Lab Exercise

Name: Yuan Qu RUID: 176007913

System Environment

Computer: MacBook (Retina, 12-inch, Early 2015)

CPU: 1.1 GHz Intel Core M

Memory: 8 GB 1600 MHz DDR3

Operation System: macOS Sierra 10.12.3

MySQL: 5.7.14

Apache: Apache/2.4.25 (Unix)

Python: 2.7.10

GCC: 4.2.1

Browser: Safari 10.0.3

PHP: 5.6.28

Bootstrap: 3.3.2

Initialization

1. Install and Configure Apache, MySQL, Python separately.
2. Configure PhpMyAdmin, a web tools for MySQL management.

   This step make sure that Apache and MySQL work.
3. Configure CGI in ‘httpd.conf’ and test Python CGI program.

   This step open the CGI and make sure that Python script can run.
4. Configure front end library Bootstrap and ‘Hightlight.js’.
5. Configure Python-MySQL connection by package ‘MySQLdb’.

   This step make sure that Python can execute operations in MySQL.

File Index

- File root
  - CGI: the files in CGI folder
    - test.py: main function, get and post HTTP response, call “conncet_mysql”.
    - mysqltest.py: define function “conncet_mysql” that execute operation in MySQL.
  - SQL
    - bdmlab.sql: the SQL code to build the database.
    - test.sql: the SQL queries of questions.
  - Web
    - bootstrap: bootstrap resources.
    - css: css files.
    - fonts: font resources.
    - highlight: highlight resources.
    - images: image recources.
    - js: javascript files.
    - plugins: some font end plugins.
    - index.html: homepage, the entry of the system.
    - result.html: the basic html structure of result page that generate by CGI.
  - This document.

Part I

Initialized, we have 3 tables:

STOCK

ticker	exchange
the stock's ticker symbol; e.g. GOOG, AAPL, GE	the exchange where the ticker is listed; e.g. NYSE, NASDAQ

PRICE

ticker	date	close
the stock's ticker symbol	the date of the price information	the closing price of the stock

BUYnSELL

buy_or_sell	ticker	date	timestamp	value	num_of_shares
BUY' or 'SELL'	the stock's ticker symbol	the date of the price information	time of transaction	the price of a single share	number of shares (bought or sold)

6. Find the tickers and closing prices of all stocks exchanged in 2017 (careful with this question

$$\pi_{ticker,close}\left(\sigma_{ticker=ticker1}\left(\rho_{ticker1}\left(\sigma_{date>='2017-01-01'}BUYnSELL\right)\times PRICE\right)\right)$$

7. Find the tickers whose closing price is both higher than 'IBM' on '3/20/2017' and no higher than 'GOOG' on the same date.

   Set P_1 implies: $\rho_{close1}\left(\pi_{close}\left(\sigma_{date='2017-03-20',ticker='IBM'}PRICE\right)\right)$

   Set P_2 implies: $\rho_{close2}\left(\pi_{close}\left(\sigma_{date='2017-03-20',ticker='GOOG'}PRICE\right)\right)$

   Final expression: $\pi_{ticker}\left(\sigma_{P_1<close\vee close<=P_2}PRICE\right)$

8. Find the tickers of all stocks that closed at the highest price on '3/20/2017'. (we are asking for "all stocks" since there may be more than one with the same "highest price")

Set S_1 implies: $\pi_{ticker,close}\left(\sigma_{date='2017-03-20'}PRICE\right)$

Set S_2 implies: $\rho_{ticker1,close1}\left(\pi_{ticker,close}\left(\sigma_{date='2017-03-20'}PRICE\right)\right)$

Final: $\pi_{ticker}\left(\sigma_{date='2017-03-20'}PRICE\right)-\pi_{ticker}\left(\sigma_{close<close1}\left(S_1\times S_2\right)\right)$

9. Find the tickers of all stocks in 'NYSE' whose closing price on '3/20/2017' was either strictly below $20 or above $100

Set T_1 implies: $\rho_{ticker1}\left(\pi_{ticker}\left(\sigma_{exchange='NYSE'}STOCK\right)\right)$

Final: $\pi_{ticker}\left(\sigma_{ticker=ticker1}\left(\left(\sigma_{date='2017-03-20',close<20\vee 100<close}PRICE\right)\times T_1\right)\right)$

10. Find the tickers of the stocks whose closing price showed the highest increase between '3/20/2017' and '3/21/2017' in 'NYSE' and whose closing price was above $100 for the entire 2017 (we are asking for "all stocks" since there may be more than one with the same increase)

Set M_1 implies:

$$\sigma_{ticker=ticker1}\left(\sigma_{date='2017-03-20'\vee date='2017-03-21'}PRICE\times\rho_{ticker1}\left(\sigma_{exchange='NYSE'}STOCK\right)\right)$$

Set M_2 implies:

$$\pi_{ticker}\left(M_1-\left(\sigma_{date>='2017-01-01',close<=100}PRICE\right)\right)$$

which means the stock <u>in 'NYSE' and whose closing price was above $100 for the entire 2017</u>

Set M_3 implies:

$$\pi_{ticker}\left(\sigma_{\substack{date1='2017\text{-}03\text{-}20'\wedge date2='2017\text{-}03\text{-}21'\wedge \\ date3='2017\text{-}03\text{-}20'\wedge date4='2017\text{-}03\text{-}21'\wedge \\ ticker1=ticker2\wedge ticker3=ticker4\wedge \\ ticker=ticker1\wedge ticker=ticker3\wedge \\ exchange='NYSE'\wedge \\ close4-close3>close2-close1}}\left(\begin{array}{l} STOCK \\ \times\rho_{ticker1,date1,close1}PRICE \\ \times\rho_{ticker2,date2,close2}PRICE \\ \times\rho_{ticker3,date3,close3}PRICE \\ \times\rho_{ticker4,date4,close4}PRICE \end{array}\right)\right)$$

which means the stock in ‘NYSE’ and closing price **NOT** showed the highest increase between ‘3/20/2017’ and ‘3/21/2017’

So, we could have $M_2 - M_3$, which implies the tickers of the stocks whose closing price showed the highest increase between ‘3/20/2017’ and ‘3/21/2017’ in ‘NYSE’ and whose closing price was above $100 for the entire 2017.

Part II—Database Part

First, we create the database like the sample, with some additional record that makes some queries have the result.

1. Create the database.

```
19  --
20  -- Database: `bdmlab`
21  --
22  CREATE DATABASE IF NOT EXISTS `bdmlab`
23  USE `bdmlab`;
24
```

2. Create the table STOCK.

Consider that the code of stock (in this case, the ticker name) will not repeat among the different exchange place. Anyway, we could also set the combination as the key.

```
109  --
110  -- Table Structure `stock`
111  --
112
113  DROP TABLE IF EXISTS `stock`;
114  CREATE TABLE IF NOT EXISTS `stock` (
115    `ticker` varchar(6) NOT NULL,
116    `exchange` varchar(8) NOT NULL
117  ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
151  --
152  -- Indexes for table `stock`
153  --
154  ALTER TABLE `stock`
155    ADD PRIMARY KEY (`ticker`);
```

3. Insert data into the table STOCK.

The new data GE is for Question 4 & 5.

```
--
-- Truncate before the insert `stock`
--

TRUNCATE TABLE `stock`;
--
-- Insert data into `stock`
--

INSERT INTO `stock` (`ticker`, `exchange`) VALUES
('AAPL', 'NASDAQ'),
('GOOG', 'NASDAQ'),
('MSFT', 'NASDAQ'),
('IBM', 'NYSE'),
('GE', 'NYSE');
```

