

BIG DATA MANAGEMENT

Assignment III

Submitted To:

Dr. Dip Shankar Banerjee

Associate Professor

School Of Artificial Intelligence and Data Engineering

Indian Institute of Technology Jodhpur – Rajasthan

Submitted By:

Name: Purushothaman S

Roll Number: G24AI1042

Email: g24ai1042@iitj.ac.in

Course: PGD-Trimester 3

Indian Institute of Technology Jodhpur – Rajasthan

Index

Introduction.....	02
What is RDS.....	02
Key Features of RDS.....	02
How It is Used.....	03
Why Use RDS.....	03
Establishment of Connection Between RDS and Java Editor.....	03
Script to connect.....	05
Script for Drop.....	07
Script for create.....	08
Insert the Data into the Table.....	10
Delete Handling.....	13
QueryOne.....	15
QueryTwo.....	18
QueryThree.....	20
Conclusion.....	22
Git Link.....	22

Introduction

This assignment focuses on building a Java-based interface for interacting with an Amazon RDS (Relational Database Service) instance using SQL operations. It aims to enhance understanding of database integration in enterprise applications by implementing standard database operations such as table creation, data insertion and deletion, and execution of analytical queries. Two tables, company and stock price, are used to simulate real-world data management in the financial sector, particularly tracking corporate profiles and daily stock movements.

What is RDS

RDS (Relational Database Service) is a **managed database service** provided by Amazon Web Services (AWS) that makes it easy to set up, operate, and scale relational databases in the cloud.

Key Features of RDS:

1. **Managed** **Service:**
AWS handles tasks like provisioning, backups, patching, monitoring, and hardware scaling.
2. **Supports** **Multiple** **Database** **Engines:**
You can choose from several popular database systems, including:
 - MySQL
 - PostgreSQL
 - MariaDB
 - Oracle
 - Microsoft SQL Server
 - Amazon Aurora (AWS's own optimized engine)
3. **Scalability:**
You can easily scale up the compute and storage capacity without downtime.
4. **High** **Availability:**
RDS supports automatic failover and multi-AZ (availability zone) deployments for resilience and uptime.
5. **Backups** **and** **Snapshots:**
It provides automated daily backups, manual snapshots, and point-in-time recovery.
6. **Security:**
Supports encryption at rest and in transit, VPC integration, and IAM-based access.

7. Monitoring:

Integrates with Amazon CloudWatch for tracking performance and usage metrics.

How It is Used:

In your assignment, RDS is used to host a **MySQL database**. Your Java code connects to the RDS instance to:

- Create tables (company and stockprice)
- Insert test data
- Run SQL queries and data analysis

Why Use RDS?

- No need to manage database hardware or software manually
- Automated scaling, backups, and failover
- High performance with less operational overhead

Establishment of Connection Between RDS and Java Editor

The establishment of a connection between a Java application and an Amazon RDS (Relational Database Service) instance involves several key steps using JDBC (Java Database Connectivity). Here's a detailed explanation of the connection process as used in your assignment:

1. Load the JDBC Driver

Before any database interaction can take place, the application must load the JDBC driver for the target database (in this case, MySQL):

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

- This line ensures that the Java application can communicate with the MySQL database by loading the MySQL JDBC driver into memory.
- The com.mysql.cj.jdbc.Driver is the class provided by MySQL Connector/J, the official JDBC driver for MySQL 8.0+.

2. Construct the JDBC URL

The JDBC URL defines the path and credentials for connecting to the database:

```
String jdbcUrl = "jdbc:mysql://" + url + "/mydb?user=" + uid + "&password=" + pw;
```

- url contains the RDS endpoint and port (e.g., iitjdb.cpmqoqqutp0.us-east-1.rds.amazonaws.com:3306).

- uid and pw are the username and password (e.g., admin, iitj@12345).
- mydb is the target database within the RDS instance.
- This URL combines everything to tell the JDBC driver where and how to connect.

3. Establish the Connection

The actual connection to the database is made using `DriverManager.getConnection()`:

```
con = DriverManager.getConnection(jdbcUrl);
```

- The DriverManager uses the previously loaded JDBC driver to establish a TCP/IP connection to the Amazon RDS instance.
- Upon successful connection, a Connection object (con) is returned, which is then used for executing SQL statements.

