



# BIG DATA MANAGEMENT

## Assignment - 3

Name : Shubham Raj  
Roll No : G23AI2028

## Boiler Plate code for Java Setup :

Starter Code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class SQLonRDS {
    private Connection con;
    private String url = "iitjdb.cpmqoqqutp0.us-east-1.rds.amazonaws.com:3306";
    private String uid = "admin";
    private String pw = "iitj@12345";
    public static void main(String[] args) {
        SQLonRDS q = new SQLonRDS();
        try {
            q.connect();
            //q.create(); // TODO AS PER THE QUESTION
            //q.drop(); // TODO AS PER THE QUESTION USING ResultSet type as return
            //q.insert(); // TODO AS PER THE QUESTION
            //q.queryOne(); // TODO AS PER THE QUESTION
            //q.queryTwo(); // TODO AS PER THE QUESTION
            //q.queryThree(); // TODO AS PER THE QUESTION
            q.close();
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public void connect() throws SQLException, ClassNotFoundException {
        // Load MySQL JDBC Driver
        Class.forName("com.mysql.cj.jdbc.Driver");
        String jdbcUrl = "jdbc:mysql://" + url + "/mydb?user=" + uid + "&password=" + pw;
        System.out.println("Connecting to database");
        con = DriverManager.getConnection(jdbcUrl);
        System.out.println("Connection Successful");
    }

    public void close() throws SQLException {
        if (con != null) {
            con.close();
        }
    }

    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())
```

```

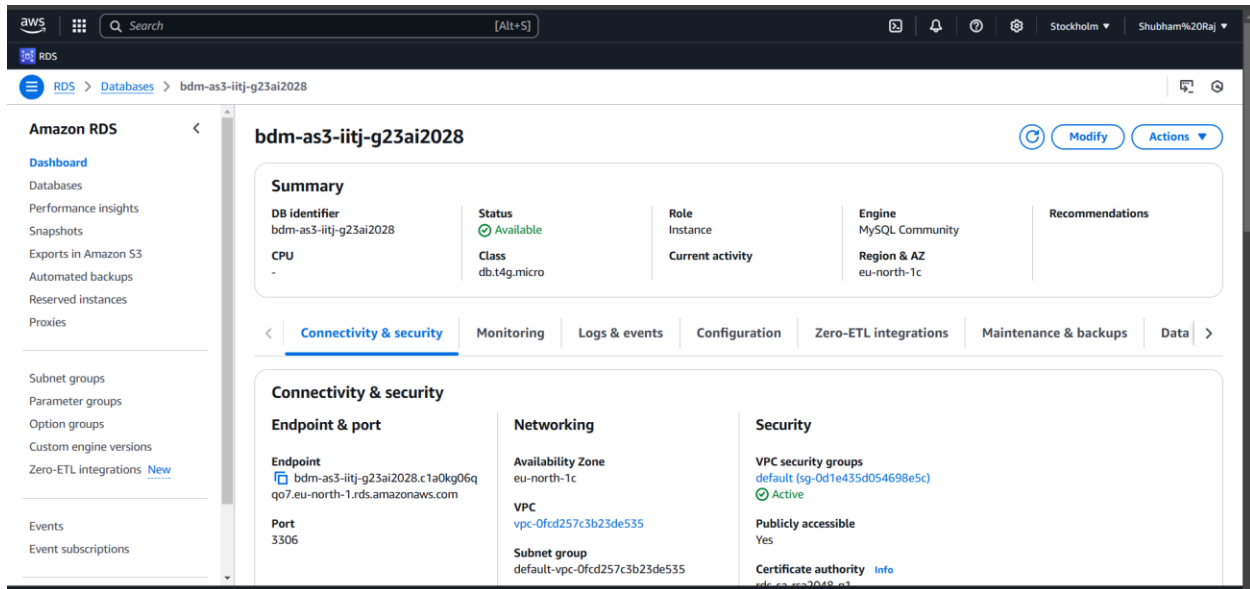
{
if (rowCount < maxrows)
{
for (int j = 0; j < meta.getColumnCount(); j++)
{
Object obj = rst.getObject(j + 1);
buf.append(obj);
if (j != meta.getColumnCount() - 1)
buf.append(", ");
}
buf.append("\n");
} rowCount++;
}
buf.append("Total results: " + rowCount);
return buf.toString();
}

public static String resultSetMetaDataToString(ResultSetMetaData meta) throws SQLException
{
StringBuffer buf = new StringBuffer(5000);
buf.append(meta.getColumnLabel(1)+" (" +meta.getColumnLabel(1)+",
"+meta.getColumnType(1)+"-"+meta.getColumnTypeName(1)+", "+meta.getColumnDisplaySize(1)+",
"+meta.getPrecision(1)+", "+meta.getScale(1)+")");
for (int j = 2; j <= meta.getColumnCount(); j++)
buf.append(", "+meta.getColumnLabel(j)+" (" +meta.getColumnLabel(j)+",
"+meta.getColumnType(j)+"-"+meta.getColumnTypeName(j)+", "+meta.getColumnDisplaySize(j)+",
"+meta.getPrecision(j)+", "+meta.getScale(j)+")");
return buf.toString();
}
}
}

