Indian Institute of Technology Jodhpur
BDM

Assignment 3
BDM

Submitted by: Jojo Joseph (G23AI2100)

gitHub : https://github.com/IITJPGD/BDMAssignment2_g23ai2100

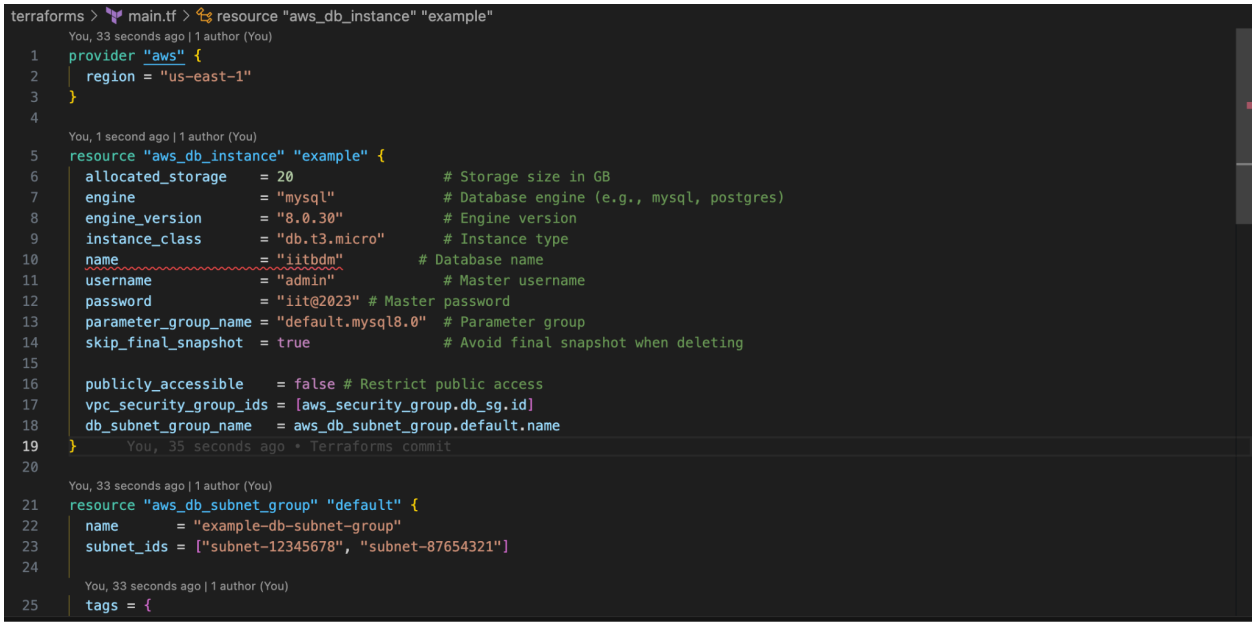
BDM Assignment

Q1

Submit a PDF with code listing, and screenshots showing outputs of insert(), delete(), and the queries. Screenshots should be uniquely distinguishable for each submission. Be careful of plagiarism from online sources/peers. Extend the Amazon RDS connector code in java to do the following in each of the respective functions.

Ans

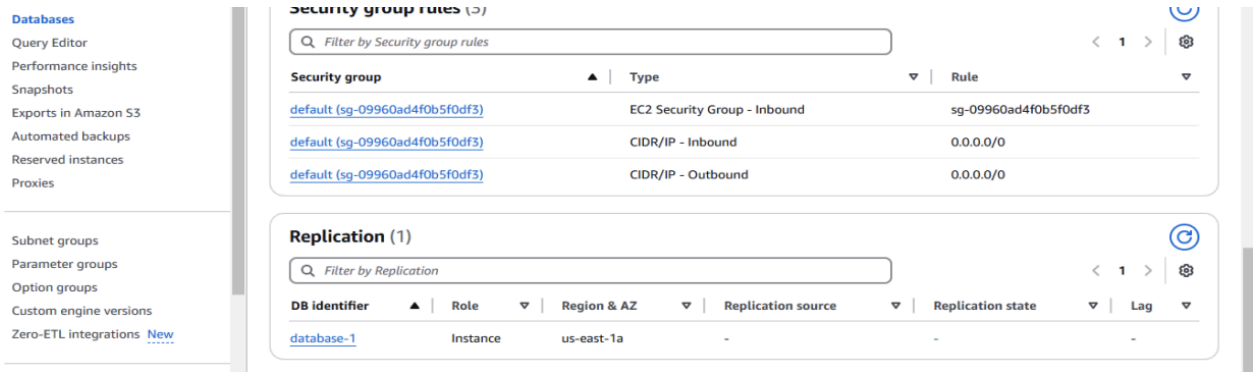
Database Setup for RDS (terraforms)



