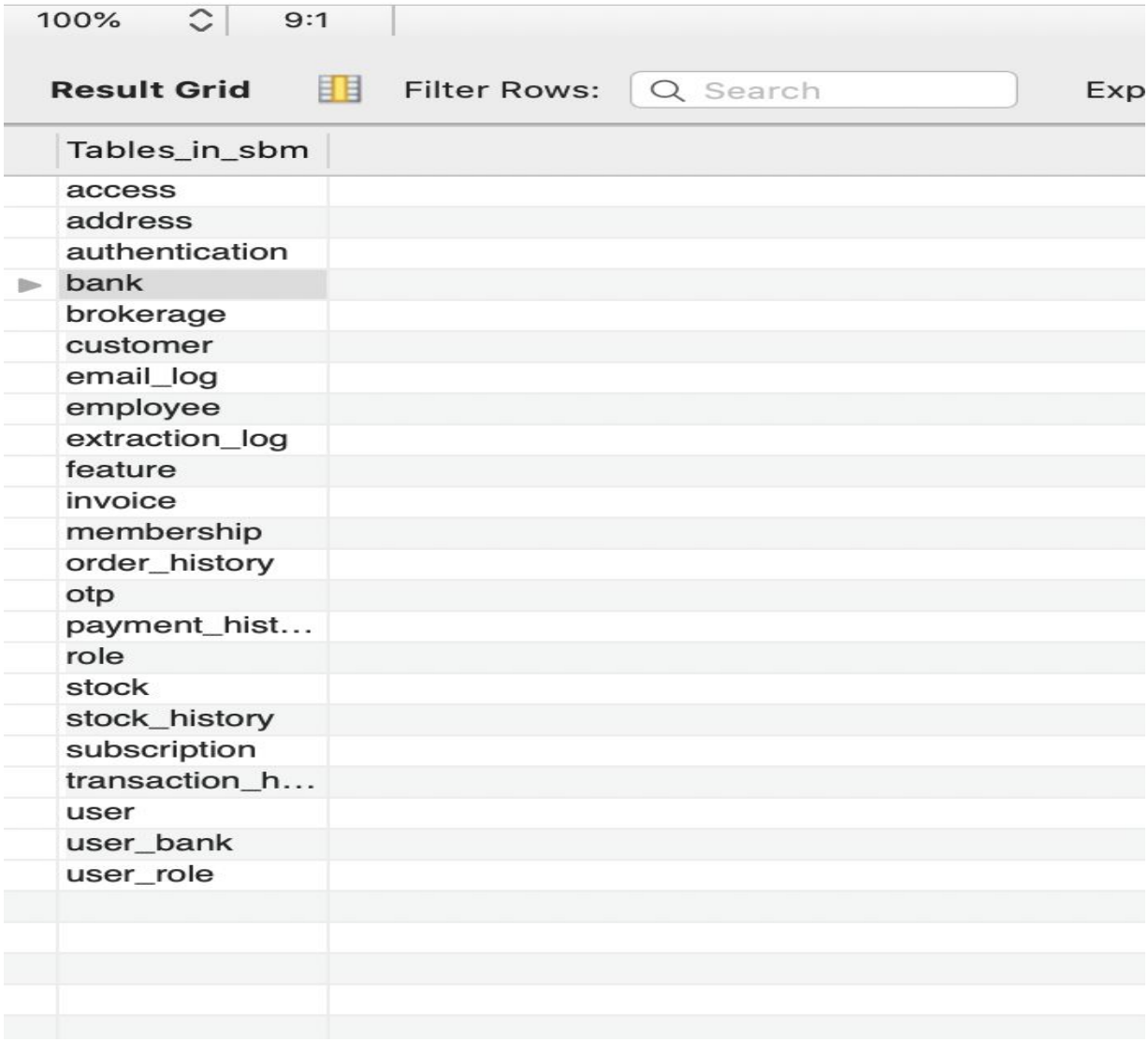
```
      select employee_id into RELMANAGER from employee where user_id = EMPID;
                while count < max do
                        set count=count+1;
         set userid = uuid();
                        CALL add_new_user(userid,'C',RandString(5, false),RandString(6,
false),'2002-12-04','M',CONCAT(RandString(10, true),'@mailinator.com'),

'4d63f211c7a9a9497d6504fa9599346626a1b06731fa7463f71b8a68804328c2',brokrage
_id,null,null,
                        RELMANAGER,null,0.0,0.0,0,uuid(),'133 BAY RIDGE AVE APT
1','BROOKLYN','NY',11220,uuid(),'133 BAY RIDGE AVE APT 1','BROOKLYN','NY'
                        ,11220,'1',5648756);
         CALL mutiple_subscription(userid,15);
         CALL add_transaction(uuid(), userid,'CREDIT',20000 + FLOOR(RAND() *
20000));
         CALL multiple_orders(userid);
        end while;
   end while;
   COMMIT;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `multiple_orders`(IN user_id_in
VARCHAR(36))
BEGIN
        declare length int unsigned default FLOOR(5 + RAND() * 10);
   declare buy int unsigned default FLOOR(1 + RAND() * 5);
   declare sell int unsigned default FLOOR(1 + RAND() * (buy-1));
   declare count int unsigned default 0;
   declare smbl varchar(36) default 'AAPL';
   declare price DECIMAL(7,2) default 121.55;
   while count < length do
                set smbl = (SELECT symbol FROM stock ORDER BY RAND() LIMIT 1);
      set price = (SELECT latestPrice FROM stock s where s.symbol = smbl);
      set buy = FLOOR(1 + RAND() * 5);
      set sell = FLOOR(1 + RAND() * (buy-1));
                call add_order(uuid(), 'INTERDAY', 'BUY', user_id_in, smbl, buy, price,
'COMPLETE');
      set price = cast((price + (RAND() * 10)) as decimal(7,2));
```

```
        call add_order(uuid(), 'INTERDAY', 'SELL', user_id_in, smbl, sell, price ,
'COMPLETE');
                    set count = count+1;
    end while;
    set count = 0;
    set length = FLOOR(RAND() * 10);
    set buy = FLOOR(2 + RAND() * 5);