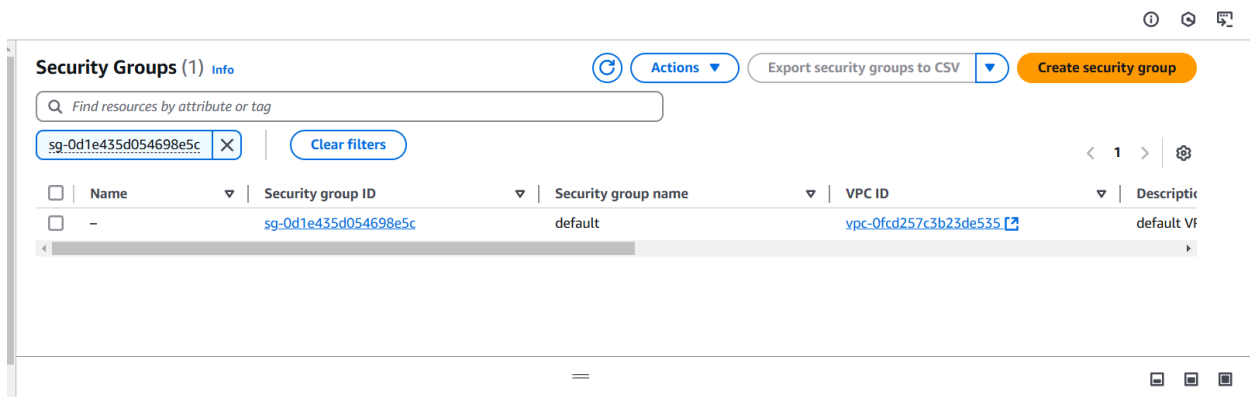
```

Github : <https://github.com/shubham14p3/iitj-bdm-as3-java-aws-rds>

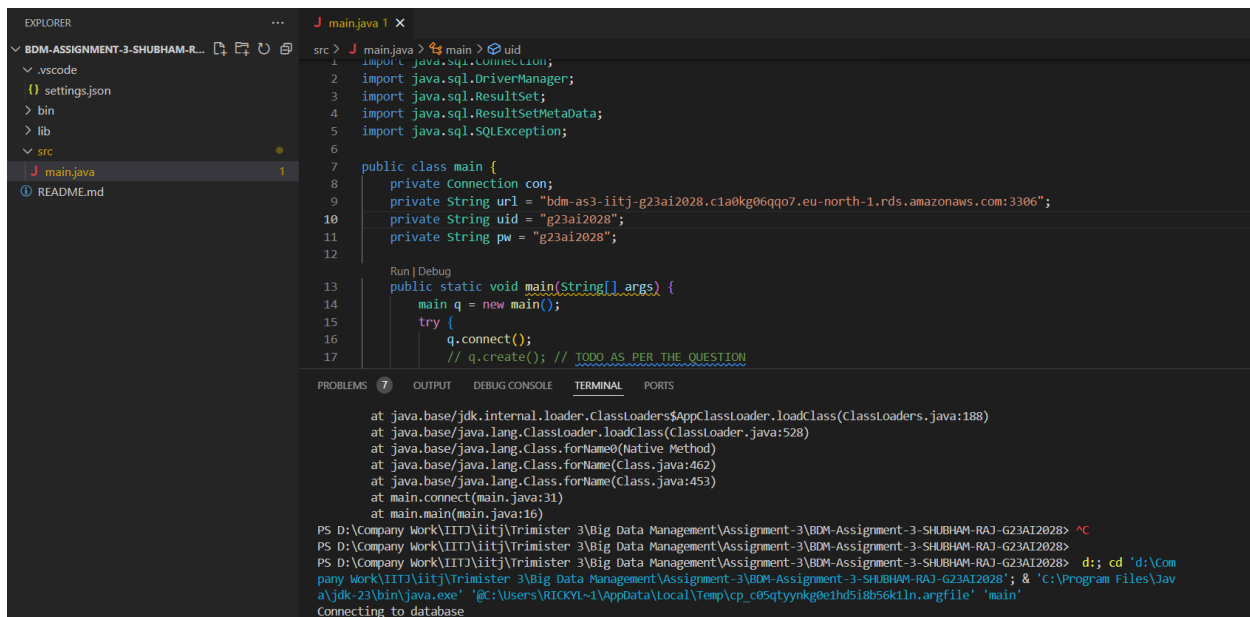
## Aws RDS: DB creation



To get access we need to set rule either by allowing our own IP or setting to 0.0.0.0/0 so it can be access worldwide



Screen after creating and modifying the details with mine, such as db name, endpoint of Db, username and password

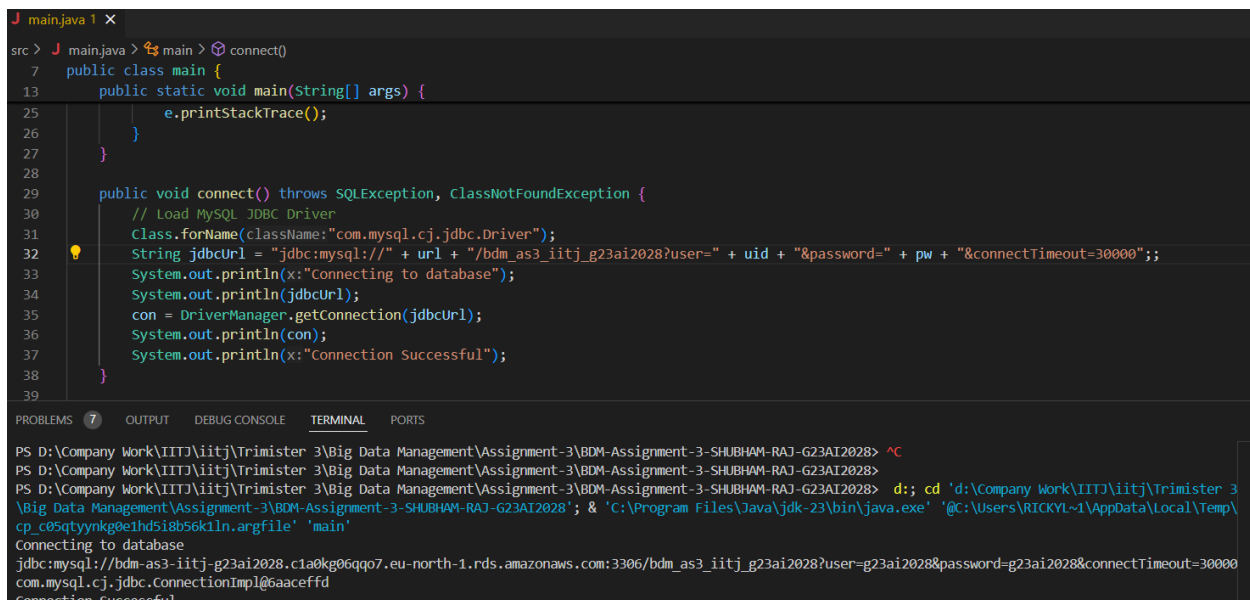


