CS 330: Database Systems
Winter, 2014
Program 2: Stock Investment Strategy
Due: February 21, 2014 (23:59)

Each student received three database accounts on the server `db.cs.wwu.edu`. The account credentials have the following form.

- `username_reader password1`
- `username_writer password2`
- `username password3`

For this assignment, you should construct a connection string as follows:

```
jdbc:mysql://db.cs.wwu.edu/CS330_201410  username_reader  password1
```

Put the connection string in a file where your program can read it. The format must be followed exactly: three strings separated by whitespace. The first string specifies the connection protocol, `jdbc`, which you might remember as "Java database connector," although the Java authorities insist that the string is not an acronym. That first string continues with the database model implementation, `mysql`, the database location, `db.cs.wwu.edu`, and finally the database name, `CS330_201410`. The second and third strings are your user account name and password.

You connect to `CS330_201410` with the following Java statements, substituting your user-name and password as appropriate.

```
String dbAccess = "jdbc:mysql://db.cs.wwu.edu/CS330_201410";
String user = "jones";
String password = "X45YY845Z";
Connection conn = DriverManager.getConnection(dbAccess, user, password);
```

Although you could hardcode this sequence in your program, that practice would likely leave your password vulnerable. Instead, your program should read the connection string from a file, split it on whitespace into three segments, and invoke the `getConnection` function with those three string arguments. If the program throws an exception at this point, the most likely reason is a missing J-connector. Consult the "Computer Issues" download to resolve this issue.

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

Your program proceed as follows. The main driver will, after connecting to the database, enter a loop that repeatedly requests a ticker symbol from the keyboard. The loop exits and the program terminates when an empty string, or a string containing only spaces, is submitted as a ticker symbol.

For each ticker symbol, the program first retrieves the full company name from the `company` table and prints it on the console (see example output at the end of this document). If the company is not found, the program indicates that the stock is not in the database and asks the user for a new ticker symbol. Otherwise, the program continues, retrieving all of the pricevolume data for that ticker. 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 ticker symbol INTC, you would use the following SQL:

```
select * from pricevolume where ticker = 'INTC' order by TransDate DESC
```

To prepare for the investment strategy computation, scan the data in reverse chronological order and adjust for splits, using the same criteria as in Assignment 1, except with a larger buffer for opening market churn. For example, deduce that the stock undergoes a 2:1 split on day x if

$$\left| \frac{\text{close}(x)}{\text{open}(x+1)} - 2.0 \right| \quad < \quad 0.13,$$

and similarly for 3:1 and 3:2 splits. To adjust for this abrupt change, all price data for day x and earlier must be divided by 2 (or 3 or 1.5 for the other split ratios). Each row of the pricevolume table represents one trading day. Note that after adjusting all price data at day x and earlier, the algorithm must continue the backward scan 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 a multiplicative factor. Initialize the factor to one and adjust it downward as you encounter splits. All references to price data in the next paragraph refer to the adjusted data.

Now, with the adjusted data stored in your program, you scan *chronologically* to implement the following investment strategy. We keep a moving average of the closing price over a 50-day window; the explanation below refers to this value as avg. If there are less than 51 days of data, we simply do no trading and report a net gain of zero. Otherwise, we proceed forward from day 51 through the second-to-last trading day in the data set. At each point, say day x, we compute as follows.

If close(x) $<$ avg and close(x) is lower than open(x) by 3% or more, we buy 100 shares of the stock at price open(x+1). We keep track of our cash and shares, both of which start at zero. This starting point means that we must borrow money to buy our initial shares, but we disregard this complication. We will simply compare our eventual net gain with our initial zero position to determine the value of this strategy.

If the criterion above is not met, then we check to see if we should sell. If shares $\geq$ 100 and open(x) $>$ avg and open(x) exceeds close($x-1$) by 1% or more, then we sell 100 shares at price (open(x) + close(x))/2.

Of course, each buy decreases cash and increases shares, while each sell increases cash and decreases shares. Moreover, for either a buy or sell transaction, our cash diminishes by a transaction fee of $8.00.

If neither the buy nor the sell criterion is met, we do not trade on that day. However, regardless of our trading activity, we update avg to reflect the average over the last 50 days, and then we move on to day $x+1$.

When we have scanned the data set through the second-to-last day, our net gain is our cash plus the opening value of our shares on the last day of the data set. The next page has a sample run that illustrates the proper output form.

```
Database connection established: jdbc:mysql://db.cs.wwu.edu/CS330_201410 jimj  xxxxxxxxx
Enter a ticker symbol: INTC
Intel Corp.
Adjusting data for splits
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;  116.12 -->   58.50
2:1 split on 1993.06.04;  112.75 -->   60.13
3:2 split on 1987.10.28;   31.75 -->   21.75

Executing investment strategy
Transactions executed: 626
Net gain: 14484.53

Enter a ticker symbol: XX
XX not found in database

Enter a ticker symbol: C
Citigroup Inc.
Adjusting data for splits
3:2 split on 2009.02.26;    2.46 -->    1.56
3:2 split on 1999.05.28;   66.25 -->   43.88
3:2 split on 1997.11.19;   73.44 -->   49.69
3:2 split on 1996.05.24;   62.38 -->   41.25
3:2 split on 1993.02.26;   54.88 -->   37.00
2:1 split on 1987.03.12;   94.50 -->   48.25

Executing investment strategy
Transactions executed: 592
Net gain: 6911.00

Enter a ticker symbol: BAC
Bank of America Corp
Adjusting data for splits
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;   42.63 -->   21.50

Executing investment strategy
Transactions executed: 532
Net gain: 40807.38

Enter a ticker symbol:
Database connection closed
```