    set sell = FLOOR(1 + RAND() * (buy-1));
    while count < length do
                    set smbl = (SELECT symbol FROM stock ORDER BY RAND() LIMIT 1);
        set price = (SELECT latestPrice FROM stock s where s.symbol = smbl);
        set buy = FLOOR(1 + RAND() * 5);
        set sell = FLOOR(1 + RAND() * (buy-1));
                    call add_order(uuid(), 'INTRADAY', 'BUY', user_id_in, smbl, buy, price,
'COMPLETE');
        set price = cast((price - (RAND() * 10)) as decimal(7,2));
        call add_order(uuid(), 'INTRADAY', 'SELL', user_id_in, smbl, sell, price,
'COMPLETE');
                    set count = count+1;
    end while;
END


CREATE DEFINER=`root`@`localhost` PROCEDURE `mutiple_subscription`(IN
user_id_in VARCHAR(36),IN length SMALLINT(3))
BEGIN
          declare count int unsigned default 0;
    while count < length do
                    set count=count+1;
        INSERT IGNORE INTO subscription(user_id, symbol)
VALUES(user_id_in,(SELECT symbol FROM stock ORDER BY RAND() LIMIT 1));
    end while;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `pay_invoice`(IN invoice_id_in
VARCHAR(36), IN payment_id_in VARCHAR(36), IN payment_method_in
VARCHAR(10),
IN amount_in DECIMAL(7,2))
BEGIN
```

```
        INSERT INTO
payment_history(invoice_id,payment_id,payment_method,amount,payment_date)
   VALUES(invoice_id_in,
payment_id_in,payment_method_in,amount_in,CURRENT_TIMESTAMP);
END

CREATE DEFINER=`root`@`localhost` FUNCTION `RandString`(length SMALLINT(3),
digit boolean) RETURNS varchar(100) CHARSET utf8
   NO SQL
BEGIN
   SET @returnStr = '';
   SET @allowedChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
   IF digit THEN SET @allowedChars =
'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
   END IF;
   SET @i = 0;

   WHILE (@i < length) DO
      SET @returnStr = CONCAT(@returnStr, substring(@allowedChars,
FLOOR(RAND() * LENGTH(@allowedChars) + 1), 1));
      SET @i = @i + 1;
   END WHILE;

   RETURN @returnStr;
END
```