```
EXPLORER
BDM-ASSIGNMENT-3-SHUBHAM-RAJ
  .vscode
  settings.json
  bin
  lib
  src
    J main.java
    README.md

src > J main.java > main > uid
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.ResultSetMetaData;
5 import java.sql.SQLException;
6
7 public class main {
8     private Connection con;
9     private String url = "bdm-as3-iitj-g23ai2028.c1a0kg06qqo7.eu-north-1.rds.amazonaws.com:3306";
10    private String uid = "g23ai2028";
11    private String pw = "g23ai2028";
12
13    Run | Debug
14    public static void main(String[] args) {
15        main m = new main();
16        try {
17            m.connect();
18            // m.create(); // TODO AS PER THE QUESTION
19        } catch (Exception e) {
20            e.printStackTrace();
21        }
22    }
23
24    public void connect() throws SQLException, ClassNotFoundException {
25        // Load MySQL JDBC Driver
26        Class.forName(className="com.mysql.cj.jdbc.Driver");
27        String jdbcUrl = "jdbc:mysql://" + url + "/" + uid + "?user=" + uid + "&password=" + pw + "&connectTimeout=30000";
28        System.out.println(x:"Connecting to database");
29        con = DriverManager.getConnection(jdbcUrl);
30        System.out.println(con);
31        System.out.println(x:"Connection Successful");
32    }
33
34    }
35
36    }
37
38    }
39
40    }
41
42    }
43
44    }
45
46    }
47
48    }
49
50    }
51
52    }
53
54    }
55
56    }
57
58    }
59
60    }
61
62    }
63
64    }
65
66    }
67
68    }
69
70    }
71
72    }
73
74    }
75
76    }
77
78    }
79
80    }
81
82    }
83
84    }
85
86    }
87
88    }
89
90    }
91
92    }
93
94    }
95
96    }
97
98    }
99
100   }
```