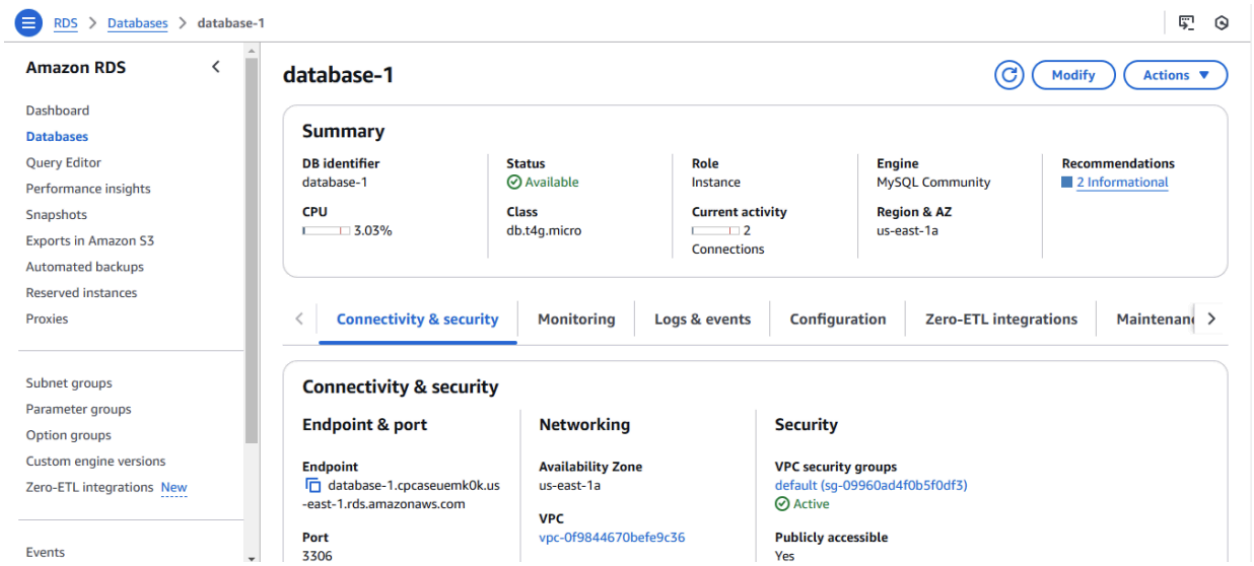

```
provider "aws" {
 region = "us-east-1"
}

resource "aws_db_instance" "example" {
 allocated_storage     = 20                    # Storage size in GB
 engine                = "mysql"               # Database engine (e.g., mysql, postgres)
 engine_version        = "8.0.30"              # Engine version
 instance_class        = "db.t3.micro"         # Instance type
 name                  = "database-1"           # Database name
 username              = "admin"               # Master username
 password              = "iit@2023" # Master password
 parameter_group_name  = "default.mysql8.0"   # Parameter group
 skip_final_snapshot   = true                  # Avoid final snapshot when deleting

 publicly_accessible     = true
 vpc_security_group_ids = [aws_security_group.db_sg.id]
 db_subnet_group_name    = aws_db_subnet_group.default.name
}

resource "aws_db_subnet_group" "default" {
 name        = "example-db-subnet-group"
 subnet_ids = ["subnet-12345678", "subnet-87654321"]

 tags = {
```