3. OLTP Data Dictionary query and results (List of tables, constraint, columns and comments of each table)

```
1 •  Use sbm;
2 •  show tables;
```



```
4 •  SELECT TABLE_NAME, COLUMN_NAME, COLUMN_TYPE, IS_NULLABLE FROM   INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA = 'sbm';
5
```

100% 13:2

Result Grid Filter Rows: Search Export:

TABLE_NAME	COLUMN_NAME	COLUMN_TYPE	IS_NULLABLE
access	created_date	datetime	YES
access	feature_id	varchar(36)	NO
access	role_id	varchar(36)	NO
access	updated_date	datetime	YES
address	address_id	varchar(36)	NO
address	address_type	varchar(10)	NO
address	city	varchar(30)	NO
address	created_date	datetime	NO
address	state	char(2)	NO
address	street_address	varchar(60)	NO
address	updated_date	datetime	NO
address	user_id	varchar(36)	NO
address	zipcode	int	NO
authentication	authentication_id	varchar(36)	NO
authentication	created_date	datetime	NO
authentication	updated_date	datetime	NO
authentication	user_id	varchar(36)	NO
bank	bank_id	varchar(36)	NO
bank	bank_name	varchar(30)	YES
bank	created_date	datetime	NO
bank	updated_date	datetime	NO
brokerage	brokerage_id	varchar(36)	NO
brokerage	brokerage_name	varchar(30)	NO
brokerage	city	varchar(30)	NO
brokerage	created_date	datetime	NO
brokerage	membership_id	varchar(36)	NO
brokerage	state	char(2)	NO
brokerage	street_address	varchar(60)	NO
brokerage	updated_date	datetime	NO
brokerage	zipcode	int	NO
customer	created_date	datetime	YES
customer	customer_id	int	NO
customer	relationship_ma…	varchar(36)	YES
customer	updated_date	datetime	YES
customer	user_id	varchar(36)	NO
email_log	body	text	NO

