CSCI 330 Database Systems
Fall 2016
Assignment 2 (Stock Investment Strategy)
Total Point: 50
Due: Nov 8 2016 (Tuesday), 11:59 pm

Goal of the assignment

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 data, write a Java program that computes the gain or loss from the trading strategy described below (see the section titled "Processing").

Connect to the Database

Each student has database account on the server ***mysql.cs.wwu.edu***. For this assignment you will need to access the database johnson330 on the ***mysql.cs.wwu.edu*** server. The credentials to access the database in that server will be provided. The section 5.1 of our book discussed in detail how to access the database from java. In order to connect to the database you will need a connection parameters file that looks like:

dburl=jdbc:mysql://mysql.cs.wwu.edu/johnson330
user=*user_name*
password=*user_password*

You should have read-only access to the johnson330 database.

Your program **must** use a file named ***"ConnectionParameters.txt"*** as the default file for the connection parameters.

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

1. Security. If you hardcode 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.

Database Schema

The database schema is as follows:

Entity	Attributes	Primary Key	Foreign keys
Company	Ticker Name Industry Location	Ticker	
PriceVolume	Ticker TransDate OpenPrice HighPrice LowPrice ClosePrice Volume AdjustedClose	Ticker TransDate	Ticker
Dividend	Ticker DivDate Amount	Ticker DivDate	Ticker

Processing

Your program should proceed as described 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.

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 symbol INTC, you could use the following SQL:

select * from PriceVolume where Ticker = 'INTC' 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, except with a larger buffer for market churn:

- 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. Since cash starts at zero, we must borrow money to buy the initial shares. Disregard this complication.

2.9.2 (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)*.

2.9.3 (Sell criterion) If the buy criterion is not met, then if 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.

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

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

2.9.6 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).

Sample Output

Here is a sample run:

```
Database connection jdbc:mysql://mysql.cs.wwu.edu/johnson330 username_reader established.
Enter a ticker symbol [start/end dates]: INTC
Intel Corp.
2:1 split on 2000.07.28  129.12 --> 65.44
2:1 split on 1999.04.09  130.81 --> 61.62
2:1 split on 1997.07.11  153.81 --> 77.25
2:1 split on 1995.06.16  116.12 --> 58.50
2:1 split on 1993.06.04  112.75 --> 60.12
3:2 split on 1987.10.28  31.75 --> 21.75
6 splits in 7470 trading days

Executing investment strategy
Transactions executed: 690
Net cash: 14717.72

Enter ticker symbol [start/end dates]: INTC 1980.01.01 1999.12.31
Intel Corp.
2:1 split on 1999.04.09  130.81 --> 61.62
2:1 split on 1997.07.11  153.81 --> 77.25
2:1 split on 1995.06.16  116.12 --> 58.50
2:1 split on 1993.06.04  112.75 --> 60.12
3:2 split on 1987.10.28  31.75 --> 21.75
5 splits in 3791 trading days

Executing investment strategy
Transactions executed: 358
Net cash: 44953.95

Enter ticker symbol [start/end dates]: T
AT&T Inc
2:1 split on 1998.03.19  83.75 --> 42.12
2:1 split on 1993.05.25  74.75 --> 37.62
3:1 split on 1987.05.22  102.50 --> 36.00
3 splits in 7470 trading days

Executing investment strategy
Transactions executed: 260
Net cash: 2028.67

Enter ticker symbol [start/end dates]: T 2000.01.01 2014.08.18
AT&T Inc
0 splits in 3679 trading days

Executing investment strategy
Transactions executed: 148
```

```
Net cash: -1568.00

Enter ticker symbol [start/end dates]: BAC
Bank of America Corp
2:1 split on 2004.08.27  89.01 --> 44.79
2:1 split on 1997.02.27  122.50 --> 61.25
2:1 split on 1986.11.20  42.62 --> 21.50
3 splits in 7116 trading days

Executing investment strategy
Transactions executed: 534
Net cash: 41846.00

Enter ticker symbol [start/end dates]: XX
XX not found in database.

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

Constraints

1. You must use PreparedStatements when you are using SQL statements where values are filled in. This is basic good practice for avoiding SQL injection attacks (This is a HUGE security issue).
2. 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.
3. 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 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-connection.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).
   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 (johnson330) is wrong or the given userid doesn't have permission to access that database. Make sure you're using your *user*_reader id.

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.

2. 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.

Good Design and Style

Although bad design and style won't hurt the performance of your program, it is utmost important, especially when you will work in the real world with other professionals, to follow a good design and style for coding. It will be helpful not only for your colleagues to understand your code better but also for you as well when you will revisit your code later.

Here are some guidelines for good design and style:

- The program is logically organized so that a reader can quickly and easily understand the program structure.
- Functions and variables have meaningful names.
- The program has proper indentation.
- The program follows object oriented approach.

Submission Instructions

- You only need to submit the source code.
- The file name which contains main method should be Assignment2.java.
- If you have more than one java files:
    - Please keep all java files in a single folder.
    - Folder name should be your last name.
    - Zip the folder and name it Assignment2.zip.
- Upload the Assignment2.java/ Assignment2.zip on canvas.

Grading (Total Points: 50)

Your program will be graded in the following way:

- 35 – The program correctly executes the stock investment strategy.
- 3 – The output is correctly formatted. Output contains the data (and just the data) shown in Sample Output.
- 3 – Program obeys prepared statement constraint (Constraint #1)
- 3 – Program does not unnecessarily duplicate processing logic (Constraint #2)
- 5 – Program has good design and style
- 1 – File name follows the rules mentioned above.

Late Policy

- Max 2 days after deadline are allowed
    - 1 day late: maximum 70% point
    - 2 days late: maximum 60% point

Academic Dishonesty

- Copying files from someone else and claiming they are your own is plagiarism.
- Providing files that you created to another student or being party to such actions also amounts to academic dishonesty.