Submitted by: Jojo Joseph (G23AI2100)

```
    Name = "example-db-subnet-group"
  }
}

resource "aws_security_group" "db_sg" {
  name_prefix = "rds-db-sg"

  ingress {
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = ["10.0.0.0/16"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "db-sg"
  }
}
```



Submitted by: Jojo Joseph (G23AI2100)



Query to Analyze Weekly Stock Data

- **Retrieves the minimum, maximum, and average closing prices, along with the average volume for each company over a specific week.**
- **Results are ordered in descending order based on the average volume.**

Query to Filter Stocks by Closing Price

- **Identifies companies whose closing prices satisfy specified conditions relative to their weekly averages.**
- **Ensures inclusion of companies without tickers in the results.**

3. create() : creates the following table in the database: [10] *

Creates the table in the database. * Table name: company * Fields: * - id - integer, must be primary key * - name - variable character field up to size 50 * - ticker - character field always of size 10 * - annualRevenue - must hold up to 999,999,999,999.99 exactly * - numEmployees - integer Table name: stockprice * Fields: * - companyId - integer * - priceDate - date of stock price * - openPrice - opening price must hold up to 99999999.99 * - highPrice - high price must hold up to 99999999.99 * - lowPrice - low price must hold up to 99999999.99 * - closePrice - closing price must hold up to 99999999.99 * - volume - number of shares traded, integer * - primary key must be companyId and priceDate * - add an appropriate foreign key

DB Client

Submitted by: Jojo Joseph (G23AI2100)

```
7  select * from stockprice;
8
9  select * from company;
10
11 -- QUERY 1
12
13 select c.name, c.ticker, min(lowprice) As Lowest_Price, max(highprice) As Highest_price, avg(closePrice) AS
14 from company c
15 join stockprice s on c.Id  = s.companyId
16 where s.pricedate between "2022-08-22" and "2022-08-26"
17 group by c.name, c.ticker
18 order by Avg_volume DESC;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

companyId	priceDate	openPrice	highPrice	lowPrice	closePrice	volume
1	2022-08-22	169.69	169.86	167.14	167.57	69026800
1	2022-08-23	167.08	168.71	166.65	167.23	54147100
1	2022-08-24	167.32	168.11	166.25	167.53	53841500
1	2022-08-25	168.78	170.14	168.35	170.03	51218200
1	2022-08-26	170.57	171.05	163.56	163.62	78823500
1	2022-08-29	161.15	162.90	159.82	161.38	73314000
1	2022-08-30	162.13	162.56	157.72	158.91	77906200
4	2022-08-22	282.08	282.46	277.22	277.75	25061100
4	2022-08-23	276.44	278.86	275.40	276.44	17527400
4	2022-08-24	275.41	277.23	275.11	275.79	18137000

stockprice 25 | company 26

Result Grid | Form Editor | Field Types | Apply

Query 1

Limit to 1000 rows

```
7  select * from stockprice;
8
9  select * from company;
10
11 -- QUERY 1
12
13 select c.name, c.ticker, min(lowprice) As Lowest_Price, max(highprice) As Highest_price, avg(closePrice) AS
14 from company c
15 join stockprice s on c.Id  = s.companyId
16 where s.pricedate between "2022-08-22" and "2022-08-26"
17 group by c.name, c.ticker
18 order by Avg_volume DESC;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	ticker	annualRevenue	numEmployees
1	Apple	AAPL	387540000000.00	154000
2	GameStop	GME	611000000.00	12000
3	Handy Repair	NULL	2000000.00	50
4	Microsoft	MSFT	198270000000.00	221000
5	StartUp	NULL	50000.00	3
NULL	NULL	NULL	NULL	NULL

Result Grid | Form Editor

Code:

```
public class SQLonRDS {
    // Assignment 3 -
    public static void main(String[] args) {
```

Submitted by: Jojo Joseph (G23AI2100)