TABLE_NAME	COLUMN_NAME	COLUMN_TYPE	IS_NULLABLE
email_log	body	text	NO
email_log	created_date	datetime	NO
email_log	log_id	varchar(36)	NO
email_log	subject	varchar(100)	NO
email_log	to_email	varchar(50)	NO
email_log	updated_date	datetime	NO
email_log	user_id	varchar(36)	NO
employee	commision	decimal(4,2)	YES
employee	created_date	datetime	NO
employee	designation	varchar(60)	NO
employee	employee_id	int	NO
employee	salary	decimal(9,2)	YES
employee	superviser	varchar(36)	YES
employee	updated_date	datetime	NO
employee	user_id	varchar(36)	NO
extraction_log	extraction_proc…	varchar(30)	NO
extraction_log	extraction_time…	datetime	NO
feature	api	varchar(100)	YES
feature	created_date	datetime	NO
feature	feature_id	varchar(36)	NO
feature	feature_name	varchar(30)	NO
feature	updated_date	datetime	NO
invoice	brokerage_id	varchar(36)	NO
invoice	created_date	datetime	NO
invoice	invoice_amount	decimal(7,2)	NO
invoice	invoice_date	datetime	NO
invoice	invoice_id	varchar(36)	NO
invoice	payment_due_d…	datetime	NO
invoice	updated_date	datetime	NO
membership	created_date	datetime	NO
membership	description	varchar(500)	YES
membership	membership_id	varchar(36)	NO
membership	membership_type	varchar(30)	NO
membership	service_charge	decimal(9,2)	NO
membership	trading_fee	decimal(5,2)	NO
membership	updated_date	datetime	NO
order_history	created_date	datetime	NO
order_history	order_end_time	datetime	YES
order_history	order_id	varchar(36)	NO
order_history	order_start_time	datetime	NO
order_history	order_status	varchar(36)	NO
order_history	order_type	varchar(10)	NO
order_history	quantity	int	NO
order_history	symbol	varchar(36)	NO
order_history	trade_type	varchar(10)	NO
order_history	unit_price	decimal(7,2)	NO

TABLE_NAME	COLUMN_NAME	COLUMN_TYPE	IS_NULLABLE
order_history	unit_price	decimal(7,2)	NO
order_history	updated_date	datetime	NO
order_history	user_id	varchar(36)	NO
otp	created_date	datetime	NO
otp	email_id	varchar(36)	NO
otp	otp	int	NO
otp	updated_date	datetime	NO
payment_hist…	amount	decimal(7,2)	NO
payment_hist…	created_date	datetime	NO
payment_hist…	invoice_id	varchar(36)	NO
payment_hist…	payment_date	datetime	NO
payment_hist…	payment_id	varchar(36)	NO
payment_hist…	payment_method	varchar(10)	NO
payment_hist…	updated_date	datetime	NO
role	created_date	datetime	NO
role	description	varchar(500)	YES
role	role_id	varchar(36)	NO
role	role_name	varchar(30)	NO
role	updated_date	datetime	NO
stock	close	decimal(7,2)	YES
stock	companyName	varchar(100)	NO
stock	created_date	datetime	NO
stock	description	text	YES
stock	high	decimal(7,2)	YES
stock	highTime	bigint	YES
stock	latestPrice	decimal(7,2)	NO
stock	latestUpdate	bigint	YES
stock	latestVolume	bigint	YES
stock	low	decimal(7,2)	YES
stock	lowTime	bigint	YES
stock	open	decimal(7,2)	YES
stock	symbol	varchar(36)	NO
stock	updated_date	datetime	NO
stock	volume	bigint	YES
stock	week52High	decimal(7,2)	YES
stock	week52Low	decimal(7,2)	YES
stock_history	close	decimal(7,2)	YES
stock_history	companyName	varchar(100)	NO
stock_history	created_date	datetime	NO
stock_history	description	text	YES
stock_history	high	decimal(7,2)	YES
stock_history	highTime	bigint	YES
stock_history	latestPrice	decimal(7,2)	NO
stock_history	latestUpdate	bigint	YES
stock_history	latestVolume	bigint	YES
stock_history	low	decimal(7,2)	YES

TABLE_NAME	COLUMN_NAME	COLUMN_TYPE	IS_NULLABLE
stock_history	created_date	datetime	NO
stock_history	description	text	YES
stock_history	high	decimal(7,2)	YES
stock_history	highTime	bigint	YES
stock_history	latestPrice	decimal(7,2)	NO
stock_history	latestUpdate	bigint	YES
stock_history	latestVolume	bigint	YES
stock_history	low	decimal(7,2)	YES
stock_history	lowTime	bigint	YES
stock_history	open	decimal(7,2)	YES
stock_history	symbol	varchar(36)	NO
stock_history	updated_date	datetime	NO
stock_history	volume	bigint	YES
stock_history	week52High	decimal(7,2)	YES
stock_history	week52Low	decimal(7,2)	YES
subscription	created_date	datetime	NO
subscription	delete_flag	tinyint	NO
subscription	symbol	varchar(36)	NO
subscription	updated_date	datetime	NO
subscription	user_id	varchar(36)	NO
transaction_history	amount	decimal(7,2)	NO
transaction_history	created_date	datetime	NO
transaction_history	transaction_id	varchar(36)	NO
transaction_history	transaction_type	varchar(10)	NO
transaction_history	updated_date	datetime	NO
transaction_history	user_id	varchar(36)	NO
user	brokerage_id	varchar(36)	NO
user	created_date	datetime	NO
user	dob	date	NO
user	email_id	varchar(50)	NO
user	first_name	varchar(30)	NO
user	gender	char(1)	NO
user	last_name	varchar(30)	NO
user	password	varchar(500)	NO
user	updated_date	datetime	NO
user	user_id	varchar(36)	NO
user	user_type	char(1)	NO
user_bank	bank_account_…	decimal(20,0)	NO
user_bank	bank_id	varchar(36)	NO
user_bank	created_date	datetime	NO
user_bank	updated_date	datetime	NO
user_bank	user_id	varchar(36)	NO
user_role	created_date	datetime	NO
user_role	role_id	varchar(36)	NO
user_role	updated_date	datetime	NO
user_role	user_id	varchar(36)	NO