```
at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java:188)
at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:528)
at java.base/java.lang.Class.forName0(Native Method)
at java.base/java.lang.Class.forName(Class.java:462)
at java.base/java.lang.Class.forName(Class.java:453)
at main.connect(main.java:31)
at main.main(main.java:16)
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028> ^C
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028>
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028> d;; cd 'd:\Com
pany Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028'; & 'c:\Program Files\jav
a\jdk-23\bin\java.exe' '@C:\Users\RICKYL~1\AppData\Local\Temp\cp_c05qtyyng0e1hd5i8b56k1ln.argfile' 'main'
Connecting to database
```

Connection Successful :



```
J main.java 1 X
src > J main.java > main > connect()
7 public class main {
13    public static void main(String[] args) {
25        e.printStackTrace();
26    }
27 }
28
29 public void connect() throws SQLException, ClassNotFoundException {
30     // Load MySQL JDBC Driver
31     Class.forName(className="com.mysql.cj.jdbc.Driver");
32     String jdbcUrl = "jdbc:mysql://" + url + "/" + uid + "?user=" + uid + "&password=" + pw + "&connectTimeout=30000";
33     System.out.println(x:"Connecting to database");
34     System.out.println(jdbcUrl);
35     con = DriverManager.getConnection(jdbcUrl);
36     System.out.println(con);
37     System.out.println(x:"Connection Successful");
38 }
39 }
```

```
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028> ^C
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028>
PS D:\Company Work\IITJ\iitj\Trimister 3\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028> d;; cd 'd:\Company Work\IITJ\iitj\Trimister 3
\Big Data Management\Assignment-3\BDM-Assignment-3-SHUBHAM-RAJ-G23AT2028'; & 'c:\Program Files\Java\jdk-23\bin\java.exe' '@C:\Users\RICKYL~1\AppData\Local\Temp\
cp_c05qtyyng0e1hd5i8b56k1ln.argfile' 'main'
Connecting to database
jdbc:mysql://bdm-as3-iitj-g23ai2028.c1a0kg06qqo7.eu-north-1.rds.amazonaws.com:3306/bdm_as3_iitj_g23ai2028?user=g23ai2028&password=g23ai2028&connectTimeout=30000
com.mysql.cj.jdbc.ConnectionImpl@6aaceffd
Connection Successful
```

## Task:

1. connect() : to connect to the RDS database

```
// Connect to the database
public void connect() throws SQLException, ClassNotFoundException {
    Class.forName("com.mysql.cj.jdbc.Driver");
    String jdbcUrl = "jdbc:mysql://" + url + "/bdm_as3_iitj_g23ai2028?user="
+ uid + "&password=" + pw + "&connectTimeout=30000";
    System.out.println("\n===== Connecting to Database =====");
    System.out.println("JDBC URL: " + jdbcUrl);
    con = DriverManager.getConnection(jdbcUrl);
    System.out.println("Connection Successful: " + con + "\n");
}
```

## Output:

```
===== Connecting to Database =====
JDBC URL: jdbc:mysql://bdm-as3-iitj-g23ai2028.c1a0kg06qqo7.eu-north-1.rds.amazonaws.com:3306/bdm_as3_iitj_g23ai2028?user=g23ai2028&password=g23ai2028&connectTim
eout=30000
Connection Successful: com.mysql.cj.jdbc.ConnectionImpl@5b94b04d
```

2. drop() : to drop the table from the database

```
public void drop() throws SQLException {
    System.out.println("\n===== Dropping Tables =====");
    String dropStockPrice = "DROP TABLE IF EXISTS stockprice";
    String dropCompany = "DROP TABLE IF EXISTS company";

    try (Statement stmt = con.createStatement()) {
        stmt.executeUpdate(dropStockPrice);
        stmt.executeUpdate(dropCompany);
        System.out.println("Tables dropped successfully.");
    }
}
```

## Output

```
===== Dropping Tables =====
Tables dropped successfully.
```

3. create() : creates the following table in the database:

- \* 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 999999999.99

- \* - highPrice - high price must hold up to 999999999.99

- \* - lowPrice - low price must hold up to 999999999.99

- \* - closePrice - closing price must hold up to 999999999.99

- \* - volume - number of shares traded, integer

- \* - primary key must be companyId and priceDate

- \* - add an appropriate foreign key

```
public void create() throws SQLException {
    System.out.println("\n==== Creating Tables =====");
    String createCompany = ""
        CREATE TABLE company (
            id INT PRIMARY KEY,
            name VARCHAR(50) NOT NULL,
            ticker CHAR(10),
            annualRevenue DECIMAL(15, 2),
            numEmployees INT
        )
    "";

    String createStockPrice = ""
        CREATE TABLE stockprice (
            companyId INT,
            priceDate DATE,
            openPrice DECIMAL(10, 2),
            highPrice DECIMAL(10, 2),
            lowPrice DECIMAL(10, 2),
            closePrice DECIMAL(10, 2),
```