4. Confirmation and Debugging

Print statements are added to confirm the connection status:

```
System.out.println("Connecting to database");
System.out.println("Connection Successful");
```

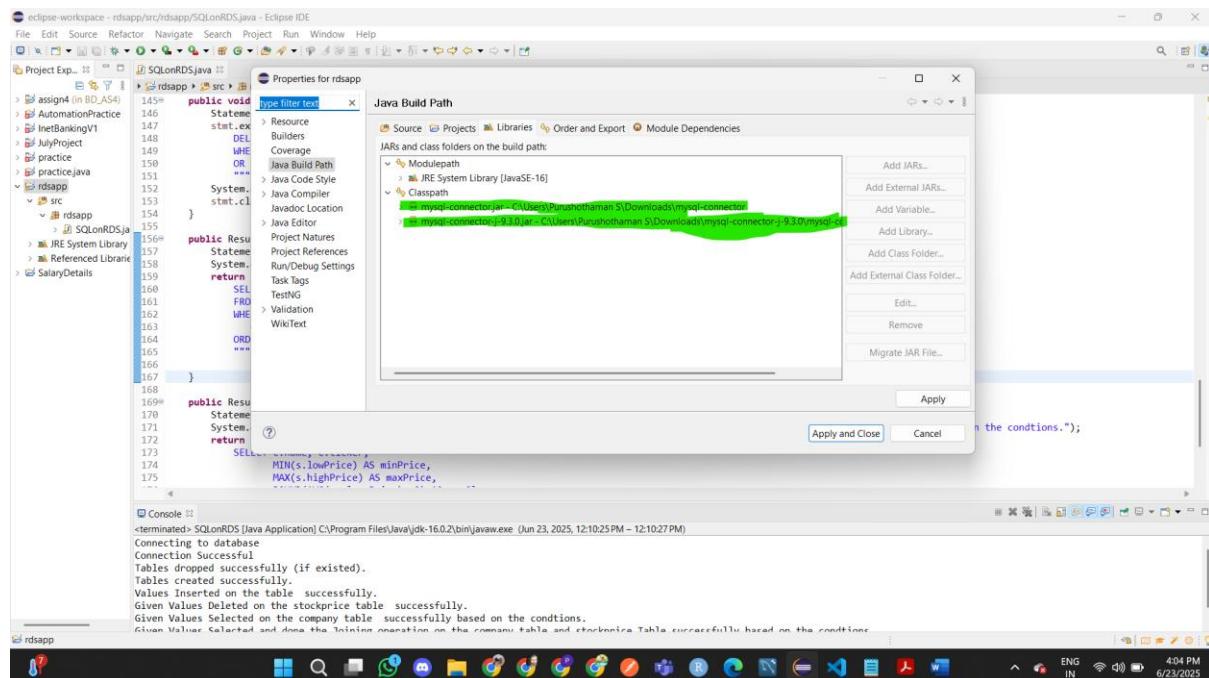
These messages provide real-time feedback to the user or developer to verify whether the connection was successfully established or if troubleshooting is needed.

5. Exception Handling

The entire process is enclosed in a try-catch block to handle potential exceptions:

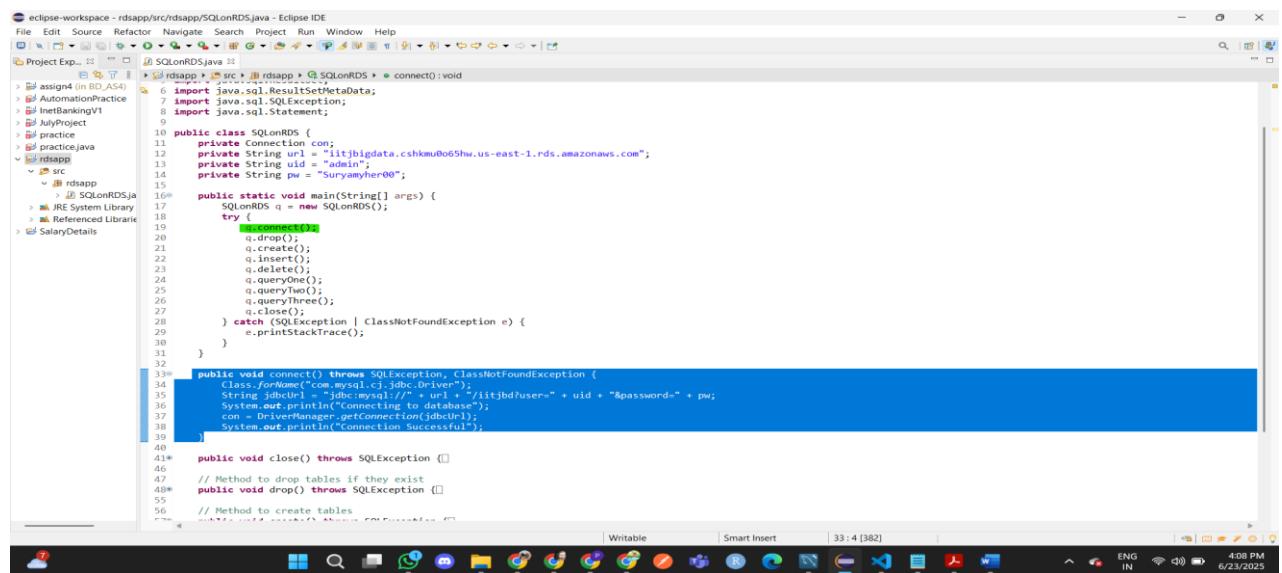
```
catch (SQLException | ClassNotFoundException e) {
    e.printStackTrace();
}
```