4. **DW DDL code (Tables, Constraints, Trigger, Partitioned tables, any function/ procedure created etc.)**

```
CREATE TABLE "Access" (
    feature_id     VARCHAR2(36) NOT NULL,
    feature_name   VARCHAR2(30) NOT NULL,
    api            VARCHAR2(100),
    role_id        VARCHAR2(36),
    role_name      VARCHAR2(30),
    description    VARCHAR2(500),
    created_date   DATE NOT NULL,
    updated_date   DATE NOT NULL,
    user_id        VARCHAR2(36 CHAR)
);

ALTER TABLE "Access" ADD CONSTRAINT access_pk PRIMARY KEY ( feature_id );

CREATE TABLE bank_account (
    bank_id              VARCHAR2(36) NOT NULL,
    bank_name            VARCHAR2(30) NOT NULL,
    bank_account_number   NUMBER(20) NOT NULL,
    crated_date          DATE NOT NULL,
    updated_date         DATE NOT NULL,
    user_id              VARCHAR2(36 CHAR)
);

ALTER TABLE bank_account ADD CONSTRAINT bank_account_pk PRIMARY KEY (
bank_id );

CREATE TABLE brokerage (
    brokerage_id      VARCHAR2(36) NOT NULL,
    brokerage_name    VARCHAR2(30) NOT NULL,
    street_address    VARCHAR2(60 CHAR) NOT NULL,
    city              VARCHAR2(60) NOT NULL,
    state             CHAR(2) NOT NULL,
    zipcode           NUMBER(5) NOT NULL,
    created_date      DATE NOT NULL,
    updated_date      DATE NOT NULL,
    membership_id     VARCHAR2(36) NOT NULL,
```

```
    membership_name   VARCHAR2(30) NOT NULL,
    description       VARCHAR2(500) NOT NULL,
    service_charge    NUMBER(9, 2) NOT NULL,
    trading_fee       NUMBER(5, 2) NOT NULL
);

ALTER TABLE brokerage ADD CONSTRAINT brokerage_pk PRIMARY KEY (
brokerage_id,
                                          membership_id );

CREATE TABLE "ORDER" (
    order_id       VARCHAR2(36 CHAR) NOT NULL,
    order_type     VARCHAR2(10) NOT NULL,
    trade_type     VARCHAR2(10) NOT NULL,
    quantity       NUMBER(5) NOT NULL,
    unit_price     NUMBER(7, 2) NOT NULL,
    created_date   DATE NOT NULL,
    updated_date   DATE NOT NULL,
    symbol         VARCHAR2(36) NOT NULL,
    company_name   VARCHAR2(100) NOT NULL,
    description    VARCHAR2(100) NOT NULL,
    user_id        VARCHAR2(36 CHAR)
);

ALTER TABLE "ORDER" ADD CONSTRAINT order_pk PRIMARY KEY ( order_id );

CREATE TABLE payment (
    invoice_id             VARCHAR2(36) NOT NULL,
    inoice_date            DATE NOT NULL,
    payment_due_date       DATE NOT NULL,
    invoice_amount         NUMBER(7, 2) NOT NULL,
    created_date           DATE NOT NULL,
    updated_date           DATE NOT NULL,
    payment_created_date   DATE NOT NULL,
    payment_update_date    DATE NOT NULL,
    payment_id             VARCHAR2(36),
    payment_date           DATE,
    payment_method         VARCHAR2(10),
    payment_amount         NUMBER(7, 2),
    brokerage_id           VARCHAR2(36),
```

```
    membership_id          VARCHAR2(36)
);

ALTER TABLE payment ADD CONSTRAINT payment_pk PRIMARY KEY ( invoice_id
);

CREATE TABLE subscription (
    symbol         VARCHAR2(36) NOT NULL,
    company_name   VARCHAR2(100) NOT NULL,
    description    VARCHAR2(500),
    created_date   DATE NOT NULL,
    updated_date   DATE NOT NULL,
    open           NUMBER(7, 2),
    close          NUMBER(7, 2),
    high           NUMBER(7, 2),
    hightime       NUMBER,
    low            NUMBER(7, 2),
    lowtime        NUMBER,
    latestprice    NUMBER(7, 2),
    latestupdate   NUMBER,
    latestvolume   NUMBER,
    volume         NUMBER,
    week52high     NUMBER(7, 2),
    week52low      NUMBER(7, 2),
    user_id        VARCHAR2(36 CHAR)
);

ALTER TABLE subscription ADD CONSTRAINT subscription_pk PRIMARY KEY (
symbol );

CREATE TABLE transaction (
    transaction_id     VARCHAR2(36) NOT NULL,
    transaction_type   VARCHAR2(10) NOT NULL,
    amount             NUMBER(7, 2) NOT NULL,
    user_id            VARCHAR2(36 CHAR)
);

ALTER TABLE transaction ADD CONSTRAINT transaction_pk PRIMARY KEY (
transaction_id );
```

```
CREATE TABLE "USER" (
    id                  NUMBER NOT NULL,
    user_id             VARCHAR2(36 CHAR) NOT NULL,
    first_name          VARCHAR2(30) NOT NULL,
    last_name           VARCHAR2(30) NOT NULL,
    dob                 DATE NOT NULL,
    gender              CHAR(1) NOT NULL,
    email_id            VARCHAR2(50) NOT NULL,
    password            VARCHAR2(500) NOT NULL,
    created_date        DATE NOT NULL,
    updated_date        DATE NOT NULL,
    bank_id             VARCHAR2(36 CHAR),
    bank_account_number    NUMBER(20),
    bank_name           VARCHAR2(30),
    address_id          VARCHAR2(36) NOT NULL,
    address_type        VARCHAR2(10 CHAR) NOT NULL,
    street_address      VARCHAR2(60 CHAR) NOT NULL,
    city                VARCHAR2(30) NOT NULL,
    state               CHAR(2) NOT NULL,
    zipcode             NUMBER(5) NOT NULL,
    designation         VARCHAR2(60 CHAR),
    salary              NUMBER(9, 2),
    commision           NUMBER(4, 2),
    superviser          VARCHAR2(36),
    relationship_manager   VARCHAR2(36),
    brokerage_id        VARCHAR2(36) NOT NULL,
    membership_id       VARCHAR2(36) NOT NULL
);

ALTER TABLE "USER" ADD CONSTRAINT user_pk PRIMARY KEY ( id );

ALTER TABLE "USER" ADD CONSTRAINT user_user_id_un UNIQUE ( user_id );

ALTER TABLE "Access"
    ADD CONSTRAINT access_user_fk FOREIGN KEY ( user_id )
        REFERENCES "USER" ( user_id );

ALTER TABLE bank_account
    ADD CONSTRAINT bank_account_user_fk FOREIGN KEY ( user_id )
        REFERENCES "USER" ( user_id );
```

```
ALTER TABLE "ORDER"
   ADD CONSTRAINT order_user_fk FOREIGN KEY ( user_id )
      REFERENCES "USER" ( user_id );

ALTER TABLE payment
   ADD CONSTRAINT payment_brokerage_fk FOREIGN KEY ( brokerage_id,
                                        membership_id )
      REFERENCES brokerage ( brokerage_id,
                             membership_id );

ALTER TABLE subscription
   ADD CONSTRAINT subscription_user_fk FOREIGN KEY ( user_id )
      REFERENCES "USER" ( user_id );

ALTER TABLE transaction
   ADD CONSTRAINT transaction_user_fk FOREIGN KEY ( user_id )
      REFERENCES "USER" ( user_id );

ALTER TABLE "USER"
   ADD CONSTRAINT user_brokerage_fk FOREIGN KEY ( brokerage_id,
                                       membership_id )
      REFERENCES brokerage ( brokerage_id,
                             membership_id );

CREATE SEQUENCE user_id_seq START WITH 1 NOCACHE ORDER;

CREATE OR REPLACE TRIGGER user_id_trg BEFORE
   INSERT ON "USER"
   FOR EACH ROW
   WHEN ( new.id IS NULL )
BEGIN
   :new.id := user_id_seq.nextval;
END;
/
```