4. Create table PRICE.

Set (ticker, date) as key because every stock has a close price at each day.

```
--
-- Table Structure `price`
--

DROP TABLE IF EXISTS `price`;
CREATE TABLE IF NOT EXISTS `price` (
  `ticker` varchar(6) NOT NULL,
  `date` date NOT NULL,
  `close` float NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
--
-- Indexes for table `price`
--
ALTER TABLE `price`
  ADD PRIMARY KEY (`ticker`,`date`);
```

5. Insert data into the table PRICE.

The new data GE is for Question 4 & 5.

```
--
-- Truncate before the insert `price`
--

TRUNCATE TABLE `price`;
--
-- Insert data into `price`
--

INSERT INTO `price` (`ticker`, `date`, `close`) VALUES
('AAPL', '2017-03-20', 100),
('AAPL', '2017-03-21', 101.5),
('AAPL', '2017-03-22', 106.5),
('GOOG', '2017-03-20', 100),
('GOOG', '2017-03-21', 130),
('GOOG', '2017-03-22', 110),
('MSFT', '2017-03-20', 184.5),
('MSFT', '2017-03-21', 188.5),
('MSFT', '2017-03-22', 210),
('IBM', '2017-03-20', 72),
('IBM', '2017-03-21', 70),
('IBM', '2017-03-22', 10),
('GE', '2017-03-20', 101.1),
('GE', '2017-03-21', 102.2);
```

6. Create table BUYnSELL.

Add new meaningless attributes ID, which is the key because there could be a contradiction even if set all combination as the key (two same deals make at same time). Just regard ID as an operation number of the dealing log.

```
CREATE TABLE IF NOT EXISTS `buynsell` (
  `ID` int(4) NOT NULL,
  `ticker` varchar(6) NOT NULL,
  `buy_or_sell` varchar(5) NOT NULL,
  `date` date NOT NULL,
  `timestamp` time NOT NULL,
  `value` float NOT NULL,
  `num_of_shares` int(6) NOT NULL
) ENGINE=MyISAM AUTO_INCREMENT=17 DEFAULT CHARSET=utf8;
```

```
--
-- Indexes for table `buynsell`
--
ALTER TABLE `buynsell`
  ADD PRIMARY KEY (`ID`);
```

```
--
-- AUTO_INCREMENT for table `buynsell`
--
ALTER TABLE `buynsell`
  MODIFY `ID` int(4) NOT NULL AUTO_INCREMENT,AUTO_INCREMENT=17;
```

7. Insert data into the table BUYnSELL.

The new data which ID is 16 about AAPL is used for the additional question.

```
--
-- Insert data into `buynsell`
--

INSERT INTO `buynsell` (`ID`, `ticker`, `buy_or_sell`, `date`, `timestamp`, `value`, `num_of_shares`) VALUES
(1, 'IBM', 'BUY', '2017-03-20', '11:55:00', 273, 1100),
(2, 'IBM', 'BUY', '2017-03-21', '10:45:00', 271, 2400),
(3, 'IBM', 'SELL', '2017-03-22', '12:09:00', 270.5, 2500),
(4, 'GOOG', 'BUY', '2017-03-20', '12:22:00', 86, 2200),
(5, 'GOOG', 'SELL', '2017-03-20', '14:00:00', 87, 1000),
(6, 'GOOG', 'SELL', '2017-03-21', '10:22:00', 87.5, 1000),
(7, 'GOOG', 'BUY', '2017-03-21', '13:28:00', 87, 800),
(8, 'GOOG', 'SELL', '2017-03-22', '11:45:00', 86, 500),
(9, 'AAPL', 'BUY', '2017-03-20', '10:01:00', 99, 1000),
(10, 'AAPL', 'BUY', '2017-03-20', '11:22:00', 99.5, 1000),
(11, 'AAPL', 'BUY', '2017-03-21', '14:22:00', 100, 1000),
(12, 'AAPL', 'SELL', '2017-03-22', '19:01:08', 1030, 3000),
(13, 'MSFT', 'BUY', '2017-03-20', '11:45:00', 186, 1500),
(14, 'MSFT', 'SELL', '2017-03-21', '10:45:00', 188, 1000),
(15, 'MSFT', 'BUY', '2017-03-22', '12:03:00', 187, 5000),
(16, 'AAPL', 'SELL', '2017-03-22', '19:37:36', 105, 5000);
```

Part II—SQL query Part

Second, we realize the five queries in 'PART I' and an additional query in SQL.

1. Find the tickers and closing prices of all stocks exchanged in 2017 (careful with this question).

The code:

```
/* Q1 */
SELECT DISTINCT P.ticker, P.close, B.timestamp
FROM buynsell B, price P
WHERE B.ticker=P.ticker AND
    B.date=P.date AND
    B.timestamp>'2017-01-01 00:00:00';
```

The result:

16 rows

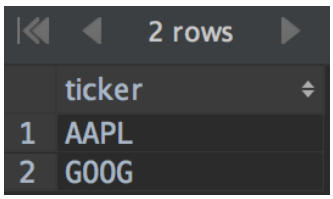
	ticker	close	timestamp (yyyy-MM-dd HH:mm:ss)
1	IBM	72	2017-03-20 11:55:00
2	IBM	70	2017-03-21 10:45:00
3	IBM	10	2017-03-22 12:09:00
4	GOOG	100	2017-03-20 12:22:00
5	GOOG	100	2017-03-20 14:00:00
6	GOOG	130	2017-03-21 10:22:00
7	GOOG	130	2017-03-21 13:28:00
8	GOOG	110	2017-03-22 11:45:00
9	AAPL	100	2017-03-20 10:01:00
10	AAPL	100	2017-03-20 11:22:00
11	AAPL	101.5	2017-03-21 14:22:00
12	AAPL	106.5	2017-03-26 19:01:08
13	MSFT	184.5	2017-03-20 11:45:00
14	MSFT	188.5	2017-03-21 10:45:00
15	MSFT	210	2017-03-22 12:03:00
16	AAPL	106.5	2017-03-22 19:37:36

2. Find the tickers whose closing price is both higher than ‘IBM’ on ‘3/20/2017’ and no higher than ‘GOOG’ on the same date.

The code:

```
 9  /* Q2 */
10  SELECT DISTINCT P.ticker
11  FROM price P
12  WHERE P.close > (
13      SELECT DISTINCT P_1.close
14      FROM price P_1
15      WHERE P_1.date='2017-03-20'
16      AND P_1.ticker='IBM')
17  AND P.close <= (
18      SELECT DISTINCT P_2.close
19      FROM price P_2
20      WHERE P_2.date='2017-03-20'
21      AND P_2.ticker='GOOG');
```

The result:

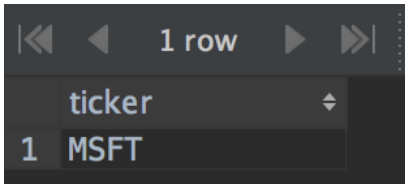


3. Find the tickers of all stocks that closed at the highest price on '3/20/2017'.
   (we are asking for "all stocks" since there may be more than one with the same "highest price".