- ClassNotFoundException occurs if the JDBC driver class is not found in the classpath.
- SQLException handles errors related to database access, such as invalid credentials or unavailable RDS instance.



Script to connect

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



On AWS - RDS

The screenshot shows the AWS RDS console with the following details:

- Region:** us-east-1
- Databases:** 4 (iitjbdatal, iitjbdatal-instance-1, iitjbdatal-instance-1-us-east-1c)
- DB identifier:** iitjbdatal
- Status:** Available
- Role:** Instance
- Engine:** MySQL Community Server
- Region:** us-east-1b
- Size:** db.t4g.micro
- Recommendations:** 3.29%
- CPU:** 2 Connections
- Current activity:** nor

iitjbdatal-instance-1:

- DB identifier:** iitjbdatal-instance-1
- Status:** Available
- Role:** Regional cluster
- Engine:** Aurora MySQL
- Region:** us-east-1
- Instances:** 2 instances

iitjbdatal-instance-1-us-east-1c:

- DB identifier:** iitjbdatal-instance-1-us-east-1c
- Status:** Available
- Role:** Writer instance
- Engine:** Aurora MySQL
- Region:** us-east-1d
- Size:** db.r7g.large
- Recommendations:** 5.88%
- CPU:** 0.00 sessions
- Current activity:** nor

iitjbdatal-instance-1-us-east-1c:

- DB identifier:** iitjbdatal-instance-1-us-east-1c
- Status:** Available
- Role:** Reader instance
- Engine:** Aurora MySQL
- Region:** us-east-1c
- Size:** db.r7g.large
- Recommendations:** 5.49%
- CPU:** 0.00 sessions
- Current activity:** nor

On Execution

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like assign4, AutomationPractice, netBankingV1, JulyProject, practice, practice.java, rdsapp, and SQLonRDS.java.
- Code Editor:** Displays the `SQLonRDS.java` file containing Java code for connecting to an RDS database.
- Console:** Shows the terminal output of the Java application, indicating a successful connection to the database.

```
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer   SQLonRDS.java
src rdsapp SQLonRDS.java main(String[])
15 public static void main(String[] args) {
16     SQLonRDS q = new SQLonRDS();
17     try {
18         q.connect();
19         // q.drop();
20         // q.create();
21         // q.insert();
22         // q.delete();
23         // q.queryOne();
24         // q.queryTwo();
25         // q.queryThree();
26         q.close();
27     } catch (SQLException | ClassNotFoundException e) {
28         e.printStackTrace();
29     }
30 }
31
32 public void connect() throws SQLException, ClassNotFoundException {
33     Class.forName("com.mysql.cj.jdbc.Driver");
34     String jdbcUrl = "jdbc:mysql://" + url + "/iitjbd?user=" + uid + "&password=" + pw;
35     System.out.println("Connecting to database");
36     con = DriverManager.getConnection(jdbcUrl);
37     System.out.println("Connection Successful");
38 }
39
40 public void close() throws SQLException {
}
}

Console
<terminated> SQLonRDS [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (Jun 23, 2025, 4:13:24 PM - 4:13:28 PM)
Connecting to database
Connection Successful
```

Script for Drop

```
// Method to drop tables if they exist

public void drop() throws SQLException {

    Statement stmt = con.createStatement();

    stmt.executeUpdate("DROP TABLE IF EXISTS stockprice");

    stmt.executeUpdate("DROP TABLE IF EXISTS company");

    System.out.println("Tables dropped successfully (if existed).");

    stmt.close();

}
```

On Script

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a file named `SQLonRDS.java` under the `rdsapp/src/rdsapp` folder.
- Code Editor:** Displays the Java code for `SQLonRDS.java`. The code includes a `drop()` method that drops two tables if they exist: `stockprice` and `company`. It also includes `connect()` and `close()` methods.
- Console Output:** Shows the terminal output of the application. It starts with "Connecting to database" and "Connection Successful". At the bottom, it displays the message "Tables dropped successfully (if existed.)".
- Bottom Bar:** Shows the Windows taskbar with various application icons and system status indicators.