External Table

```
BEGIN
```

```
   DBMS_CLOUD.CREATE_EXTERNAL_TABLE(
    table_name =>'subscription_ext',
    credential_name =>'CRED_NAME',
    file_uri_list
=>'https://objectstorage.us-ashburn-1.oraclecloud.com/n/idzv0nf0nloe/b/SBM-External-
Table-Data/o/dw_subscription_stage.csv',
    format => json_object('type' value 'csv','delimiter' value ','),
    column_list => 'symbol         VARCHAR2(36) NOT NULL, company_name
VARCHAR2(100) NOT NULL, description VARCHAR2(500),
        open           NUMBER(7, 2),
        close          NUMBER(7, 2),
        high           NUMBER(7, 2),
        hightime       NUMBER(20),
        low            NUMBER(7, 2),
        lowtime        NUMBER(20),
        latestprice    NUMBER(7, 2),
        latestupdate   NUMBER(20),
        latestvolume   NUMBER(20),
        volume         NUMBER(20),
        week52high     NUMBER(7, 2),
        week52low      NUMBER(7, 2),
        user_id        VARCHAR2(36 CHAR),
        delete_flag    NUMBER(1)' );
END;
```

5. **ETL code used** (at least one example of extract, transform, load)

Create stage table

```
CREATE TABLE DW_SUBSCRIPTION_STAGE (
    symbol         VARCHAR2(36) NOT NULL,
    company_name   VARCHAR2(100) NOT NULL,
    description    VARCHAR2(500),
    created_date   DATE NOT NULL,
```

```
    updated_date   DATE NOT NULL,
    open           NUMBER(7, 2),
    close          NUMBER(7, 2),
    high           NUMBER(7, 2),
    hightime       NUMBER(20),
    low            NUMBER(7, 2),
    lowtime        NUMBER(20),
    latestprice    NUMBER(7, 2),
    latestupdate   NUMBER(20),
    latestvolume   NUMBER(20),
    volume         NUMBER(20),
    week52high     NUMBER(7, 2),
    week52low      NUMBER(7, 2),
    user_id        VARCHAR2(36 CHAR),
    delete_flag    NUMBER(1)
);


ALTER TABLE DW_SUBSCRIPTION_STAGE ADD CONSTRAINT subscription_pk
PRIMARY KEY ( symbol,user_id );
```

Trigger of stage table

```
CREATE OR REPLACE TRIGGER SUBSCRIPTION_STAGE_BEFORE_INSERT
BEFORE INSERT ON DW_SUBSCRIPTION_STAGE FOR EACH ROW BEGIN
        :new.created_date := sysdate;
        :new.updated_date := sysdate;
END;
CREATE OR REPLACE TRIGGER SUBSCRIPTION_STAGE_BEFORE_UPDATE
BEFORE UPDATE ON DW_SUBSCRIPTION_STAGE FOR EACH ROW BEGIN
    :new.updated_date := sysdate;
END;
```

Create DW table

```
CREATE TABLE DW_SUBSCRIPTION AS SELECT * FROM
DW_SUBSCRIPTION_STAGE WHERE 1=2;

ALTER TABLE DW_SUBSCRIPTION ADD CONSTRAINT subscription_dw_pk
PRIMARY KEY ( symbol,user_id );
```

Extraction

For User

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `extract_user`()
BEGIN
	declare last_time DATETIME default '2020-01-01';
   select MAX(extraction_timestamp) INTO last_time from extraction_log where
extraction_procedure = 'extract_user';
	select u.user_id, u.first_name, u.last_name, u.dob, u.gender, u.email_id,
u.password, COALESCE(b.bank_id,0), b.bank_name,
	 COALESCE(ub.bank_account_number,0), a.address_id, a.address_type,
a.street_address, a.city, a.state, a.zipcode, e.designation,
	 COALESCE(e.salary,0), COALESCE(e.commision,0), e.superviser,
c.relationship_manager, bk.brokerage_id, bk.membership_id
	 INTO OUTFILE '/var/lib/mysql-files/dw_user_stage.csv'
	FIELDS TERMINATED BY ','  OPTIONALLY ENCLOSED BY '"' ESCAPED BY ''
	LINES TERMINATED BY '\n'
	from user u
   LEFT JOIN user_bank ub ON u.user_id = ub.user_id
   LEFT JOIN bank b on b.bank_id = ub.bank_id
	LEFT join address a ON a.user_id = u.user_id
   LEFT JOIN employee e ON e.user_id = u.user_id
   LEFT JOIN customer c ON c.user_id = u.user_id
   LEFT JOIN brokerage bk ON u.brokerage_id = bk.brokerage_id
	WHERE u.updated_date> last_time or ub.updated_date> last_time
	or a.updated_date> last_time or e.updated_date> last_time
	or c.updated_date> last_time or bk.updated_date> last_time;
   INSERT into extraction_log(extraction_procedure,extraction_timestamp)
VALUES('extract_user',CURRENT_TIMESTAMP);
END
```

For Subscription

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `extract_subscription`()
BEGIN

	declare lastdate datetime default '2020-01-01';
```

```
        select MAX(extraction_timestamp) into lastdate from extraction_log where
extraction_procedure = 'extract_subscription';
        SELECT s.symbol, st.companyName, st.description, st.open, st.close, st.high,
st.highTime, st.low, st.lowTime,
   st.latestPrice, st.latestUpdate, st.latestVolume, st.volume, st.week52High,
st.week52Low, s.user_id, s.delete_flag
   INTO OUTFILE '/var/lib/mysql-files/dw_subscription_stage.csv'
        FIELDS TERMINATED BY ','  OPTIONALLY ENCLOSED BY '"' ESCAPED BY ''
        LINES TERMINATED BY '\n'
        FROM stock st JOIN subscription s ON st.symbol=s.symbol
        WHERE st.updated_date> lastdate or s.updated_date> lastdate;
        INSERT into extraction_log(extraction_procedure,extraction_timestamp)
VALUES('extract_subscription',CURRENT_TIMESTAMP);
END
```

For Order

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `extract_order`()
BEGIN
        declare last_time DATETIME default '2020-01-01';
   select MAX(extraction_timestamp) INTO last_time from extraction_log where
extraction_procedure = 'extract_order';
        select o.order_id, o.order_type, o.trade_type, o.quantity, o.unit_price, o.symbol,
        s.companyName, s.description, o.user_id
        INTO OUTFILE '/var/lib/mysql-files/dw_order_stage.csv'
        FIELDS TERMINATED BY ','  OPTIONALLY ENCLOSED BY '"' ESCAPED BY ''
        LINES TERMINATED BY '\n'
        FROM order_history o JOIN stock s ON o.symbol = s.symbol
        WHERE o.updated_date> last_time or s.updated_date> last_time;
   INSERT into extraction_log(extraction_procedure,extraction_timestamp)
VALUES('extract_order',CURRENT_TIMESTAMP);
END
```

Transform

Transformation of data is done by oracle autonomous database on load of extracted csv files.

Data load into Data warehouse

First Time

Procedure for data load into user table