The code:

```
/* The Q3 */
SELECT DISTINCT P.ticker
FROM price P LEFT JOIN
    (SELECT DISTINCT P_1.ticker
        FROM price P_1, price P_2
        WHERE P_1.date='2017-03-20'
        AND P_2.date='2017-03-20'
        AND P_1.close<P_2.close) AS S
    USING (ticker)
WHERE P.date='2017-03-20' AND S.ticker is NULL;
```

The result:

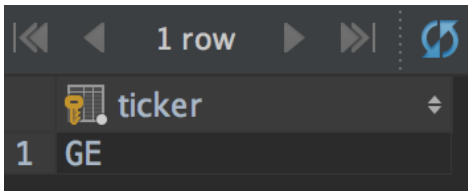


4. Find the tickers of all stocks in ‘NYSE’ whose closing price on ‘3/20/2017’ was either strictly below $20 or above $100

The code:

```
44  /* Q4 */
45  /*
46      The empty result,
47      ADD record in PRICE: GE 2017-03-20 101.1
48  */
49  SELECT DISTINCT S.ticker
50  FROM stock S, price P
51  WHERE S.exchange='NYSE'
52  AND (P.close>100 OR P.close<20)
53  AND P.date='2017-03-20'
54  AND P.ticker = S.ticker;
```

The result: (the new added record)

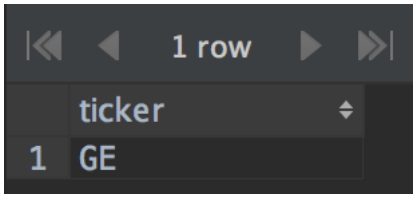


5. Find the tickers of the stocks whose closing price showed the highest increase between '3/20/2017' and '3/21/2017' in 'NYSE' and whose closing price was above $100 for the entire 2017 (we are asking for "all stocks" since there may be more than one with the same increase)

The code:

```
/* Q5 */
/* The empty result, ADD record in PRICE: GE 2017-03-21 200 */
SELECT DISTINCT N.ticker FROM
(SELECT DISTINCT M.ticker FROM
    (SELECT DISTINCT P.ticker, P.date, P.close
    FROM stock S, price P
    WHERE S.ticker = P.ticker
    AND S.exchange = 'NYSE'
    AND (P.date = '2017-03-20' OR P.date = '2017-03-21')) AS M
    LEFT JOIN
    (SELECT DISTINCT P_1.ticker
        FROM price P_1
        WHERE P_1.date>'2017-01-01'
        AND P_1.close<100) AS T
    USING (ticker)
WHERE T.ticker is NULL) AS N
LEFT JOIN
(SELECT DISTINCT P_1.ticker
    FROM price P_1, price P_2, price P_3, price P_4, stock S
    WHERE P_1.date='2017-03-20'
    AND P_2.date='2017-03-21'
    AND P_3.date='2017-03-20'
    AND P_4.date='2017-03-21'
    AND P_1.ticker=P_2.ticker
    AND P_3.ticker=P_4.ticker
    AND P_1.ticker=S.ticker
    AND P_3.ticker=S.ticker
    AND S.exchange='NYSE'
    AND P_4.close-P_3.close>P_2.close-P_1.close) AS K
USING (ticker)
WHERE K.ticker is NULL;
```

The result:

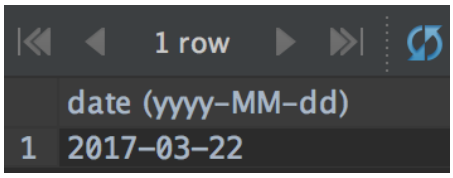


6. (The addition question) Find the dates where the total price (i.e. price times num of shares) of 'AAPL' the firm (i.e. the trading firm which is using this database) sold was higher than what the firm bought in 'NASDAQ'.

The code:

```
 99  /* The Q6 Version 2*/
100  /* ADD record in buynsell:
101  'AAPL','SELL','2017-03-22','19:42:00',102,5000
102  */
103  SELECT DISTINCT N.date FROM
104  (SELECT DISTINCT B.date, SUM(B.value*B.num_of_shares) AS SUM
105  FROM buynsell B
106  WHERE B.ticker='AAPL'
107  AND B.buy_or_sell='SELL'
108  GROUP by date) AS N,
109  (SELECT DISTINCT B.date, SUM(B.value*B.num_of_shares) AS SUM
110  FROM buynsell B, stock S
111  WHERE S.exchange='NASDAQ'
112  AND B.ticker=S.ticker
113  AND B.buy_or_sell='BUY'
114  GROUP by date) AS M
115  WHERE M.date=N.date
116  AND N.SUM > M.SUM
```

The result:

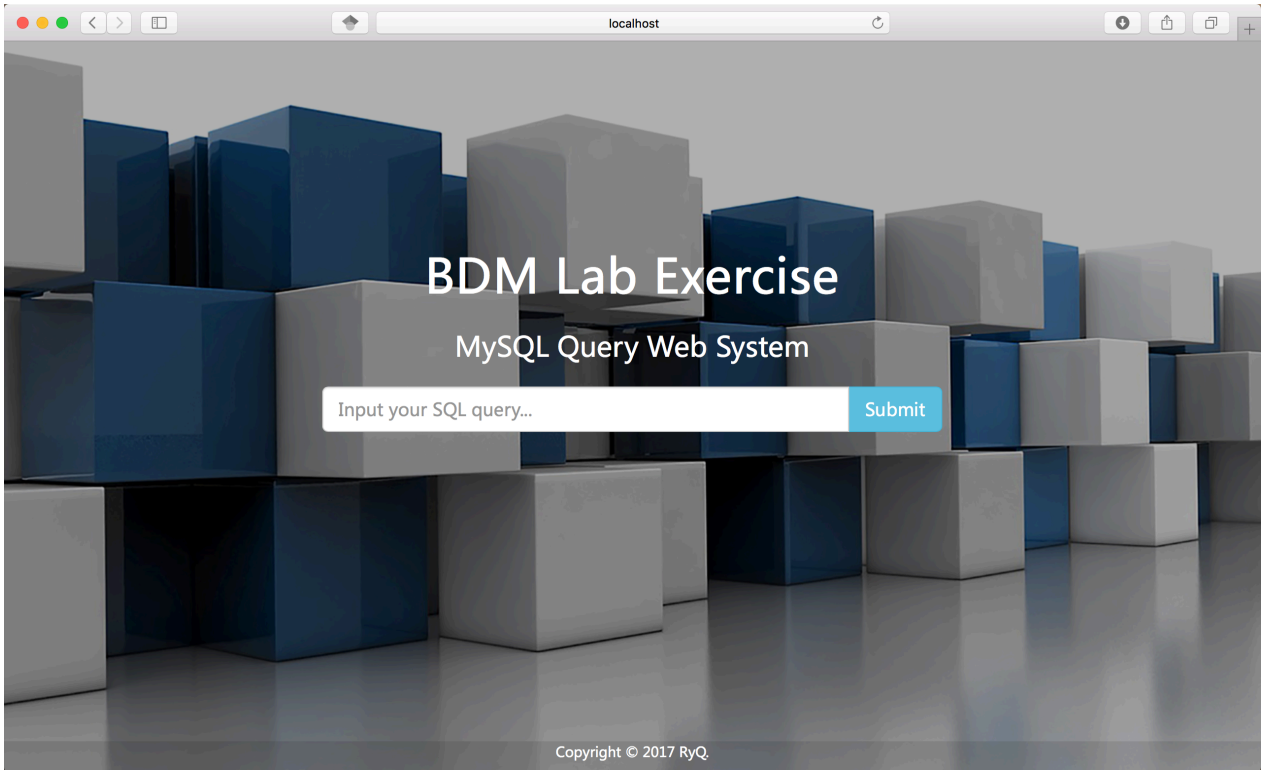


The reason of this query can't be explained in relational algebra is that the function SUM in SQL query can't be explained in relational algebra precisely.

If the table has N rows, the SUM function is equal to join N tables and select the sum of the a row that each two elements are different, which is hard to explained in relational algebra. To make sure that "each two elements are different", it will need $C_N^2 = \frac{N \times (N-1)}{2}$ constraints in WHERE clause.

