## General Housekeeping:

**Initializing the database:**

In order to get the SQL Database set up initially, run the team10.sql file. Note: At the end of this file it performs 2 inserts, 1.) to create an administrator 2.) to initialize the mutual date. If no administrator is initialized, then no one will be able to login to create users. We decided that we would not allow a user to create an administrator from the UI since we thought that would be “unsafe”.

**Running the Java Code:**

To run the java code, just add the postgresql-42.2.18.jar file to the same folder as the .java file. Then navigate to the folder containing the .java file and execute:

javac -cp "postgresql-42.2.18.jar;."team10.java && java -cp "postgresql-42.2.18.jar;." team10

(if you are on windows)

OR

javac -cp postgresql-42.2.18.jar team10.java && java -cp postgresql-42.2.18.jar:. team10

(if you are on Linux)

## Navigating the UI

**Logging In:**

1. When you first start the program, you will be asked to log in. In order to login as an administrator, type “admin”. To login as a user, type “customer”.
2. You will then be asked for a login and password. If the username and password do not match with a login and password in the database, you will be brought back to the initial login menu.

Once inside of each of the interfaces, we set up the ui so that each numbered task corresponds to the input to begin that task. This is also explained in the UI.

**Administrator Interface**

1. Erase the database
   1. Once this option is selected, you will be prompted as to whether or not you want to proceed. For yes, type “y” for no type “n”.
   2. Once the database is erased, it will ask you to create a new admin. This is so you won’t be locked out and unable to create any new users or populate the tables. It will also ask you to initialize the mutual date.
2. Add a customer
   1. Once this option is selected, you will be prompted for input about the new customer being added. Follow the prompts and input the requested information for the customer. For the balance you will be prompted for if you want to insert a balance. If you enter “y” to input a balance then follow the requests or if you enter “n” the balance will be initialized to 0.
3. Add new mutual fund
   1. Once this option is selected, you will be prompted for required information about the mutual fund. After providing the information the new mutual fund will be created.
4. Update share quotes for a day
   1. Once this option is selected, you will be prompted for the filename of the location for where the mutual fund symbols and prices are located. After the filename is entered the share quotes will be updated.
5. Show top-k highest volume categories
   1. Once this option is selected, you will be prompted to supply a number. This number is the number of categories to display from the top based on the shares owned by the customers.
6. Rank all the investors
   1. Once this option is selected, you will not be prompted for any further input. Instead you will be shown the customers ranked by the total value of shares they own.
7. Update the current date (i.e., the “pseudo” date
   1. Once this option is selected, you will be prompted to enter in the date. The format for the date is very specific so ensure to follow the format provided in the message prompt. It should be <YYYY-MM-DD>

**Customer Interface**

1. Show the customer’s balance and total number of shares
   1. Once this option is selected, the customers balance and their total number of shares will be returned.
2. Show mutual funds sorted by name
   1. The user will not be prompted after this is selected. The mutual funds will simply be sorted by name and output.
3. Show mutual funds sorted by prices on a date
   1. When this is selected, the user will then be prompted to input what date they would like to check the mutual funds for. This should be input in the format <YYYY-MM-DD>
4. Search for a mutual fund
   1. The customer will be asked to input two keywords. A list of mutual funds will be returned which shows the mutual funds that have those keywords in their description.
5. Deposit an amount for investment
   1. The user will be prompted to input how much they would like to deposit. If the user has allocation preferences defined, the deposited money will be divided into the respective mutual funds. Any leftover money will be added to their balance.
6. Buy shares
   1. Once you choose this option you will be prompted to decide whether you would like to buy shares by the number of shares or by the amount. Once that is decided, you will then be prompted to designate the share’s symbol and the number or amount that should be bought. If something goes wrong (i.e. there is not enough money in the account) then it will print that you do not have enough funds.
7. Sell shares
   1. Once you choose this option, you will be prompted to designate the share symbol and the number of shares that you would like to sell. If there are not enough shares, the user will be informed of the error.
8. Show ROI (return of investment)
   1. Once this option is chosen, the return on investment for each of the funds currently owned by the user will be shown along with the fund name.
9. Predict the gain or loss of the customer’s transactions
   1. There is no input for this option and once this option is chosen it will simply print out the loss/profit/hold for each of the users past transactions.
10. Change allocation preference
    1. First this option will ask the user to confirm that they want to change their allocation.
    2. Next it will ask them for the symbol of the mutual fund they would like to allocate followed by the percent they would like to allocate to that mutual fund.
    3. It will repeat that process until the user has allocated 100%.
11. Rank the customer’s allocations
    1. Once this option is chosen, the allocations for the user will be printed in the order of their ROI. For this, we assumed that by customer’s allocation, it was meant the allocation number instead of the individual mutual funds for the customer’s most recent allocation
12. Show portfolio
    1. The user will not be prompted for this, the portfolio will simply be output as was described in the prompt for the project.

## Assumptions/Improvements

To the best of our knowledge/testing, we implemented the various tasks as described in the prompt. We did make a couple of assumptions as to how the program should run:

1. We made the assumption for selling the shares that simply rebalancing the customers balance was not sufficient. So we also updated the owns and trxlog when a sale is made.
2. For task 11, we made the assumption that we should aggregate the ROI for each allocation instead of the ROI for the individual funds within the allocation.
3. Unless the user was specifically asked to supply a date (e.g. Sort funds by date) the p\_date was used.
4. The mutual\_date table only maintains the current date. This date is updated as the date changes instead of continually concatenating to the table.

For the concurrency portion of the project we believe we implemented proper concurrency controls on the buy and sell functions. That being said, we did not have a feasible way to test whether the concurrency controls actually work, so we are not 100% sure that they are sufficient.

**Improvements:**

One of the first things that we would improve if we had more time to work on this project would be improving the UI. While it works well enough and gets the job done, it can be clunky at times and could most likely be streamlined.

Another thing that we would improve would be to allow the customer to turn some of the triggers that we created on and off. Based on the project description, we assumed the triggers like “buy\_on\_date” and “price\_jump” were supposed to exist, but not be accessible by the users. So, we would include a way for the user to access/modify those triggers.

We would also likely refactor our code if we had more time. Again, to our knowledge and testing, it works well, but the code can be a bit obscure in some places and some of the functions/triggers can be challenging to find/edit. There were a couple of times where changing one piece of our code impacted other areas, which made developing our application challenging at times. This would probably be largely mitigated if we were to refactor the code.