```
CREATE OR REPLACE EDITIONABLE PROCEDURE "ADMIN"."LOAD_DW_USER"
IS
dcnt NUMBER;
icnt NUMBER;
err_code NUMBER;
err_msg VARCHAR2(32000);
BEGIN
dcnt := 0;
icnt := 0;
for s in (select * from DW_USER_STAGE)
loop
   delete from DW_USER where user_id = s.user_id and address_id = s.address_id
and bank_id = s.bank_id;
   dcnt := dcnt+1;
     INSERT INTO DW_USER(id, user_id, first_name, last_name, dob, gender,
email_id, password, bank_id, bank_name,
 bank_account_number, address_id, address_type, street_address, city, state, zipcode,
designation,
 salary, commision, superviser, relationship_manager, brokerage_id, membership_id,
created_date, updated_date )
     VALUES (s.id, s.user_id, s.first_name, s.last_name, s.dob, s.gender, s.email_id,
s.password, s.bank_id, s.bank_name,
 s.bank_account_number, s.address_id, s.address_type, s.street_address, s.city,
s.state, s.zipcode, s.designation,
 s.salary, s.commision, s.superviser, s.relationship_manager, s.brokerage_id,
s.membership_id, SYSDATE,SYSDATE);
    icnt := icnt+1;
    commit;
    dbms_output.put_line('Total Deleted Record: '||dcnt||' Total Inserted Records: '||icnt);
end loop;
    commit;
    dbms_output.put_line('Total Deleted Record: '||dcnt||' Total Inserted Records: '||icnt);
EXCEPTION
   WHEN OTHERS THEN
       err_code := SQLCODE;
```

```
        err_msg  := SQLERRM;
        dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);
END;
/
```

Procedure for data load into order table

```
CREATE OR REPLACE EDITIONABLE PROCEDURE "ADMIN"."LOAD_DW_ORDER"
IS
dcnt NUMBER;
icnt NUMBER;
err_code NUMBER;
err_msg VARCHAR2(32000);
BEGIN
dcnt := 0;
icnt := 0;
for s in (select * from DW_ORDER_STAGE ORDER BY order_id OFFSET 0 ROWS
FETCH NEXT 10000 ROWS ONLY)
loop
   delete from dw_order where order_id=s.order_id;
   dcnt := dcnt+1;
    INSERT INTO DW_ORDER (order_id, order_type, trade_type, quantity, unit_price,
symbol, company_Name, description, user_id, created_date, updated_date )
          VALUES (s.order_id, s.order_type, s.trade_type, s.quantity, s.unit_price,
s.symbol, s.company_Name, s.description, s.user_id, SYSDATE,SYSDATE);
         icnt := icnt+1;
    commit;
    dbms_output.put_line('Total Deleted Record: '||dcnt||' Total Inserted Records: '||icnt);
end loop;
    commit;
    dbms_output.put_line('Total Deleted Record: '||dcnt||' Total Inserted Records: '||icnt);
EXCEPTION
   WHEN OTHERS THEN
        err_code := SQLCODE;
        err_msg  := SQLERRM;
        dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);
END;
/
```

Procedure for data load into subscription table

```
CREATE OR REPLACE EDITIONABLE PROCEDURE
"ADMIN"."LOAD_DW_SUBSCRIPTION" IS
dcnt NUMBER;
icnt NUMBER;
err_code NUMBER;
err_msg VARCHAR2(32000);
BEGIN
dcnt := 0;
icnt := 0;
for s in (select * from DW_SUBSCRIPTION_STAGE ORDER BY symbol,user_id
OFFSET 0 ROWS FETCH NEXT 10000 ROWS ONLY)
loop
   delete from dw_subscription where symbol=s.symbol and user_id = s.user_id;
   dcnt := dcnt+1;
    IF s.delete_flag < 1 THEN
       INSERT INTO DW_SUBSCRIPTION(symbol, company_name, description, open,
close, high, highTime, low, lowTime, latestPrice, latestUpdate, latestVolume, volume,
week52High, week52Low, user_id,  created_date, updated_date)
       VALUES (s.symbol, s.company_name, s.description, s.open, s.close, s.high,
s.highTime, s.low, s.lowTime, s.latestPrice, s.latestUpdate, s.latestVolume, s.volume,
s.week52High, s.week52Low, s.user_id, SYSDATE,SYSDATE);
       icnt := icnt+1;
       commit;
       dbms_output.put_line('Total Deleted Record: '||dcnt||' Total Inserted Records:
'||icnt);
   end if;
end loop;
    commit;
    dbms_output.put_line('Total Deleted Record: '||dcnt||' Total Inserted Records: '||icnt);
EXCEPTION
   WHEN OTHERS THEN
       err_code := SQLCODE;
       err_msg  := SQLERRM;
       dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);
END;
/
```

On subsequent loads after first load:

For user