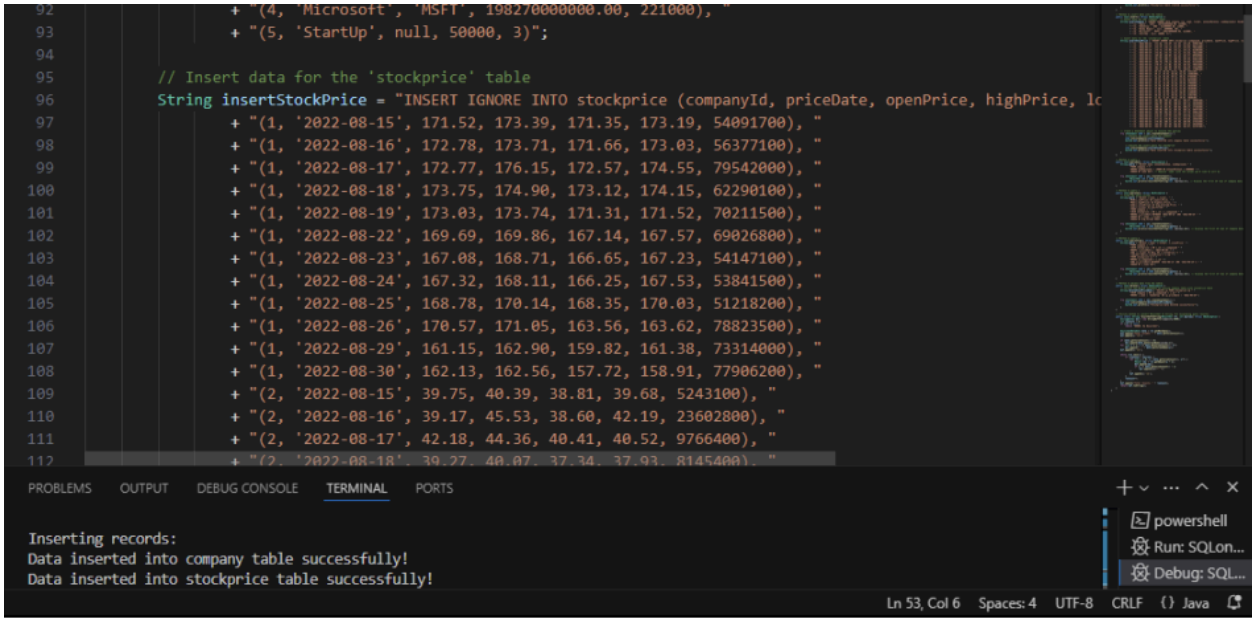
```
        System.out.println("Hello, World!");
    }
    //3. create() : creates the following table in the database: [10] *
    public void insert() throws SQLException {
        // Insert data for the 'company' table
        String insertCompany = "INSERT IGNORE INTO company (id, name,
        ticker, annualRevenue, numEmployees) VALUES "
        + "(1, 'Apple', 'AAPL', 387540000000.00, 154000), "
        + "(2, 'GameStop', 'GME', 611000000.00, 12000), "
        + "(3, 'Handy Repair', null, 2000000, 50), "
        + "(4, 'Microsoft', 'MSFT', 198270000000.00, 221000), "
        + "(5, 'StartUp', null, 50000, 3)";
        // Insert data for the 'stockprice' table
        String insertStockPrice = "INSERT IGNORE INTO stockprice
(companyId, priceDate, openPrice, highPrice, lowPrice, closePrice, volume)
VALUES "
+ "(1, '2022-08-15', 171.52, 173.39, 171.35, 173.19,
54091700), "
+ "(1, '2022-08-16', 172.78, 173.71, 171.66, 173.03,
56377100), "
+ "(1, '2022-08-17', 172.77, 176.15, 172.57, 174.55,
79542000), "
+ "(1, '2022-08-18', 173.75, 174.90, 173.12, 174.15,
62290100), "
+ "(1, '2022-08-19', 173.03, 173.74, 171.31, 171.52,
70211500), "
+ "(1,'2022-08-22', 169.69, 169.86, 167.14, 167.57,
69026800), "
+ "(1, '2022-08-23', 167.08, 168.71, 166.65, 167.23,
54147100), "
+ "(1, '2022-08-24', 167.32, 168.11, 166.25, 167.53,
53841500), "
+ "(1, '2022-08-25', 168.78, 170.14, 168.35, 170.03,
51218200), "
+ "(1, '2022-08-26', 170.57, 171.05, 163.56, 163.62,
78823500), "
+ "(1, '2022-08-29', 161.15, 162.90, 159.82, 161.38,
73314000), "