Part II—Website Part

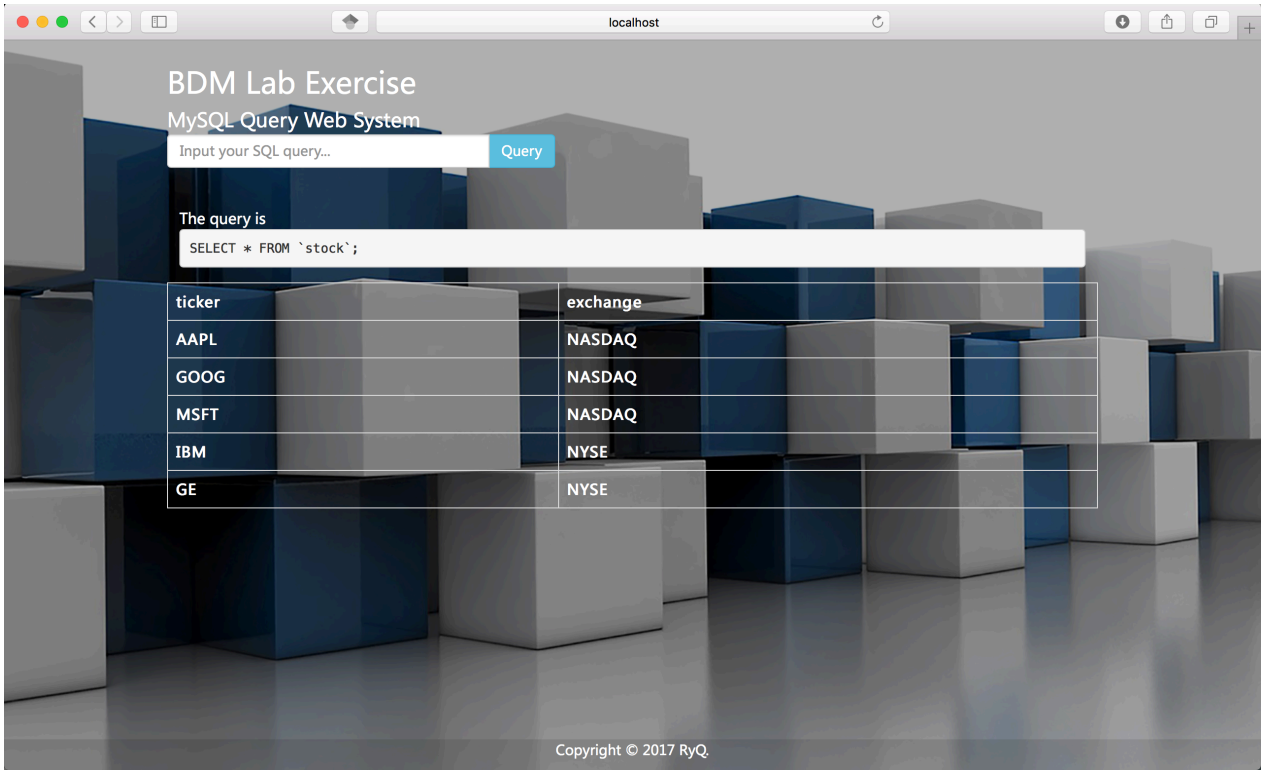
The home page:



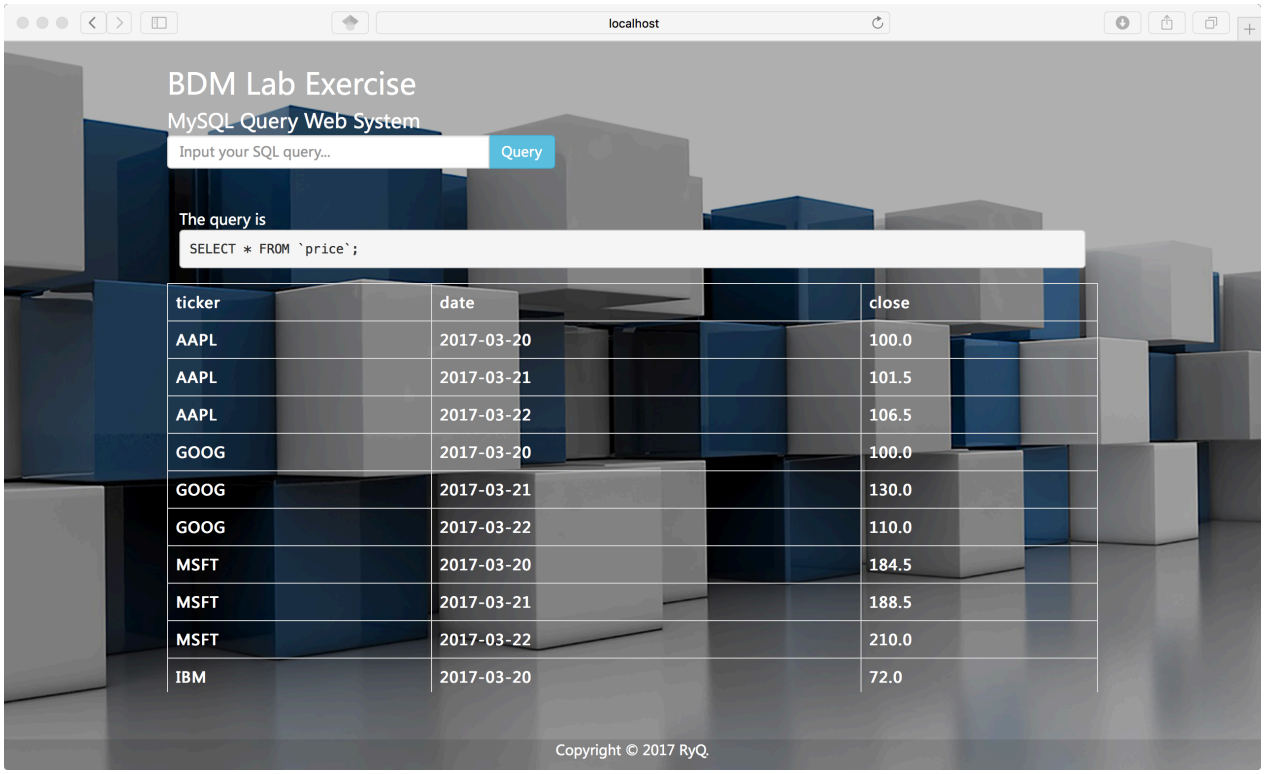
The basic query of three tables:

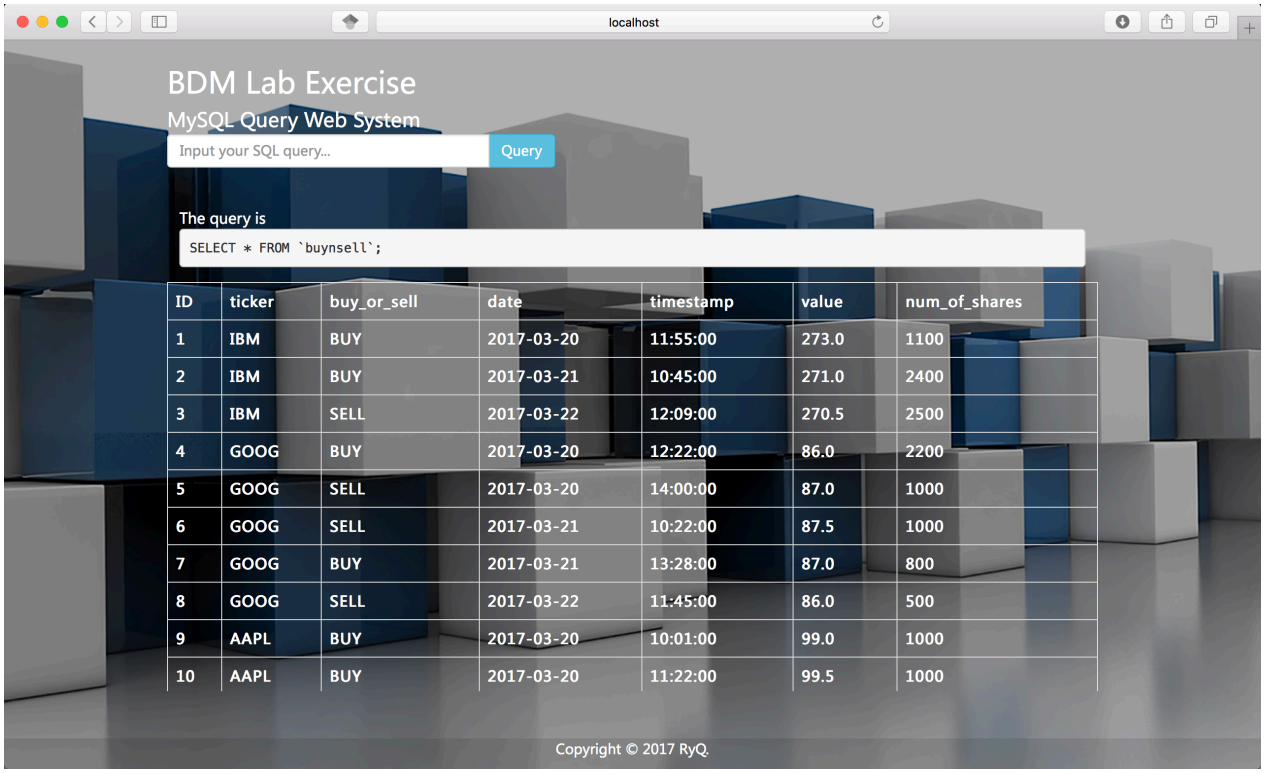
The query will show above the result, and the query text box can be query again.

Because of the height limit, the screen shot may not contain all the rows in result.

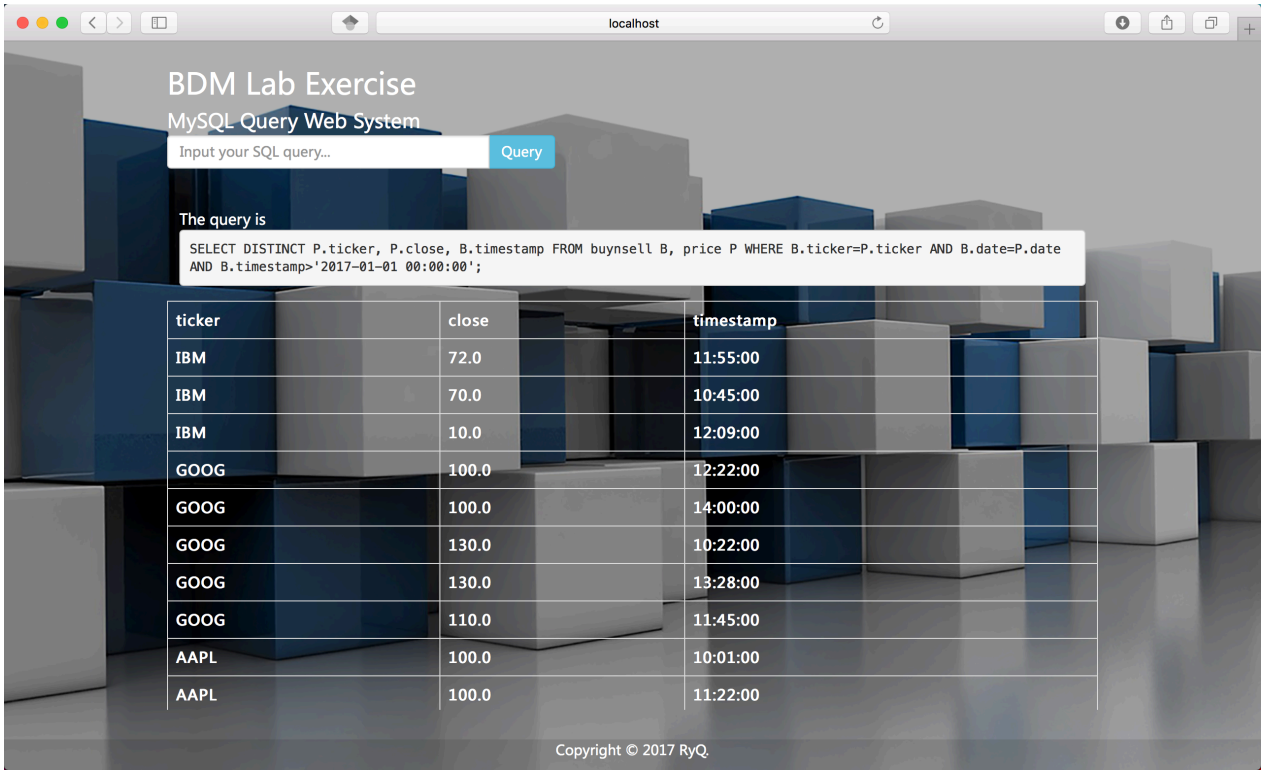


The scroll bar inside the table can view the entire table.

