CSCI 330
Assignment 2
Due: Friday, February 19

Introduction

The purpose of this assignment is to use SQL to access a database from a Java program.

Problem Statement

Using a database of stock price and dividend data (thanks to Prof. Johnson), similar to the data from Assignment 1, write a program that computes the gain or loss from the trading strategy described below under Processing.

Accessing the Database

Each student has three database accounts on the server `mysql.cs.wwu.edu`. The credentials, will be provided to you in a separate email. "DatabaseAccess.pdf", on Canvas, discusses how to setup and run Java programs accessing the database and provides additional information about your database accounts.

For this assignment you will need to access the database `reedy330` on the `mysql.cs.wwu.edu` server. You should access the database in the same way as the demo program discussed in class and available on Canvas. Also on canvas is a dummy connectparams.txt file that provides a sample of the connect parameters file that works with the demo program.

You should have read-only access to the `reedy330` database.

Your program **must** use a file named "`connectparams.txt`" as the default file for the connection parameters. The demo program shows one way to do this.

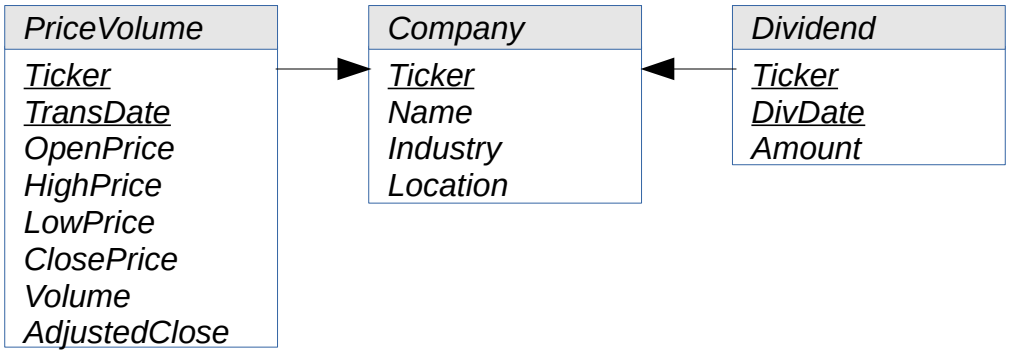
Comment: There are two good reasons for moving the connection parameters to an external file:

1. Security. If you hard code the connection parameters into your program, the password is vulnerable to some clever person decoding the executable. And, there are programs called dis-assemblers that do all the hard work of doing this for you.
2. Testing. You don't want to have to change a program to move it from a test environment to an operational environment. Moving the connection parameters to an external file enables this.

Also, see the hints below for problems with establishing a connection.

Database Schema

The database schema is as follows:



Processing

Your program should proceed as described below. (There is an example session below.)

1 Connect to the database.

2 Repeat

2.1 Request a ticker symbol and optional start and end dates from `System.in`. The loop (step 2) exits and the program terminates when an empty string, or a string containing only spaces, is submitted as input. (The demo program has an example of how to do this.)

2.2 Retrieve the full company name from the company table and print it on the console. If the company is not found, indicate that the stock is not in the database and start the loop again by requesting user input.

2.3 Retrieve all the PriceVolume data in the input data range for the ticker. If no dates were specified, retrieve all PriceVolume data. Because the first analysis phase involves adjusting for splits, it is useful to request the data in reverse chronological order. For example, to retrieve the data for all dates for a ticker, you could use the following SQL, remembering to use a PreparedStatement, substituting the actual ticker for '?':

```
select *
  from PriceVolume
  where Ticker = ?
  order by TransDate DESC
```

2.4 To prepare for the investment strategy computation (2.6 and following), scan the data in reverse chronological order and identify stock splits, using the same criteria as in Assignment 1:

- A 2:1 stock split occurs if |closing/opening ratio – 2.0| < 0.20
- A 3:1 stock split occurs if |closing/opening ratio – 3.0| < 0.30
- A 3:2 stock split occurs if |closing/opening ratio – 1.5| < 0.15

2.5 To adjust for splits on a given day (meaning the split occurs between that day and the next day), all price data for the given day and earlier must be divided by 2 (or 3 or 1.5 depending on the split ratio). Each row of the PriceVolume table represents one trading day, so the open, high, low, and close prices for that day must be adjusted.