Script for create: create () creates the following table in the database

create (): Creates two tables—company and stockprice—with specific schemas and constraints, including primary and foreign keys.

Script

```
public void create() throws SQLException {  
    Statement stmt = con.createStatement();  
  
    String createCompanyTable = """";  
  
    CREATE TABLE company (  
        id INT PRIMARY KEY,  
        name VARCHAR(50),  
        ticker CHAR(10),  
        annualRevenue DECIMAL(15,2),  
        numEmployees INT  
    )  
    """";  
  
    String createStockPriceTable = """";  
  
    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)  
    )  
    """";  
  
    stmt.executeUpdate(createCompanyTable);  
    stmt.executeUpdate(createStockPriceTable);  
    System.out.println("Tables created successfully.");  
    stmt.close();  
}
```

Execution

The screenshot shows the Eclipse IDE interface. The left pane displays a project structure with a file named SQLonRDS.java selected. The code in the editor window is as follows:

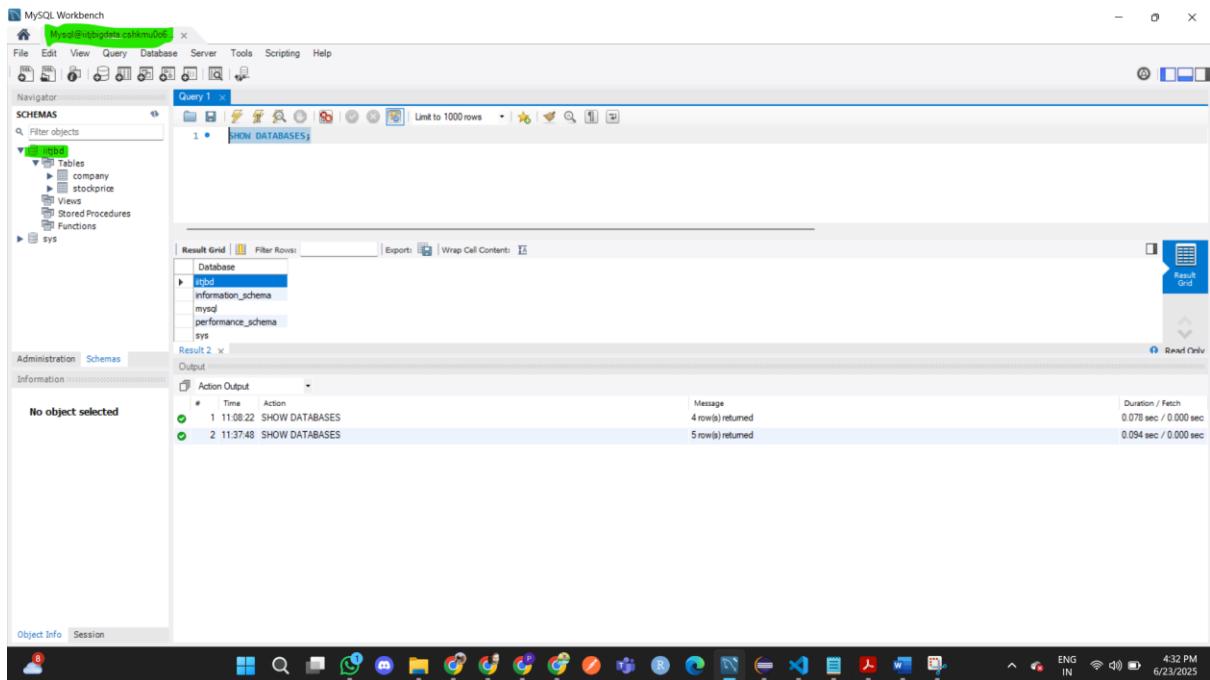
```
1 package rdsapp;
2 import java.sql.*;
3
4 public class SQLonRDS {
5     public void assign4 (in BD_A54)
6     {
7         // Method to drop tables if they exist
8         public void drop() throws SQLException {
9             q.create();
10            q.insert();
11            q.delete();
12            q.update();
13            q.queryOne();
14            q.queryTwo();
15            q.queryThree();
16            q.close();
17        } catch (SQLException | ClassNotFoundException e) {
18            e.printStackTrace();
19        }
20    }
21
22    public void connect() throws SQLException, ClassNotFoundException {
23        Connection con = DriverManager.getConnection("jdbc:mysql://"+host+":"+port+"/"+db,"root","password");
24        Statement stmt = con.createStatement();
25        String createCompanyTable = """
26            CREATE TABLE company (
27                id INT PRIMARY KEY,
28                name VARCHAR(255),
29                address VARCHAR(255),
30                phone VARCHAR(15),
31                email VARCHAR(255)
32            );
33
34        public void close() throws SQLException {
35            // Method to drop tables if they exist
36            public void drop() throws SQLException {
37                q.create();
38            }
39        }
40    }
41
42    public void close() throws SQLException {
43        // Method to drop tables if they exist
44        public void drop() throws SQLException {
45            q.create();
46        }
47    }
48    // Method to create tables
49    public void create() throws SQLException {
50        Statement stmt = con.createStatement();
51        String createCompanyTable = """
52            CREATE TABLE company (
53                id INT PRIMARY KEY,
54                name VARCHAR(255),
55                address VARCHAR(255),
56                phone VARCHAR(15),
57                email VARCHAR(255)
58            );
59        public void close() throws SQLException {
60            q.create();
61        }
62    }
63
64    public void insert() throws SQLException {
65        Statement stmt = con.createStatement();
66        String insertQuery = "INSERT INTO company (name, address, phone, email) VALUES ('ITI', 'Chennai', '9842112345', 'iti@iti.ac.in')";
67        stmt.executeUpdate(insertQuery);
68    }
69
70    public void update() throws SQLException {
71        Statement stmt = con.createStatement();
72        String updateQuery = "UPDATE company SET address = 'Mumbai' WHERE id = 1";