```

        volume INT,
        PRIMARY KEY (companyId, priceDate),
        FOREIGN KEY (companyId) REFERENCES company(id)
    )
""";

try (Statement stmt = con.createStatement()) {
    stmt.executeUpdate(createCompany);
    stmt.executeUpdate(createStockPrice);
    System.out.println("Tables created successfully.");
}
}

```

## Output

```

===== Creating Tables =====
Tables created successfully.

```

4. insert() : Inserts test records in the database:

\* Data for company table:

- 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

\*\*

Data for stockprice table:

- 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, 22767100

```
public void insert() throws SQLException {
    System.out.println("\n===== Inserting Records =====");
    String insertCompany = ""
        INSERT INTO company (id, name, ticker, annualRevenue, numEmployees)
VALUES
        (1, 'Apple', 'AAPL', 387540000000.00, 154000),
        (2, 'GameStop', 'GME', 611000000.00, 12000),
        (3, 'Handy Repair', NULL, 2000000.00, 50),
        (4, 'Microsoft', 'MSFT', 198270000000.00, 221000),
        (5, 'StartUp', NULL, 50000.00, 3)
    "";

    String insertStockPrice = ""
        INSERT 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, 22767100)
""";

try (Statement stmt = con.createStatement()) {
    stmt.executeUpdate(insertCompany);
    stmt.executeUpdate(insertStockPrice);
    System.out.println("Test records inserted successfully.");
}

printTable("company");
printTable("stockprice");
}

```

## Output

==== Contents of Table: company ====

Total columns: 5

id	name	ticker	annualRevenue	numEmployees
1	Apple	AAPL	38754000000.00	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

Total results: 5

==== Contents of Table: stockprice ====

Total columns: 7

companyId	priceDate	openPrice	highPrice	lowPrice	closePrice	volume
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
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	22767100

Total results: 36

### 5. delete() [5]

Delete all stock price records where the date is before 2022-08-20 or the company is GameStop

```
public void delete() throws SQLException {
    System.out.println("\n==== Deleting Records =====");
    String deleteQuery = ""
        DELETE sp
        FROM stockprice sp
        JOIN company c ON sp.companyId = c.id
        WHERE sp.priceDate < '2022-08-20' OR c.name = 'GameStop'
```

```

        """;

        try (Statement stmt = con.createStatement()) {
            int rowsDeleted = stmt.executeUpdate(deleteQuery);
            System.out.println(rowsDeleted + " records deleted from stockprice
table.");
        }

        printTable("stockprice");
    }

```

## Output

```

===== Deleting Records =====
22 records deleted from stockprice table.

===== Contents of Table: stockprice =====
Total columns: 7
-----
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
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        22767100
-----
Total results: 14

```

6. queryOne(): //TODO for returning ResultSet [5]

Query returns company info (name, revenue, employees) that have more than 10000 employees or annual revenue less than 1 million dollars. Order by company name ascending.

```

public void queryOne() throws SQLException {
    System.out.println("\n===== Query One =====");
    String query = """
        SELECT name, annualRevenue, numEmployees
        FROM company
        WHERE numEmployees > 10000 OR annualRevenue < 1000000
        ORDER BY name ASC
    """;
}

```

```

    try (Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query)) {
        System.out.println(resultSetToString(rs));
    }
}

```

Output

```

===== Query One =====
Total columns: 3
-----
name            annualRevenue  numEmployees
-----
Apple           387540000000.00 154000
GameStop        611000000.00    12000
Microsoft       198270000000.00 221000
Startup         50000.00        3
-----
Total results: 4

```

7. 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.

```

public void queryTwo() throws SQLException {
    System.out.println("\n===== Query Two =====");
    String query = ""
        SELECT companyId,
            MIN(lowPrice) AS lowestPrice,
            MAX(highPrice) AS highestPrice,
            AVG(closePrice) AS avgClosePrice,
            AVG(volume) AS avgVolume
        FROM stockprice
        WHERE priceDate BETWEEN '2022-08-22' AND '2022-08-26'
        GROUP BY companyId
        ORDER BY avgVolume DESC
    "";

    try (Statement stmt = con.createStatement();

```

```

        ResultSet rs = stmt.executeQuery(query)) {
            System.out.println(resultSetToString(rs));
        }
    }
}

```

## Output

===== Query Two =====

Total columns: 5

companyId	lowestPrice	highestPrice	avgClosePrice	avgVolume
1	163.56	171.05	167.196000	61411420.0000
4	267.98	282.46	275.384000	20968280.0000

Total results: 2

Total columns: 5

companyId	lowestPrice	highestPrice	avgClosePrice	avgVolume
1	163.56	171.05	167.196000	61411420.0000
4	267.98	282.46	275.384000	20968280.0000

Total results: 2

4	267.98	282.46	275.384000	20968280.0000
---	--------	--------	------------	---------------

Total results: 2

Total results: 2

8. queryThree() : //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  $\leq 110$ . Companies without a stock ticker should always be shown in the list. Order by company name ascending.