Note that after adjusting all price data on the given day, the algorithm must continue scanning to detect splits in the adjusted data. If another 2:1 split appears, for example, then earlier data, already adjusted for the first split, would again be divided by 2.

You should be able to accomplish all adjustments in one pass over the data, by keeping track of the total divisor. Initialize the divisor to one and adjust it upward as you encounter splits.

2.6 From this point forward, all references to price data refer to the adjusted data from the previous step. With the adjusted data stored in your program, scan forward in time to implement the following investment strategy. In the remaining steps, *d* will refer to a trading day, *d+1* (*d*-1) to the next (prior) trading day, and *close(d)*, *open(d)*, etc. to the closing, opening, etc. prices for day *d*.

2.7 Maintain a moving average of the closing prices over a 50-day window. So for a given trading day *d*, the *50-day average* is the average closing price for the 50 previous trading days (days *d-50* to *d-1*).

2.8 If there are less than 51 days of data, do no trading and report a net gain of zero and repeat from step 2 to get the next user input.

2.9 If there are more than 51 days of data, compute *50-day average* for the first fifty days. Proceeding forward from day 51 through the second-to-last trading day in the data set, execute the following strategy:

2.9.1 Track current cash and shares, both of which start at zero. When buying stock, cash decreases and shares increase. When selling stock, cash increases and shares decrease. When dividends are paid, cash increases. Since cash starts at zero, we must borrow money to buy the initial shares. Disregard this complication.

2.9.2 (Dividends) If there is a Dividend for the ticker on day *d* or earlier, but later than day *d-1* (on occasion, dividends have been paid on non-trading days), increase cash by the *dividend amount * current shares*.

2.9.3 (Buy criterion) If the *close(d) < 50-day average* and *close(d)* is less than *open(d)* by 3% or more (*close(d) / open(d)* <= 0.97), buy 100 shares of the stock at price *open(d+1)*.

2.9.4 (Sell criterion) If the buy criterion is not met and *current shares* >= 100 and *open(d) > 50-day average* and *open(d)* exceeds *close(d-1)* by 1% or more (open*(d) / close(d-1)* >= 1.01), sell 100 shares at price (*open(d) + close(d)*)/2.

2.9.5 (Transaction Fee) For either a buy or sell transaction, cash is reduced by a transaction fee of $8.00.

2.9.6 If neither the buy nor the sell criterion is met, do not trade on that day.

2.9.7 Regardless of trading activity, update *50-day average* to reflect the average over the last 50 days, and continue with day *d+1*.

2.10 After having processed the data through the second-to-last day, if there are any shares remaining, on the last day add *open(d) * shares remaining* to cash to account for the value of those remaining shares. (No transaction fee applies to this.)

Note on timing: On any given day the following events occur in this order:

1. If there are dividends payable, they are payable at the start of the day, before any trading occurs.
2. If there was a buy decision at the close yesterday, shares are bought at the opening price today.
3. If there is no buy, and the opening price meets the sell criteria, shares are sold during the day at the average of the opening and closing prices.
4. At the end of the day, the closing price is checked, and, if the price meets the buy criteria, shares will be bought at the opening price the next day in step 2.

Sample Output

Here is a sample run:

```
>java -cp .:mysql-connector.jar Assign2
Database connection jdbc:mysql://mysql.cs.wwu.edu/reedy330 reedyc2
established.
Enter ticker symbol [start/end dates]: INTC
Intel Corp.
2:1 split on 2000-07-28  129.13 --> 65.44
2:1 split on 1999-04-09  130.81 --> 61.63
2:1 split on 1997-07-11  153.81 --> 77.25
2:1 split on 1995-06-16  116.13 --> 58.50
2:1 split on 1993-06-04  112.75 --> 60.13
3:2 split on 1987-10-28  31.75 --> 21.75
6 splits in 7816 trading days
93 Dividends

Executing investment strategy
Transactions executed: 694
Net cash: 19821.18

Enter ticker symbol [start/end dates]: INTC 1980-01-01 1990-01-01
Intel Corp.
3:2 split on 1987-10-28  31.75 --> 21.75
1 splits in 1516 trading days
0 Dividends

Executing investment strategy
Transactions executed: 162
Net cash: 58826.84

Enter ticker symbol [start/end dates]: T
AT&T Inc
2:1 split on 1998-03-19  83.75 --> 42.13
```

```
2:1 split on 1993-05-25  74.75 --> 37.63
3:1 split on 1987-05-22  102.50 --> 36.00
3 splits in 7816 trading days
123 Dividends

Executing investment strategy
Transactions executed: 262
Net cash: 14514.36

Enter ticker symbol [start/end dates]: T 2000-01-01 2016-01-01
AT&T Inc
0 splits in 3773 trading days
60 Dividends

Executing investment strategy
Transactions executed: 126
Net cash: 8734.90

Enter ticker symbol [start/end dates]: GOOGL
Alphabet Inc Class A
2:1 split on 2014-04-02  1135.10 --> 573.39
1 splits in 2863 trading days
0 Dividends

Executing investment strategy
Transactions executed: 112
Net cash: 126247.25

Enter ticker symbol [start/end dates]:
Database connection closed.

>
```