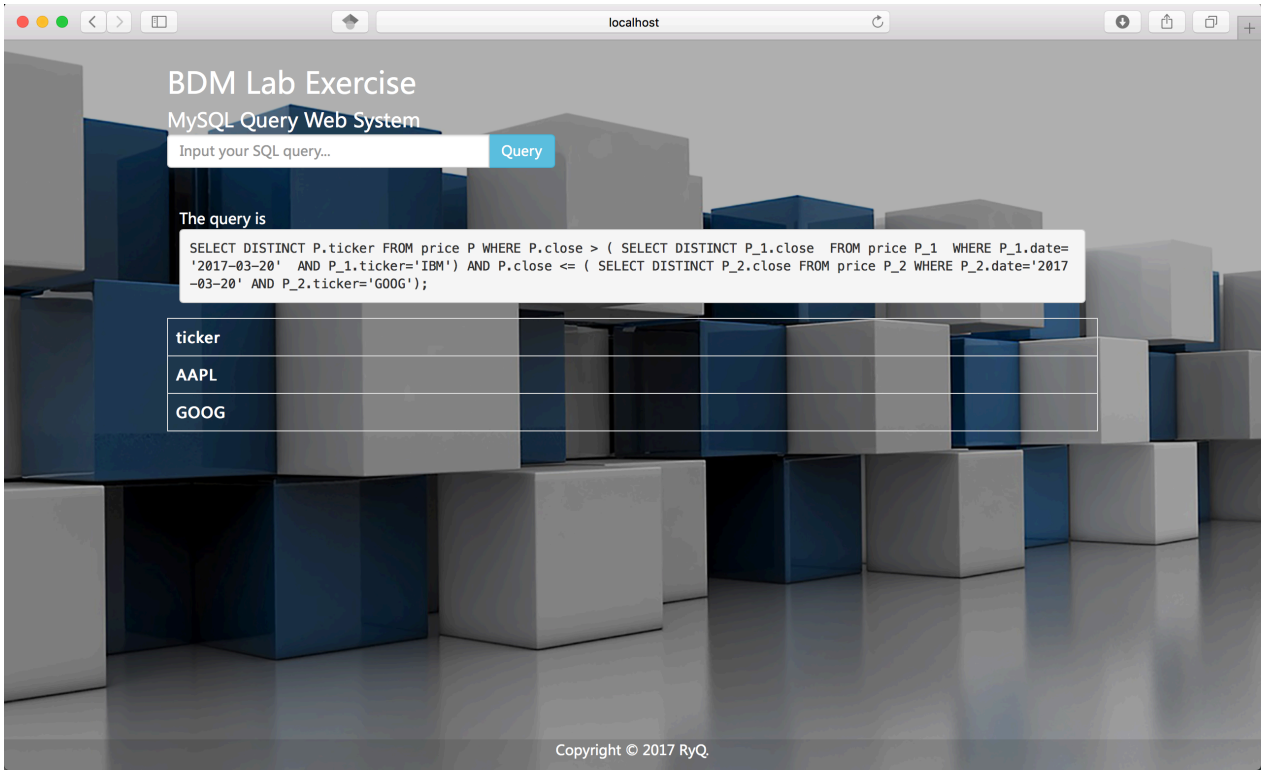




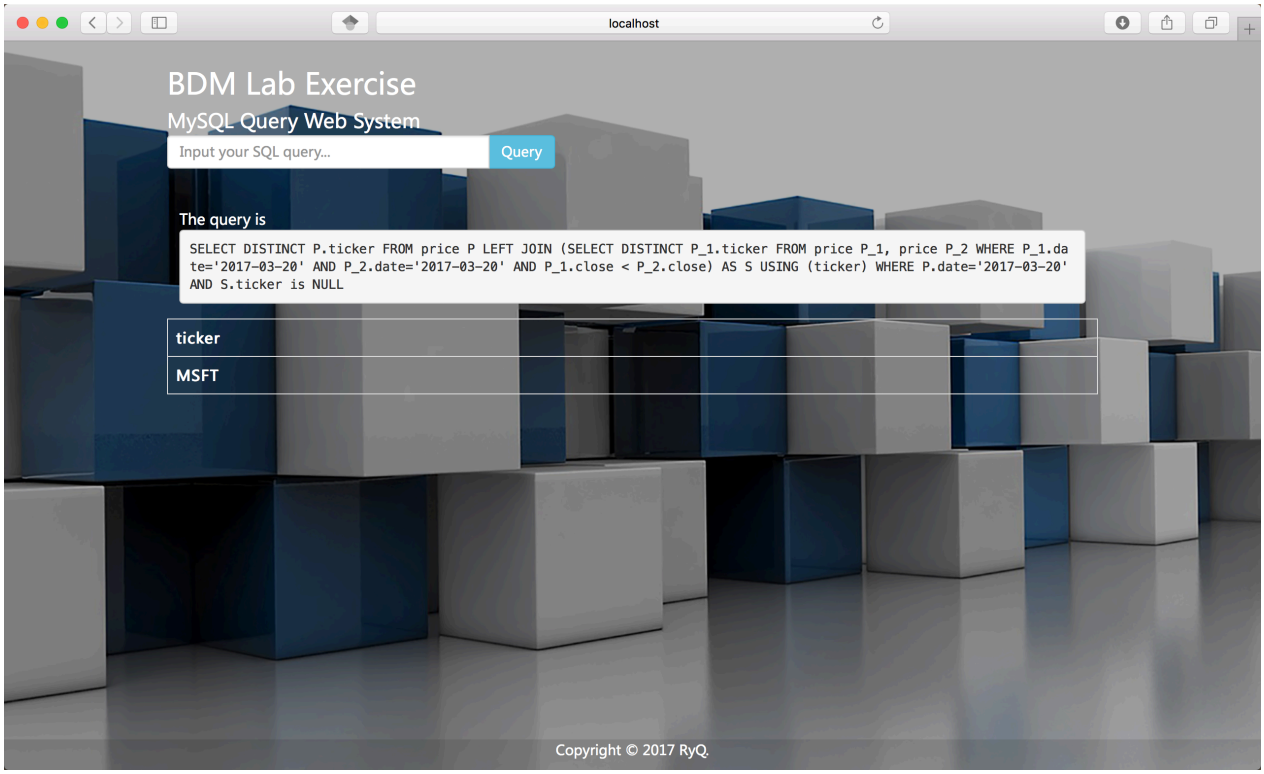
The result of Question 1: (Because of the height limit, the screen shot may not contain all the rows in result. The scroll bar inside can view the entire table.)



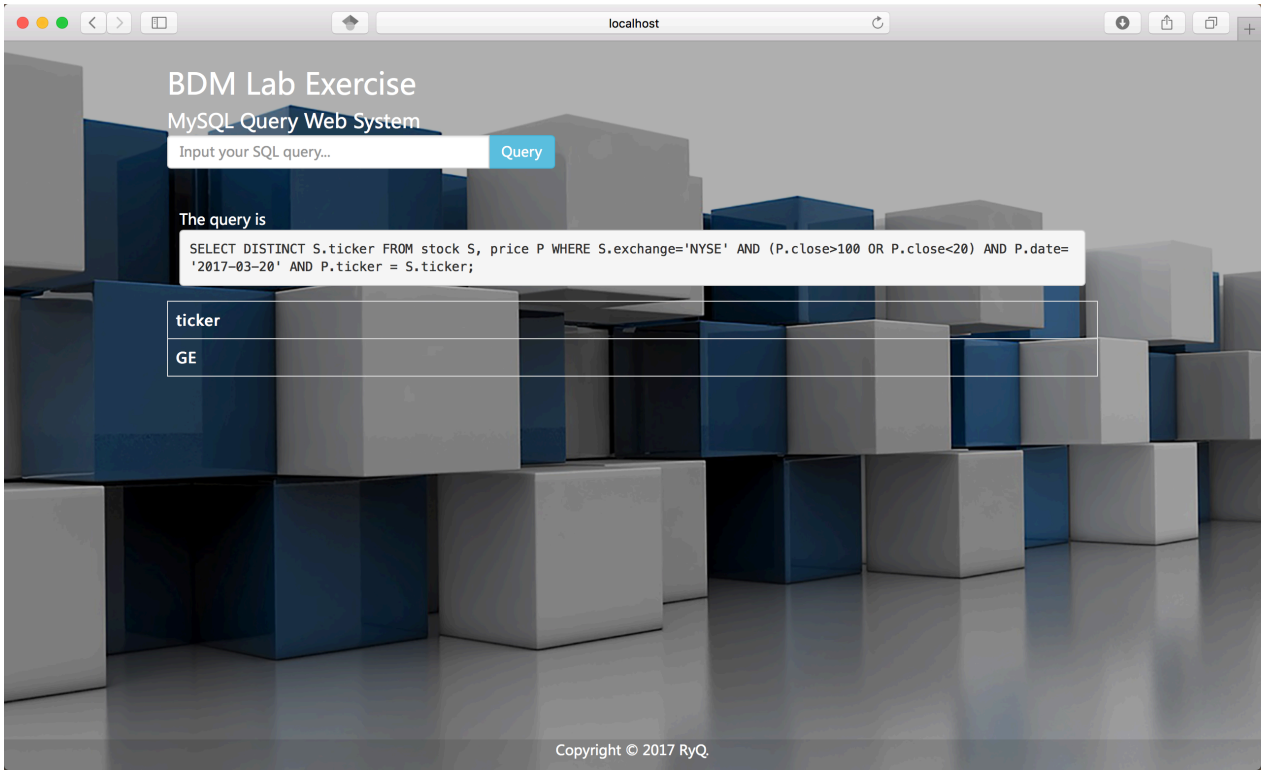
The query result of Question 2:



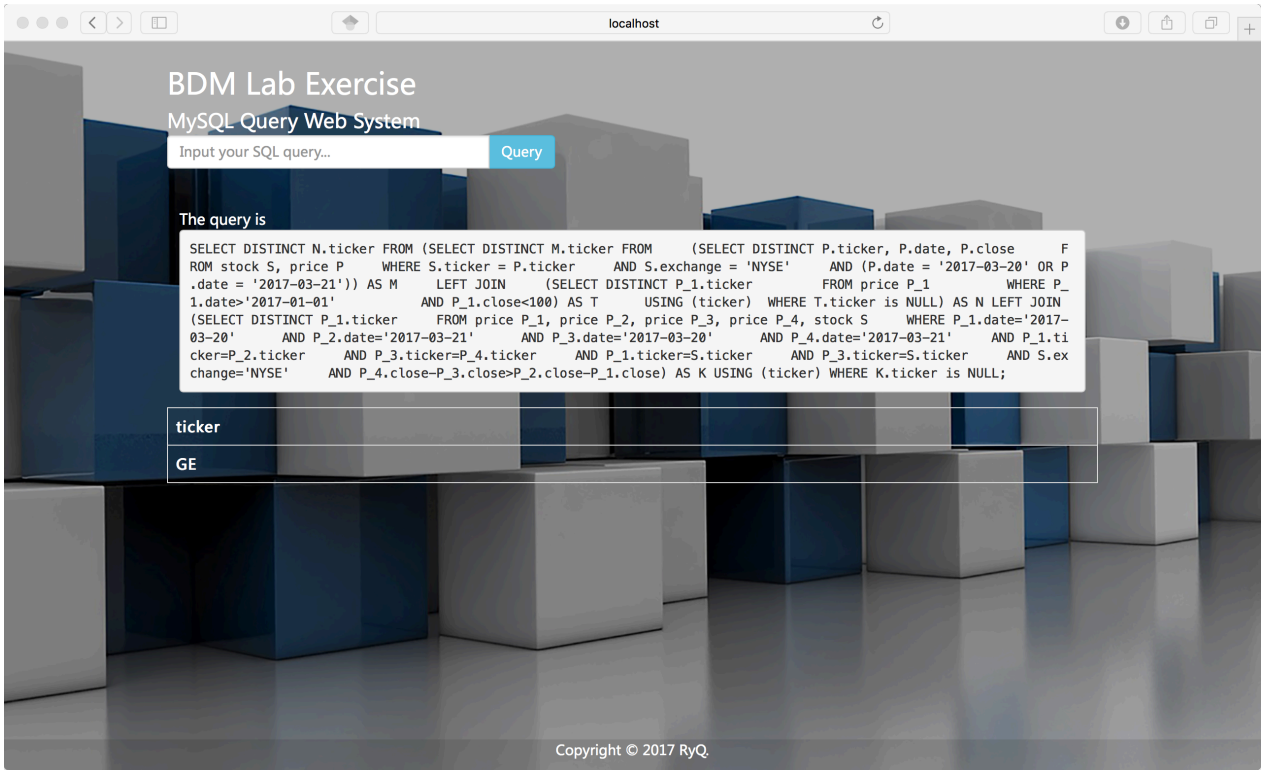
The query result of Question 3:



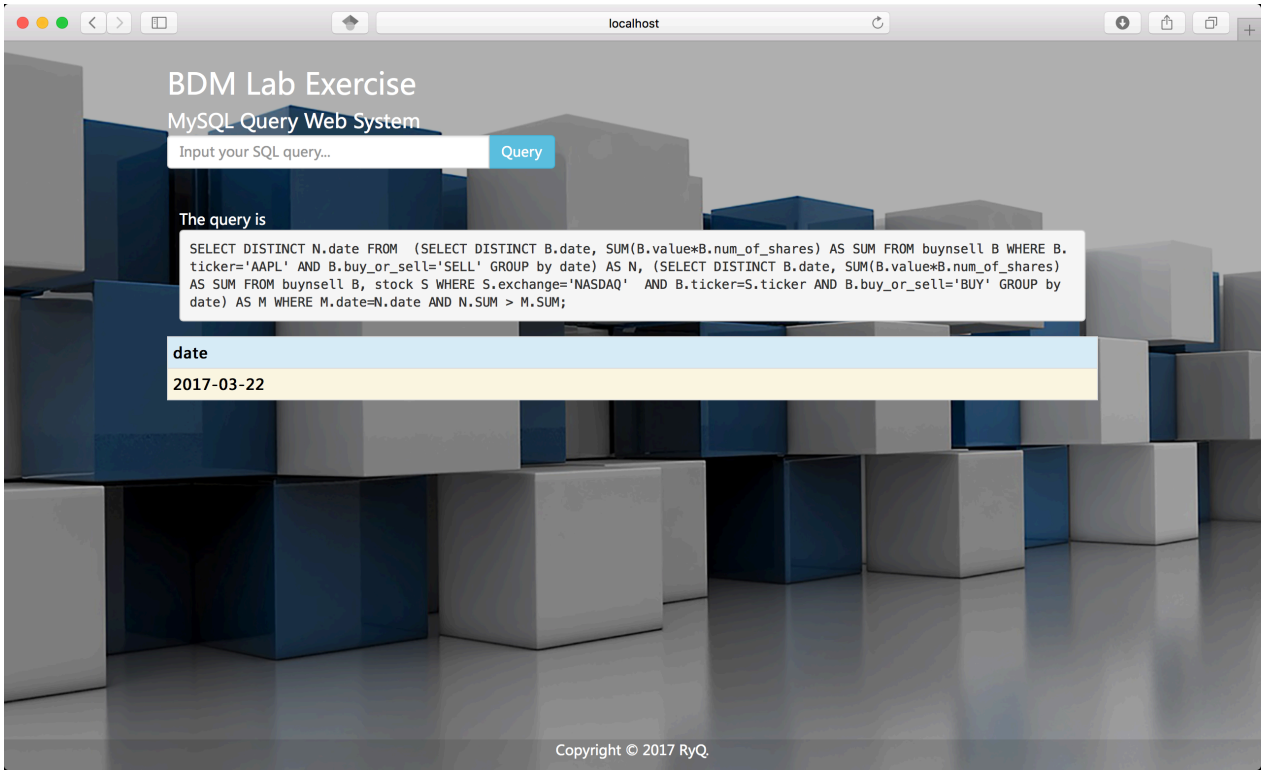
The query result of Question 4:



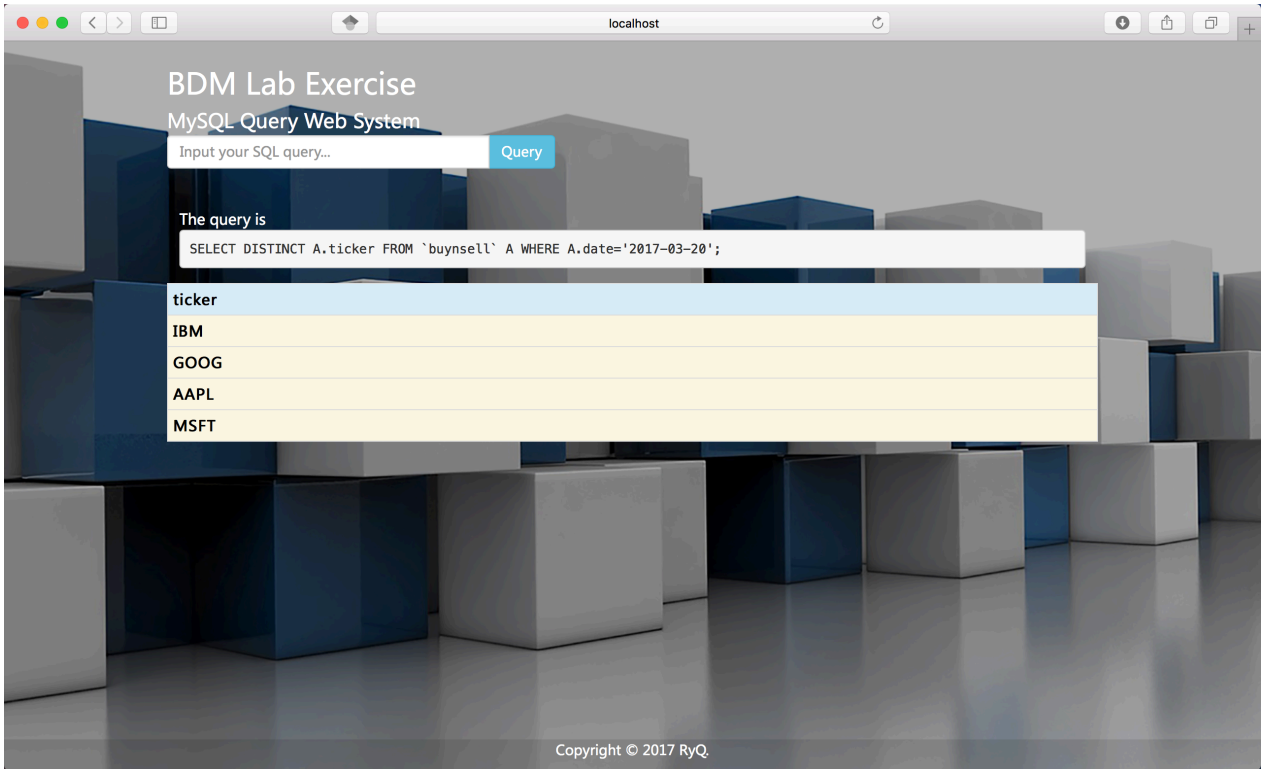
The query result of Question 5:



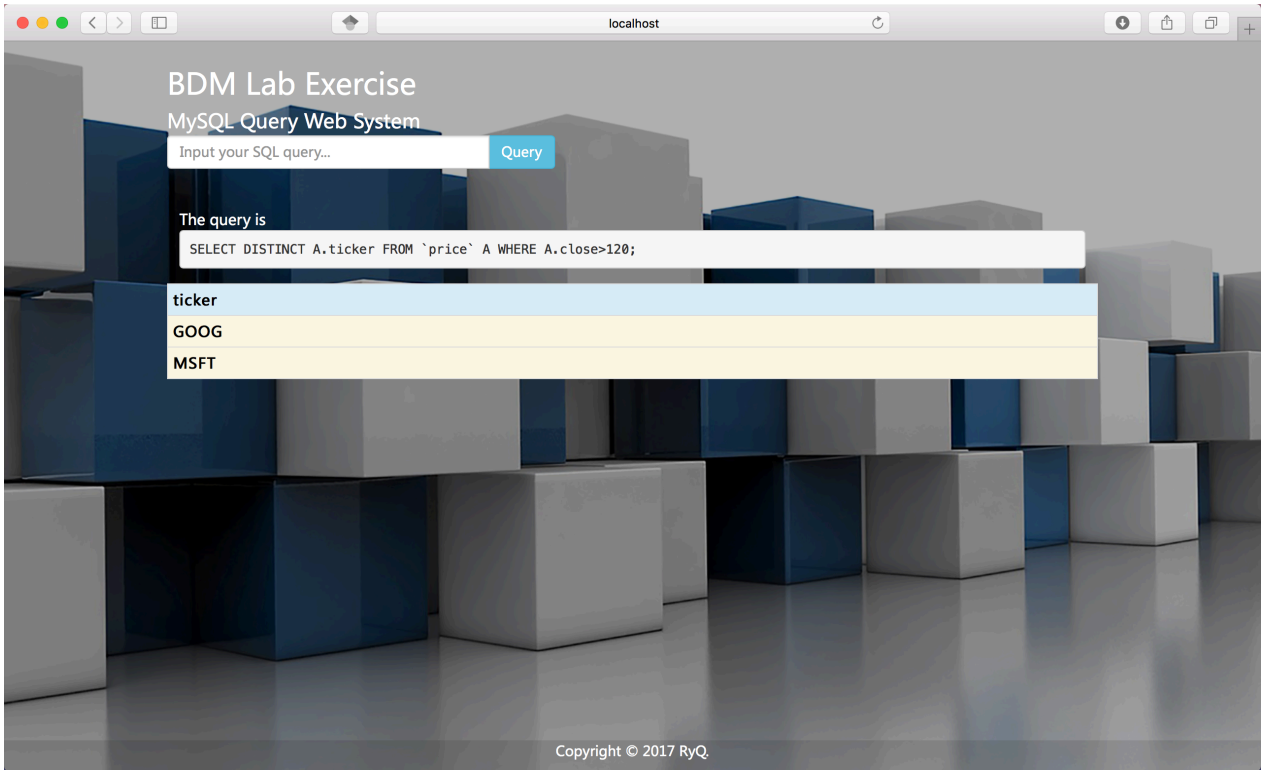
The query result of Question 6:



Additional query: SELECT DISTINCT A.ticker FROM `buynsell` A WHERE A.date='2017-03-20';



Additional query: SELECT DISTINCT A.ticker FROM `price` A WHERE A.close>120;



Part III

1. Explain how network programming and sockets come together.

Consider a network program as a black box in a network that has some functions, and in the two side of black box are inputs and outputs, which will be used by other programs. Socket is the way that the parameters transfer between programs.

During the process between user interface program and the fundamental server program, there are many different module and program connected by the socket.

2. Explain what is a “server”.

“Server” is that a program serves other programs on same or other computers. By the way, the computer that runs the “Server” program can be also called a server, because in the real world, one server computer often runs few specific server programs to keep the services has enough resources to load.

Server is a kind of fundamental program of most of computer-to-computer services, which need sockets to communication, interface to call the function and enough resources to store, compute and dispense. For instance, web server, application server, email server and so on.

3. Write an essay and give a detailed diagram explaining which network services are related to your “Part 2”.

Totally, 3 servers are used in Part II, web server, application server and database server.

In the system built in Part II,

- Web server, Apache, builds up a service that can run a website, including HTML parse, HTTP response, GET and POST method and delegation to CGI.
- Application server, the Python code, builds up a service that can deal with the delegation, the business logic part, from web server, including connection with MySQL, sending queries to MySQL, get the response from MySQL and rendering the response web pages.
- Database server, PhpMyAdmin and MySQL, which helps the database operation, builds up a service that can deal with database issues.

The entire process is:

1. Apache runs services to show the web page at Browser (HTML file).

2. Apache gets the input (query) and calls up CGI in Python.

   A “message”, the query, is sent to CGI (Python) from Apache (HTML).

3. Python gets the query, connects to MySQL and sends the query to MySQL. Python sleeps.

   A “message”, the query, is sent to Database (MySQL) from CGI (Python).

4. MySQL gets and executes the query and send the result back to Python. Connection closes and wakes up Python.

   A “message”, the result, is sent to CGI (Python) from Database (MySQL).

5. Python gets the result, outputs as HTML format and sends POST request to Apache. CGI closes.

   A “message”, the result page, is sent to Apache from CGI (Python).

6. Apache shows the result page at Browser.

During the process, the “message” refers to the parameter transfer via socket between programs.