```

public void queryThree() throws SQLException {

```

```

System.out.println("\n==== Query Three =====");
String query = "
    SELECT c.name, c.ticker, sp.closePrice
    FROM company c
    LEFT JOIN stockprice sp ON c.id = sp.companyId
    WHERE sp.priceDate = '2022-08-30'
        AND (sp.closePrice <= 1.1 * (
            SELECT AVG(closePrice)
            FROM stockprice
            WHERE priceDate BETWEEN '2022-08-15' AND '2022-08-19'
            AND companyId = sp.companyId
        ) OR c.ticker IS NULL)
    ORDER BY c.name ASC
";

try (Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query)) {
    System.out.println(resultSetToString(rs));
}
}

```

## Output

```

==== Query Three =====
Total columns: 3
-----
name      ticker    closePrice
-----
Total results: 0

==== Contents and Metadata of Table: company ====
Table Metadata:
Total columns: 5
-----
Column 1:
Name: id
Type: INT
Size: 10
-----
Column 2:
Name: name
Type: VARCHAR
Size: 50
-----
Column 3:
Name: ticker
Type: CHAR
Size: 10
-----
Column 4:
Name: annualRevenue
Type: DECIMAL
Size: 15
-----
Column 5:
Name: numEmployees
Type: INT
Size: 10
-----
Table Data:
Total columns: 5
-----
id      name      ticker    annualRevenue  numEmployees
-----
1      Apple     AAPL      38754000000.00  15400
2      GameStop  GME       61100000.00    1200
3      Handy Repair  null      200000.00      50
4      Microsoft MSFT      19827000000.00  22100
5      StartUp   null      50000.00       3
-----
Total results: 5

```

9. resultSetToString(): converts a ResultSet obtained from the queries to String (Given)

```
public static String resultSetToString(ResultSet rst) throws SQLException {
    StringBuilder buf = new StringBuilder();
    int rowCount = 0;

    if (rst == null) {
        return "ERROR: No ResultSet";
    }

    ResultSetMetaData meta = rst.getMetaData();
    buf.append("Total columns: ").append(meta.getColumnCount()).append('\n');
    buf.append("-----\n");

    for (int j = 1; j <= meta.getColumnCount(); j++) {
        buf.append(String.format("%-15s", meta.getColumnName(j)));
    }
    buf.append("\n-----\n");

    while (rst.next()) {
        for (int j = 1; j <= meta.getColumnCount(); j++) {
            buf.append(String.format("%-15s", rst.getObject(j)));
        }
        buf.append('\n');
        rowCount++;
    }
    buf.append("-----\n");
    buf.append("Total results: ").append(rowCount).append("\n");
    return buf.toString();
}
```

10. resultSetMetaDataToString() : converts resultSetMetaData to String or the String of the metadata (Schema) (Given)

```
public void printTableWithMetadata(String tableName) throws SQLException {
    System.out.println("\n==== Contents and Metadata of Table: " + tableName
+ " =====");
    String query = "SELECT * FROM " + tableName;
    try (Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query)) {
        System.out.println("Table Metadata:");
        System.out.println(resultSetMetaDataToString(rs.getMetaData()));
        System.out.println("Table Data:");
        System.out.println(resultSetToString(rs));
    }
}
```



```

    // resultSetMetaDataToString(): Converts ResultSetMetaData to a readable
    string format
    public static String resultSetMetaDataToString(ResultSetMetaData meta) throws
    SQLException {
        StringBuilder buf = new StringBuilder();
        int columnCount = meta.getColumnCount();

        buf.append("Total columns: ").append(columnCount).append('\n');
        buf.append("-----\n");

        for (int i = 1; i <= columnCount; i++) {
            buf.append("Column ").append(i).append(":\n");
            buf.append("  Name: ").append(meta.getColumnName(i)).append('\n');
            buf.append("  Type:
").append(meta.getColumnTypeName(i)).append('\n');
            buf.append("  Size:
").append(meta.getColumnDisplaySize(i)).append('\n');
            buf.append("-----\n");
        }

