

CSCI 330 Database Systems

Assignment 2 (Stock Investment Strategy)

Total Points: 60 (15% of course grade)

1. Goal of the assignment

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

2. 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**").

2.1 *ConnectionParameters.txt*

For this assignment, you will need to access the database called **johnson330**, hosted at **mysql.cs.wvu.edu**. You should have **read-only** access to the johnson330 database. We have already provided your credentials to access the database on that server. Section 5.1 of our book discussed how to access the database from java. To connect to the database, you need an external text file named **ConnectionParameters.txt**.

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

1. Security. If you hardcode the connection parameters into your program, the password is vulnerable to some clever person decoding the executable. Also, disassembler programs (translate machine language to assembly language) do all the hard work of doing this for you.
2. Testing. You do not 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.

If you are running your program from a lab computer, **ConnectionParameters.txt** should look as follows:

```
dburl=jdbc:mysql://mysql.cs.wvu.edu/johnson330
user=your_user_name
password=your_password_we_provided
```

Please ensure that this external text file (**ConnectionParameters.txt**) doesn't contain any extra lines at the end.

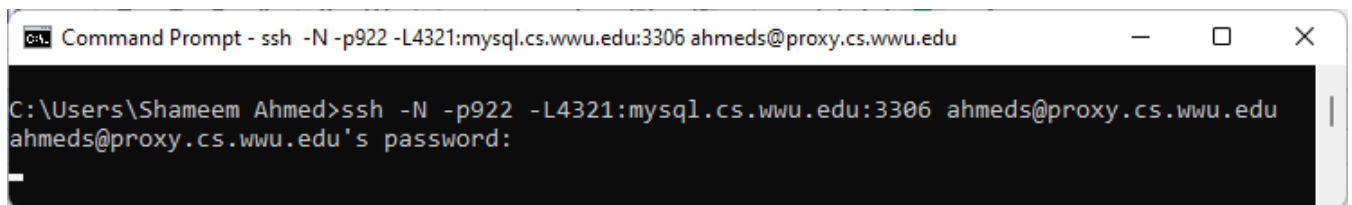
2.2 Remote Database Connection

To connect to the remote database server (mysql.cs.wvu.edu) from off-campus computers (e.g., your laptop), first, you will have to install VPN (Virtual Private Network). Here are the details on how to do that: <https://atus.wvu.edu/kb/vpn-virtual-private-network>

Second, you will have to set up SSH Tunneling using the following command in the terminal/command prompt/PowerShell.

```
ssh -N -p922 -L4321:mysql.cs.wvu.edu:3306 your_user_name@proxy.cs.wvu.edu
```

It will ask for a password; please use your CS password (which you use to log in to any lab computer). Here is a screenshot from Windows Command prompt.



```
Command Prompt - ssh -N -p922 -L4321:mysql.cs.wvu.edu:3306 ahmeds@proxy.cs.wvu.edu
C:\Users\Shameem Ahmed>ssh -N -p922 -L4321:mysql.cs.wvu.edu:3306 ahmeds@proxy.cs.wvu.edu
ahmeds@proxy.cs.wvu.edu's password:
```

Please notice that after you enter the password, the cursor will go to the next line and start blinking. It means that the connection is successful. There will be no additional message.

Since you are running your program from an off-campus computer, you need to change **ConnectionParameters.txt** in the following way:

```
dburl=jdbc:mysql://127.0.0.1:4321/johnson330
user=your_user_name
password=your_password_we_provided
```

3. Database Schema

The database schema is as follows:

Tables	Columns	Primary Key	Foreign keys
company	Ticker Name Industry Location	Ticker	
pricevolume	Ticker TransDate OpenPrice	Ticker TransDate	Ticker

	HighPrice LowPrice ClosePrice Volume AdjustedClose		
dividend	Ticker DivDate Amount	Ticker DivDate	Ticker

We only need to use the first two tables (company and pricevolume) for this assignment.

4. 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:
 - A 2:1 stock split occurs if $|C_x / O_{x+1} - 2.0| < 0.20$
 - A 3:1 stock split occurs if $|C_x / O_{x+1} - 3.0| < 0.30$
 - A 3:2 stock split occurs if $|C_x / O_{x+1} - 1.5| < 0.15$

Where C_x is the closing price on day x , and O_{x+1} is the opening price on the next trading day $x + 1$.

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
- $(d+1)$ will refer to the next trading day
- $(d-1)$ will refer to the prior trading day
- $close(d)$ closing price for day d .
- $open(d)$ opening price for day d and so on

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:

- 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.
- (Buy criterion) If the $close(d) < 50\text{-day average}$ and $close(d)$ is less than $open(d)$ by 3% or more ($close(d) / open(d) \leq 0.97$), buy 100 shares of the stock at price $open(d+1)$.
- (Sell criterion) If the buy criterion is not met, then if $shares \geq 100$ and $open(d) > 50\text{-day average}$ and $open(d)$ exceeds $close(d-1)$ by 1% or more ($open(d) / close(d-1) \geq 1.01$), sell 100 shares at price $(open(d) + close(d))/2$.