Constraints

1. Do not build any lists, queues, etc. Use the ones that are available in java.util.
2. You must use PreparedStatements when you are using SQL statements where values are filled in. As described in class, this is basic good practice for avoiding SQL injection attacks. (This is a BIG security issue.)
3. You will likely need two SQL statements for reading the stock trading data from the database (step 2.3), one for no date range specified and one for a specified date range. You do not need to duplicate any of the remaining logic (2.4 and following) to handle those two separate conditions.
4. In order to help with round-off-error discrepancies (a) do all computations with `doubles`, not `floats`, and (b) use the code in the following table:

When the description says:	Use the following:
value <= 0.97	*value* < 0.97000001
value >= 1.01	*value* > 1.00999999

Hints

1. If you want to break up the work a little, you can use the following strategy:
   a) Make sure that you can run the demo program with your own connectparams.txt file.
   b) Reproduce the split results by reading in the PriceVolume data from the database.
   c) Get the Buy and Sell portions of the trading strategy working. Not some of the sample output above and the GOOGL transaction log have no dividends.
   d) Finally, get the dividends working.
2. If you are having problems getting a connection established, here are some possible problems to check:
   a) If your program terminates with a ClassNotFoundException, that is most likely either (1) the Java run time is not seeing the mysql-connector.jar file that is needed for JDBC to talk to MySQL or (2) you misspelled the name "com.mysql.jdbc.Driver" (capitalization is important here). See DatabaseAccess.pdf for more information about the first problem.
   b) If your program terminates with an SQLException with a message of the form "Access denied for user … (using password: YES)", that is most likely a problem with the userid or password. Check to make sure that these are correctly specified in your connection parameters file.
   c) If your program terminates with an SQLException with a message of the form "Access denied for user … to database ...", that most likely means that the userid and password are valid but the database name (`reedy330`) is wrong or the given userid doesn't have permission to access that database.
   d) If your program terminates with an SQLException with a message of the form "Communications link failure", that is a problem communicating with the database server. Check to make sure that the server is correctly specified and that you have the needed Internet connectivity.
3. Dates are stored in the database as character strings, not SQL date types. However, the date format (YYYY-MM-DD) means that string comparisons also give the right answer as date comparisons. So, there's no need in this assignment for you to do the work to decompose database dates, just continue to treat them as strings.
4. Here are two approaches for handling dividends: (1) Query the database for all the dividends for the stock and put this in a queue. Check for the dividend on the queue and work through the queue as dividends are paid. (2) Query the database for the dividends and use the ResultSet returned by executing the query as your queue. Move to the next result as each dividend is paid.
5. I have included a pair of transaction logs (INTCTransLog.txt and GOOGLTransLog.txt) from my version of the assignment showing all the data and all the trades and dividends to get the values shown in the sample output.

Design and Good Style

Similar to assignment 1, I will be grading your program for design and good style as well as correctness.

What to Submit

This assignment is due by midnight (end of the day), February 19. You only need to submit the source for your program. I don't need the `connectparams.txt` file—I will supply my own when testing your program.

Your assignment should be submitted on Canvas. If you want to submit a compressed file, please submit a .zip file. Do not submit a .tar, .tgz, .rar, .7z, or anything other than a zip.

Grading

Grading will be as follows:

- 60% – The program correctly executes the stock trading strategy.
- 10% – Program output contains the data (and just the data) shown in Sample Output
- 10% – Program obeys prepared statement constraint (Constraint #2)
- 10% – Program does not unnecessarily duplicate processing logic (Constraint #3)
- 10% – Program has good design and style (including Constraints #1 and #4)