        return buf.toString();
    }

```

```

===== Contents and Metadata of Table: stockprice =====
Table Metadata:
Total columns: 7
-----
Column 1:
  Name: companyId
  Type: INT
  Size: 10
-----
Column 2:
  Name: priceDate
  Type: DATE
  Size: 10
-----
Column 3:
  Name: openPrice
  Type: DECIMAL
  Size: 10
-----
Column 4:
  Name: highPrice
  Type: DECIMAL
  Size: 10
-----
Column 5:
  Name: lowPrice
  Type: DECIMAL
  Size: 10
-----
Column 6:
  Name: closePrice
  Type: DECIMAL
  Size: 10
-----
Column 7:
  Name: volume
  Type: INT
  Size: 10
-----

```

Table Data:

Total columns: 7

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
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	22767100

Total results: 14

Connection closed.

Table Data:

Total columns: 7

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
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	22767100

Table Data:

Total columns: 7

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

Table Data:

Total columns: 7

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

Table Data:

Total columns: 7

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
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	22767100

Table Data:  
Total columns: 7

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
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	2022-08-24	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	2022-08-26	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
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	22767100

Total results: 14

Connection closed.

Connection closed

Connection closed.

Complete Code :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

public class App {
    private Connection con;
```

```

    private final String url = "bdm-as3-iitj-g23ai2028.c1a0kg06qqo7.eu-north-
1.rds.amazonaws.com:3306";
    private final String uid = "g23ai2028";
    private final String pw = "g23ai2028";

    // 1. connect() : to connect to the RDS database
    public void connect() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String jdbcUrl = "jdbc:mysql://" + url + "/bdm_as3_iitj_g23ai2028?user="
+ uid + "&password=" + pw + "&connectTimeout=30000";
        System.out.println("\n===== Connecting to Database =====");
        System.out.println("JDBC URL: " + jdbcUrl);
        con = DriverManager.getConnection(jdbcUrl);
        System.out.println("Connection Successful: " + con + "\n");
    }

    // 2. drop() : to drop the table from the database
    public void drop() throws SQLException {
        System.out.println("\n===== Dropping Tables =====");
        String dropStockPrice = "DROP TABLE IF EXISTS stockprice";
        String dropCompany = "DROP TABLE IF EXISTS company";

        try (Statement stmt = con.createStatement()) {
            stmt.executeUpdate(dropStockPrice);
            stmt.executeUpdate(dropCompany);
            System.out.println("Tables dropped successfully.");
        }
    }

    // 3. create() : creates the following table in the database
    // * 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 999,999,999.99 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

public void create() throws SQLException {
    System.out.println("\n==== Creating Tables =====");
    String createCompany = ""
        CREATE TABLE company (
            id INT PRIMARY KEY,
            name VARCHAR(50) NOT NULL,
            ticker CHAR(10),
            annualRevenue DECIMAL(15, 2),
            numEmployees INT
        )
    "";

    String createStockPrice = ""
        CREATE TABLE stockprice (
            companyId INT,
            priceDate DATE,
            openPrice DECIMAL(10, 2),
            highPrice DECIMAL(10, 2),
            lowPrice DECIMAL(10, 2),
            closePrice DECIMAL(10, 2),
            volume INT,
            PRIMARY KEY (companyId, priceDate),
            FOREIGN KEY (companyId) REFERENCES company(id)
        )
    "";

    try (Statement stmt = con.createStatement()) {
        stmt.executeUpdate(createCompany);
        stmt.executeUpdate(createStockPrice);
        System.out.println("Tables created successfully.");
    }
}

// 4. insert() : Inserts test records in the database
// * Data for company table:
// 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
// **
// Data for stockprice table:
// 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, 22767100
```

```
public void insert() throws SQLException {
    System.out.println("\n==== Inserting Records =====");
    String insertCompany = ""
        INSERT INTO company (id, name, ticker, annualRevenue, numEmployees)
VALUES
    (1, 'Apple', 'AAPL', 38754000000.00, 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)
    """;
```

```

String insertStockPrice = ""
    INSERT 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, 22767100)
"";

try (Statement stmt = con.createStatement()) {
    stmt.executeUpdate(insertCompany);
    stmt.executeUpdate(insertStockPrice);
    System.out.println("Test records inserted successfully.");
}

```

```

        printTable("company");
        printTable("stockprice");
    }

    // 5. delete() : Deletes specific records
    // Delete all stock price records where the date is before 2022-08-20 or the
company is
// GameStop
    public void delete() throws SQLException {
        System.out.println("\n===== Deleting Records =====");
        String deleteQuery = ""
            DELETE sp
            FROM stockprice sp
            JOIN company c ON sp.companyId = c.id
            WHERE sp.priceDate < '2022-08-20' OR c.name = 'GameStop'
        "";

        try (Statement stmt = con.createStatement()) {
            int rowsDeleted = stmt.executeUpdate(deleteQuery);
            System.out.println(rowsDeleted + " records deleted from stockprice
table.");
        }

        printTable("stockprice");
    }

    // queryOne(): //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.
    public void queryOne() throws SQLException {
        System.out.println("\n===== Query One =====");
        String query = ""
            SELECT name, annualRevenue, numEmployees
            FROM company
            WHERE numEmployees > 10000 OR annualRevenue < 1000000
            ORDER BY name ASC
        "";

        try (Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query)) {
            System.out.println(resultSetToString(rs));
        }
    }
}