```
CREATE OR REPLACE EDITIONABLE PROCEDURE
"ADMIN"."LOAD_MERGE_DW_USER" IS
err_code NUMBER;
err_msg VARCHAR2(32000);
BEGIN
   MERGE INTO DW_USER e
   USING DW_USER_STAGE s
   ON (e.user_id = s.user_id and e.address_id = s.address_id and e.bank_id =
s.bank_id)
  WHEN MATCHED THEN
   UPDATE SET
         e.id = s.id,
       e.user_id=s.user_id,
       e.first_name=s.first_name,
       e.last_name=s.last_name,
       e.dob=s.dob,
       e.gender=s.gender,
       e.email_id=s.email_id,
       e.password=s.password,
       e.bank_id=s.bank_id,
       e.bank_name=s.bank_name,
       e.bank_account_number=s.bank_account_number,
       e.address_id=s.address_id,
       e.address_type=s.address_type,
       e.street_address=s.street_address,
       e.city=s.city,
       e.state=s.state,
       e.zipcode=s.zipcode,
       e.designation=s.designation,
       e.salary=s.salary,
       e.commision=s.commision,
       e.superviser=s.superviser,
       e.relationship_manager=s.relationship_manager,
       e.brokerage_id=s.brokerage_id,
       e.membership_id=s.membership_id,
       e.updated_date=sysdate
  WHEN NOT MATCHED THEN
```

```
    INSERT (id, user_id, first_name, last_name, dob, gender, email_id, password,
bank_id, bank_name,
 bank_account_number, address_id, address_type, street_address, city, state, zipcode,
designation,
 salary, commision, superviser, relationship_manager, brokerage_id, membership_id,
created_date, updated_date )
    VALUES (s.id, s.user_id, s.first_name, s.last_name, s.dob, s.gender, s.email_id,
s.password, s.bank_id, s.bank_name,
 s.bank_account_number, s.address_id, s.address_type, s.street_address, s.city,
s.state, s.zipcode, s.designation,
 s.salary, s.commision, s.superviser, s.relationship_manager, s.brokerage_id,
s.membership_id, SYSDATE,SYSDATE);
    commit;
EXCEPTION
   WHEN OTHERS THEN
       err_code := SQLCODE;
       err_msg  := SQLERRM;
       dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);
END;
/
```

For Subscription

```
CREATE OR REPLACE EDITIONABLE PROCEDURE
"ADMIN"."LOAD_MERGE_DW_SUBSCRIPTION" IS
err_code NUMBER;
err_msg VARCHAR2(32000);
BEGIN
   MERGE INTO DW_SUBSCRIPTION e
   USING DW_SUBSCRIPTION_STAGE s
   ON (e.symbol= s.symbol and e.user_id = s.user_id)
  WHEN MATCHED THEN
   UPDATE SET
       e.symbol=s.symbol,
       e.company_name=s.company_name,
       e.description=s.description,
       e.open=s.open,
       e.close=s.close,
       e.high=s.high,
       e.highTime=s.highTime,
       e.low=s.low,
```

```
        e.lowTime=s.lowTime,
        e.latestPrice=s.latestPrice,
        e.latestUpdate=s.latestUpdate,
        e.volume=s.volume,
        e.week52High=s.week52High,
        e.week52Low=s.week52Low,
        e.user_id=s.user_id,
        e.updated_date=sysdate
  WHEN NOT MATCHED THEN
     INSERT (symbol, company_name, description, open, close, high, highTime, low,
lowTime, latestPrice, latestUpdate, latestVolume, volume, week52High, week52Low,
user_id,  created_date, updated_date )
     VALUES (s.symbol, s.company_name, s.description, s.open, s.close, s.high,
s.highTime, s.low, s.lowTime, s.latestPrice, s.latestUpdate, s.latestVolume, s.volume,
s.week52High, s.week52Low, s.user_id , SYSDATE,SYSDATE);
     commit;
EXCEPTION
   WHEN OTHERS THEN
        err_code := SQLCODE;
        err_msg  := SQLERRM;
        dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);
END;
```

For order

```
CREATE OR REPLACE PROCEDURE          ADMIN.load_merge_dw_order IS
err_code NUMBER;
err_msg VARCHAR2(32000);
BEGIN
   MERGE INTO DW_ORDER e
   USING DW_ORDER_STAGE s
   ON (e.order_id= s.order_id)
  WHEN MATCHED THEN
   UPDATE SET
        e.order_id=s.order_id,
        e.order_type=s.order_type,
        e.trade_type=s.trade_type,
        e.quantity=s.quantity,
```

```
        e.unit_price=s.unit_price,
        e.symbol=s.symbol,
        e.company_name=s.company_name,
        e.description=s.description,
        e.user_id=s.user_id,
        e.updated_date=sysdate
  WHEN NOT MATCHED THEN
     INSERT (order_id, order_type, trade_type, quantity, unit_price, symbol,
company_name, description, user_id, created_date, updated_date )
     VALUES (s.order_id, s.order_type, s.trade_type, s.quantity, s.unit_price, s.symbol,
s.company_name, s.description, s.user_id, SYSDATE,SYSDATE);
     commit;
EXCEPTION
   WHEN OTHERS THEN
        err_code := SQLCODE;
        err_msg  := SQLERRM;
        dbms_output.put_line('Error code ' || err_code || ': ' || err_msg);
END;
/
```