- (Transaction Fee) For either a buy or sell transaction, cash is reduced by a transaction fee of \$8.00.
- If neither the buy nor the sell criterion is met, do not trade on that day.
- Regardless of trading activity, update the *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).

5. Sample Output

Database connection is 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.

6. Constraints

1. You **must** use PreparedStatements when you are using SQL statements where values are filled in. This is a good practice for avoiding SQL injection attacks (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 remaining logic (2.4 and following from section 4) to handle those two separate conditions.
3. 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:
<i>value</i> <= 0.97	<i>value</i> < 0.97000001
<i>value</i> >= 1.01	<i>value</i> > 1.00999999

7. Hints

1. If you are having problems getting a connection established, here are some possible problems to check:
 - a) 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 ensure that these are correctly specified in your connection parameters file.
 - b) 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 *userid*.
 - c) 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 ensure 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 correct answer as date comparisons. Therefore, for this assignment, you don't need to decompose database dates; just continue to treat them as strings.

8. Good Design and Style

Although bad design and style will not hurt the performance of your program, it is of utmost importance, especially when you will work in the real world with other professionals, to follow 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 a proper indentation.
- The program follows an object-oriented approach.

9. Point Distribution

Grading Criteria	Points
The program gives the correct output	48
The program obeys the constraints	5
The program has good design and style	2.5
The write up is complete (YourLastName-Assignment2.pdf)	2.5
The output format is correct	1
The filename format is correct	1

For the correct output (48 points), we will check the output of your program with six separate companies (8 points for each company). Suppose, if we check your program with INTC, then we will distribute the 8 points in the following way:

```

Database connection jdbc:mysql://mysql.cs.wvu.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

```

1 point

1.5 points

1.5 points

4 points

10. Submission Instructions

- You have to submit the following on Canvas.
 - Java source code.
 - The file name containing the main method should be **YourLastNameAssignment2.java**.
 - If you have more than one java file:
 - Please keep all java files in a single folder.
 - Zip the folder and name it **YourLastName-Assignment2.zip**.
 - A pdf file (Filename format: **YourLastName-Assignment2.pdf**) with the following write-up (**If you have a zip file for your java source code in the previous step, please DON'T include the PDF file in your zip file**):
 - Your full name
 - Does your program compile correctly? Yes/No
 - Does your program run correctly? Yes/No

- An acknowledgment and discussion of any parts of the program that are not working. Failure to disclose obvious problems will result in additional penalties.
- An acknowledgment and discussion of any parts of the program that appear to be inefficient (in either time or space complexity).
- How many hours did you spend on completing this assignment?
- Your expected score (please fill out the following table)

Grading Criteria	Points	Your Expected Score
The program gives the correct output	48	
The program obeys the constraints	5	
The program has good design and style	2.5	
The write up is complete (YourLastName-Assignment2.pdf)	2.5	
The output format is correct	1	
The filename format is correct	1	

- Upload the following **two files** on canvas
 1. **YourLastNameAssignment2.java** or **YourLastName-Assignment2.zip**
 2. **YourLastName-Assignment2.pdf**

11. Debugging Tips

1. Since this assignment deals with millions of data points, it is tough to determine the cause of the error if you get an incorrect output. Hence, for this assignment (and for the next assignment as well), you need to know how to debug the java code efficiently. Printing messages on the console might not be sufficient to find the problem.

There are tons of tutorials and YouTube videos available on the web to learn how to use the debugger for Eclipse, jGrasp, IntelliJ, and jdb. Please feel free to look at the tutorials to learn how to do the debugging using your favorite java editor.

2. There are two log files (in .txt format) available on canvas. Besides using the debugging tool, you may look at these log files to find the problem if your program does not provide the correct output. These two log files are as follows:
 - transLog-INTC-1980.01.01-1999.12.31.txt
 - transLog-T-All Dates.txt

The first log file contains the stock data for **Intel Corporation** from Jan 1, 1980 to December 31, 1999 and the second log file contains the stock data for **AT&T** for all dates available. Each line of these two log files has the following information:

1. date
 - Example: 1985.01.02
2. opening price

- Example: open: 1.1666667
- 3. high price
 - Example: high: 1.1770833
- 4. low price
 - Example: low: 1.1354167
- 5. closing price
 - Example: close: 1.1458333
- 6. 50-day average [After first fifty days]
 - Example: average 1.2229167
- 7. Buy stocks [If buy condition fulfills]
 - Example: Buy: 1985.03.26 100 shares @ 1.0675000, total shares = 100, cash = -114.7500000
- 8. Sell stocks [If sell condition fulfills]
 - Example: Sell: 1999.11.22 100 shares @ 80.6900000, total shares = 0, cash = 44319.9479167
- 9. Final Sale [At the end of the file]
 - Example: Final sale: 1999.12.31 0 shares @ 83.8100000, cash = 44953.9479167 (average = 77.8544000)

12. Late Policy

No late work will be accepted.

13. Academic Dishonesty

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