+ "(1, '2022-08-30', 162.13, 162.56, 157.72, 158.91,
77906200), "
+ "(2, '2022-08-15', 39.75, 40.39, 38.81, 39.68, 5243100),
" + "(2, '2022-08-16', 39.17, 45.53, 38.60, 42.19,
```

```
23602800), "
+ "(2, '2022-08-17', 42.18, 44.36, 40.41, 40.52, 9766400),
"
+ "(2, '2022-08-18', 39.27, 40.07, 37.34, 37.93, 8145400),
"
+ "(2, '2022-08-19', 35.18, 37.19, 34.67, 36.49, 9525600),
"
+ "(2, '2022-08-22', 34.31, 36.20, 34.20, 34.50, 5798600),
"
+ "(2, '2022-08-23', 34.70, 34.99, 33.45, 33.53, 4836300),
"
+ "(2, '2022-08-24', 34.00, 34.94, 32.44, 32.50, 5620300),
"
+ "(2, '2022-08-25', 32.84, 32.89, 31.50, 31.96, 4726300),
"
+ "(2, '2022-08-26', 31.50, 32.38, 30.63, 30.94, 4289500),
"
+ "(2, '2022-08-29', 30.48, 32.75, 30.38, 31.55, 4292700),
"
+ "(2, '2022-08-30', 31.62, 31.87, 29.42, 29.84, 5060200),
"
+ "(4, '2022-08-15', 291.00, 294.18, 290.11, 293.47,
18085700), " + "(4, '2022-08-16', 291.99, 294.04, 290.42, 292.71,
18102900), "
+ "(4, '2022-08-17', 289.74, 293.35, 289.47, 291.32,
18253400), " + "(4, '2022-08-18', 290.19, 291.91, 289.08, 290.17,
17186200), "
+ "(4, '2022-08-19', 288.90, 289.25, 285.56, 286.15,
20557200), "
+ "(4, '2022-08-22', 282.08, 282.46, 277.22, 277.75,
25061100), "
+ "(4, '2022-08-23', 276.44, 278.86, 275.40, 276.44,
17527400), "
+ "(4, '2022-08-24', 275.41, 277.23, 275.11, 275.79,
18137000), "
+ "(4, '2022-08-25', 277.33, 279.02, 274.52, 278.85,
16583400), "
+ "(4, '2022-08-26', 279.08, 280.34, 267.98, 268.09,
27532500), "
+ "(4, '2022-08-29', 265.85, 267.40, 263.85, 265.23,
20338500), "
+ "(4, '2022-08-30', 266.67, 267.05, 260.66, 262.97,
```