```



```

//      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.

public void queryTwo() throws SQLException {
    System.out.println("\n===== Query Two =====");
    String query = ""
        SELECT companyId,
            MIN(lowPrice) AS lowestPrice,
            MAX(highPrice) AS highestPrice,
            AVG(closePrice) AS avgClosePrice,
            AVG(volume) AS avgVolume
        FROM stockprice
        WHERE priceDate BETWEEN '2022-08-22' AND '2022-08-26'
        GROUP BY companyId
        ORDER BY avgVolume DESC
    "";

    try (Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query)) {
        System.out.println(resultSetToString(rs));
    }
}

//      8. queryThree() : //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.
public void queryThree() throws SQLException {
    System.out.println("\n===== Query Three =====");
    String query = ""
        SELECT c.name, c.ticker, sp.closePrice
        FROM company c
        LEFT JOIN stockprice sp ON c.id = sp.companyId
    "";
}

```

```

        WHERE sp.priceDate = '2022-08-30'
            AND (sp.closePrice <= 1.1 * (
                SELECT AVG(closePrice)
                FROM stockprice
                WHERE priceDate BETWEEN '2022-08-15' AND '2022-08-19'
                AND companyId = sp.companyId
            ) OR c.ticker IS NULL)
    ORDER BY c.name ASC
""";

try (Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery(query)) {
    System.out.println(resultSetToString(rs));
}

// resultSetToString(): converts a ResultSet obtained front he queries to
String (Given)
public void printTable(String tableName) throws SQLException {
    System.out.println("\n==== Contents of Table: " + tableName + " =====");
    String query = "SELECT * FROM " + tableName;
    try (Statement stmt = con.createStatement();
         ResultSet rs = stmt.executeQuery(query)) {
        System.out.println(resultSetToString(rs));
    }
}

// resultSetMetaDataToString() : converts resultSetMetaData to String or the
String of the
// metadata (Schema) (Given)
public static String resultSetToString(ResultSet rst) throws SQLException {
    StringBuilder buf = new StringBuilder();
    int rowCount = 0;

    if (rst == null) {
        return "ERROR: No ResultSet";
    }

    ResultSetMetaData meta = rst.getMetaData();
    buf.append("Total columns: ").append(meta.getColumnCount()).append('\n');
    buf.append("-----\n");

    for (int j = 1; j <= meta.getColumnCount(); j++) {
        buf.append(String.format("%-15s", meta洗洗洗洗洗洗洗(j)));
    }
    buf.append("\n-----\n");
}

```

```

        while (rst.next()) {
            for (int j = 1; j <= meta.getColumnCount(); j++) {
                buf.append(String.format("%-15s", rst.getObject(j)));
            }
            buf.append('\n');
            rowCount++;
        }
        buf.append("-----\n");
        buf.append("Total results: ").append(rowCount).append("\n");
        return buf.toString();
    }

//      10. resultSetMetaDataToString() : converts resultSetMetaData to String or
// the String of the
// metadata (Schema) (Given)

    public void printTableWithMetadata(String tableName) throws SQLException {
        System.out.println("\n==== Contents and Metadata of Table: " + tableName
+ " =====");
        String query = "SELECT * FROM " + tableName;
        try (Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query)) {
            System.out.println("Table Metadata:");
            System.out.println(resultSetMetaDataToString(rs.getMetaData()));
            System.out.println("Table Data:");
            System.out.println(resultSetToString(rs));
        }
    }

// resultSetMetaDataToString(): Converts ResultSetMetaData to a readable
// string format
    public static String resultSetMetaDataToString(ResultSetMetaData meta) throws
SQLException {
        StringBuilder buf = new StringBuilder();
        int columnCount = meta.getColumnCount();

        buf.append("Total columns: ").append(columnCount).append('\n');
        buf.append("-----\n");

        for (int i = 1; i <= columnCount; i++) {
            buf.append("Column ").append(i).append(":\n");
            buf.append("  Name: ").append(meta.getColumnName(i)).append('\n');
            buf.append("  Type: ").append(meta.getColumnTypeName(i)).append('\n');
            buf.append("  Size: ").append(meta.getColumnDisplaySize(i)).append('\n');
            buf.append("-----\n");
        }
    }

```

```

        return buf.toString();
    }

    // Main Method
    public static void main(String[] args) {
        App app = new App();
        try {
            app.connect();
            app.drop();
            app.create();
            app.insert();
            app.delete();
            app.queryOne();
            app.queryTwo();
            app.queryThree();

            // Print table data with metadata
            app.printTableWithMetadata("company");
            app.printTableWithMetadata("stockprice");
            app.close();

        } catch (SQLException | ClassNotFoundException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    // Close the connection
    public void close() throws SQLException {
        if (con != null) {
            con.close();
            System.out.println("Connection closed.");
        }
    }
}

```

Thank you sir for such a good hands on Assignment 3.  
Regards

Shubham Raj  
Roll No : G23AI2028