Submitted by: Jojo Joseph (G23AI2100)

```
22767100)";

// Create a Statement object to execute the queries
try (Statement stmt = con.createStatement()) {
    // Execute the insert query for company
    stmt.executeUpdate(insertCompany);
    System.out.println("Data inserted into company table
    successfully!");
    // Execute the insert query for stockprice
    stmt.executeUpdate(insertStockPrice);
System.out.println("Data inserted into stockprice table
successfully!");
}
}

}
```

Output



Q2. delete() [5] Delete all stock price records where the date is before 2022-08-20 or the company is GameStop

Code

```

```

Submitted by: Jojo Joseph (G23AI2100)

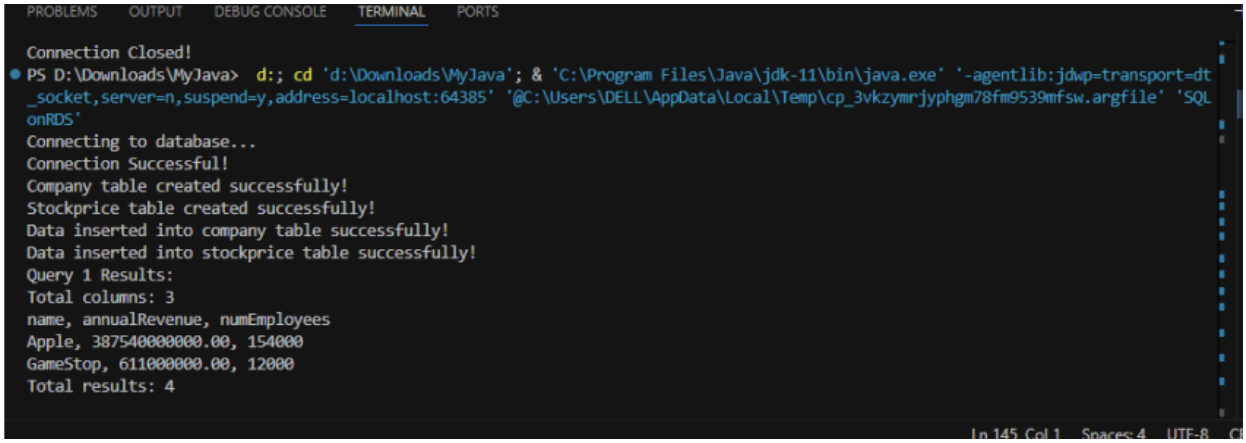
```
// Method to delete data from the tables - Correct the DELETE statement by joining
company table with -- stockprice --  table
public void delete() throws SQLException {
        String deleteStockPriceData = "DELETE sp FROM stockprice sp " + "JOIN company c
ON sp.companyId = c.id " + "WHERE c.name = 'GameStop' OR sp.priceDate < '2022-08-20'";
        try (Statement stmt = con.createStatement()) {
            // Execute the delete query for stockprice
            stmt.executeUpdate(deleteStockPriceData);
            System.out.println("Stockprice data deleted successfully!");
        }
    }

}
```

Q3 //TODO for returning ResultSet [5] Query returns company info (name, revenue, employees) that have more than 10000 employees or annual revenue less that 1 million dollars. Order by company name ascending.

QUERY

```
    public void queryOne() throws SQLException {
        String query = "SELECT name, annualRevenue, numEmployees "
                + "FROM company "
                + "WHERE numEmployees > 10000 OR annualRevenue < 1000000 "
                + "ORDER BY name ASC";
        // Get the stock price
        try (Statement stmt = con.createStatement(); ResultSet rst =
stmt.executeQuery(query)) {
            System.out.println(resultSetToString(rst, 2));
        }
    }
```

OUTPUT:



Q4. queryTwo() ://TODO for returning ResultSet [5] Query returns the company name and ticker and calculates the lowest price, highest price, average closing price, and average volume in the week of August 22nd to 26th inclusive. Order by average volume descending

Query

```
public void querytwo() throws SQLException {
        // Get the stock price and return
        String query = "SELECT c.name, c.ticker, "
                + "MIN(s.lowPrice) AS Lowest_Price, "
                + "MAX(s.highPrice) AS Highest_Price, "
                + "AVG(s.closePrice) AS Avg_Closing_Price, "
                + "AVG(s.volume) AS Avg_Volume "
                + "FROM company c "
                + "JOIN stockprice s ON c.id = s.companyId "
                + "WHERE s.priceDate BETWEEN '2022-08-22' AND '2022-08-26' "
                + "GROUP BY c.name, c.ticker "
                + "ORDER BY Avg_Volume DESC";
        try (Statement stmt = con.createStatement(); ResultSet rst =
stmt.executeQuery(query)) {
            System.out.println(resultSetToString(rst, 10)); // Display the
            first 10 rows of company data
        }
    }
```

Output:

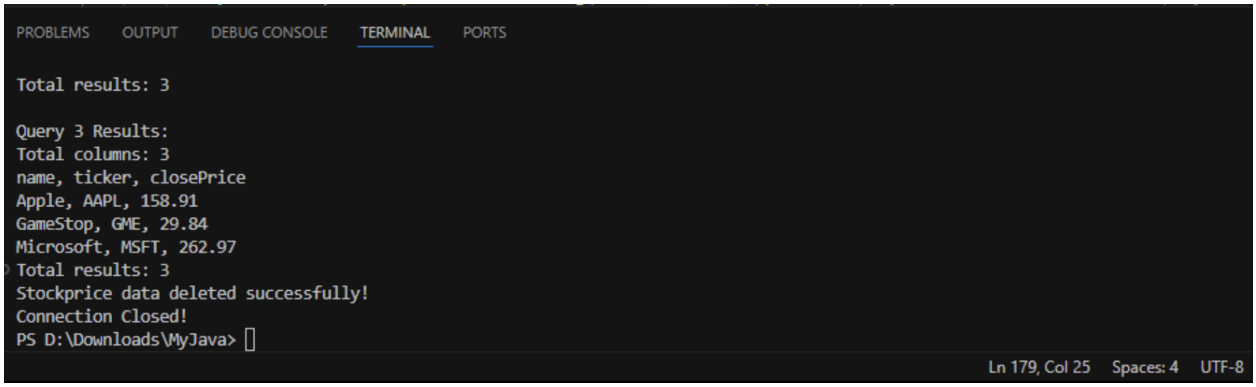
Q5 :

//TODO for returning ResultSet [5] Query returns a list of all companies that displays their name, ticker, and closing stock price on August 30, 2022 (if exists). Only show companies where their closing stock price on August 30, 2022 is no more than 10% below the closing average for the week of August 15th to 19th inclusive. That is, if closing price is currently 100, the average closing price must be <= 110. Companies without a stock ticker should always be shown in the list. Order by company name ascending.

Code:

```
    public void queryThree() throws SQLException {


        String query = "SELECT c.name, c.ticker, s.closePrice "
                + "FROM company c "
                + "JOIN stockprice s ON c.id = s.companyId "
                + "WHERE s.priceDate = '2022-08-30' "
                + "AND (c.ticker IS NULL OR s.closePrice <= "
                + "(SELECT AVG(s1.closePrice) * 1.10 "
                + "FROM stockprice s1 "
                + "WHERE s1.companyId = c.id " + "AND s1.priceDate BETWEEN '2022-08-15'
AND '2022-08-19'))" + "ORDER BY c.name ASC";
        //run
        try (Statement stmt = con.createStatement(); ResultSet rst =
stmt.executeQuery(query)) {
            System.out.println(resultSetToString(rst, 10));
        }
    }
```

Output:



Q6 converts a ResultSet obtained front he queries to String (Given) and 10. resultSetMetaDataToString() : converts resultSetMetaData to String or the String of the metadata (Schema) (Given)

Code :

```
public static String resultSetToString(ResultSet rst, int maxrows) throws
            SQLException {
        StringBuffer buf = new StringBuffer(5000);
        int rowCount = 0;
        if (rst == null) {
            return "ERROR: No ResultSet";
        }
        ResultSetMetaData meta = rst.getMetaData();
        buf.append("Total columns: " + meta.getColumnCount());
        buf.append('\n');
        if (meta.getColumnCount() > 0) {
            buf.append(meta.getColumnName(1));
        }
        for (int j = 2; j <= meta.getColumnCount(); j++) {
            buf.append(", " + meta.getColumnName(j));
        }
        buf.append('\n');
        while (rst.next()) {
            if (rowCount < maxrows) {
                for (int j = 0; j < meta.getColumnCount(); j++) {
                    Object obj = rst.getObject(j + 1);
                    buf.append(obj);
                    if (j != meta.getColumnCount() - 1) {
                        buf.append(", ");
                    }
                }
```

```
                buf.append('\n');
            }
            rowCount++;
        }
        buf.append("Total results: " + rowCount);
        return buf.toString();
    }
```

Submitted by: Jojo Joseph (G23AI2100)