73        stmt.executeUpdate(updateQuery);
74    }
75
76    public void delete() throws SQLException {
77        Statement stmt = con.createStatement();
78        String deleteQuery = "DELETE FROM company WHERE id = 1";
79        stmt.executeUpdate(deleteQuery);
80    }
81
82    public void queryOne() throws SQLException {
83        Statement stmt = con.createStatement();
84        String selectQuery = "SELECT * FROM company WHERE id = 1";
85        ResultSet rs = stmt.executeQuery(selectQuery);
86        while (rs.next()) {
87            System.out.println("ID: " + rs.getInt("id"));
88            System.out.println("Name: " + rs.getString("name"));
89            System.out.println("Address: " + rs.getString("address"));
90            System.out.println("Phone: " + rs.getString("phone"));
91            System.out.println("Email: " + rs.getString("email"));
92        }
93    }
94
95    public void queryTwo() throws SQLException {
96        Statement stmt = con.createStatement();
97        String selectQuery = "SELECT * FROM company";
98        ResultSet rs = stmt.executeQuery(selectQuery);
99        while (rs.next()) {
100            System.out.println("ID: " + rs.getInt("id"));
101            System.out.println("Name: " + rs.getString("name"));
102            System.out.println("Address: " + rs.getString("address"));
103            System.out.println("Phone: " + rs.getString("phone"));
104            System.out.println("Email: " + rs.getString("email"));
105        }
106    }
107
108    public void queryThree() throws SQLException {
109        Statement stmt = con.createStatement();
110        String selectQuery = "SELECT * FROM company WHERE id > 1";
111        ResultSet rs = stmt.executeQuery(selectQuery);
112        while (rs.next()) {
113            System.out.println("ID: " + rs.getInt("id"));
114            System.out.println("Name: " + rs.getString("name"));
115            System.out.println("Address: " + rs.getString("address"));
116            System.out.println("Phone: " + rs.getString("phone"));
117            System.out.println("Email: " + rs.getString("email"));
118        }
119    }
120
121    public void close() throws SQLException {
122        con.close();
123    }
124}
```

The right pane shows the Eclipse IDE's status bar and a taskbar at the bottom.

Verification on Work bench

The MySQL Work bench and the RDS has been connected using Instance Id and UID and PW

The screenshot shows the AWS RDS Database Details page for the database 'itibigdata'. The left sidebar shows navigation options like Aurora and RDS, Databases, and Monitoring. The main summary section provides details such as DB identifier (itibigdata), Status (Available), Role (Instance), Engine (MySQL Community), and Region & AZ (us-east-1b). The Connectivity & security tab is selected, displaying information about the endpoint (IP address), VPC (vpc-01f603c59f501b08a), and subnet group (default-vpc-01f603c59f501b08a). It also shows network settings (IPv4) and security groups (allow-jdbc-access, default, and another active group). The bottom of the page includes standard AWS footer links and a taskbar.



Insert the Data into the Table

`insert()`: Inserts predefined sample data into both tables.

Script

```
public void insert() throws SQLException {
    Statement stmt = con.createStatement();
    // Insert into company
    stmt.executeUpdate("INSERT INTO company 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 into stockprice
    stmt.executeUpdate("""
        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)
    ");
}

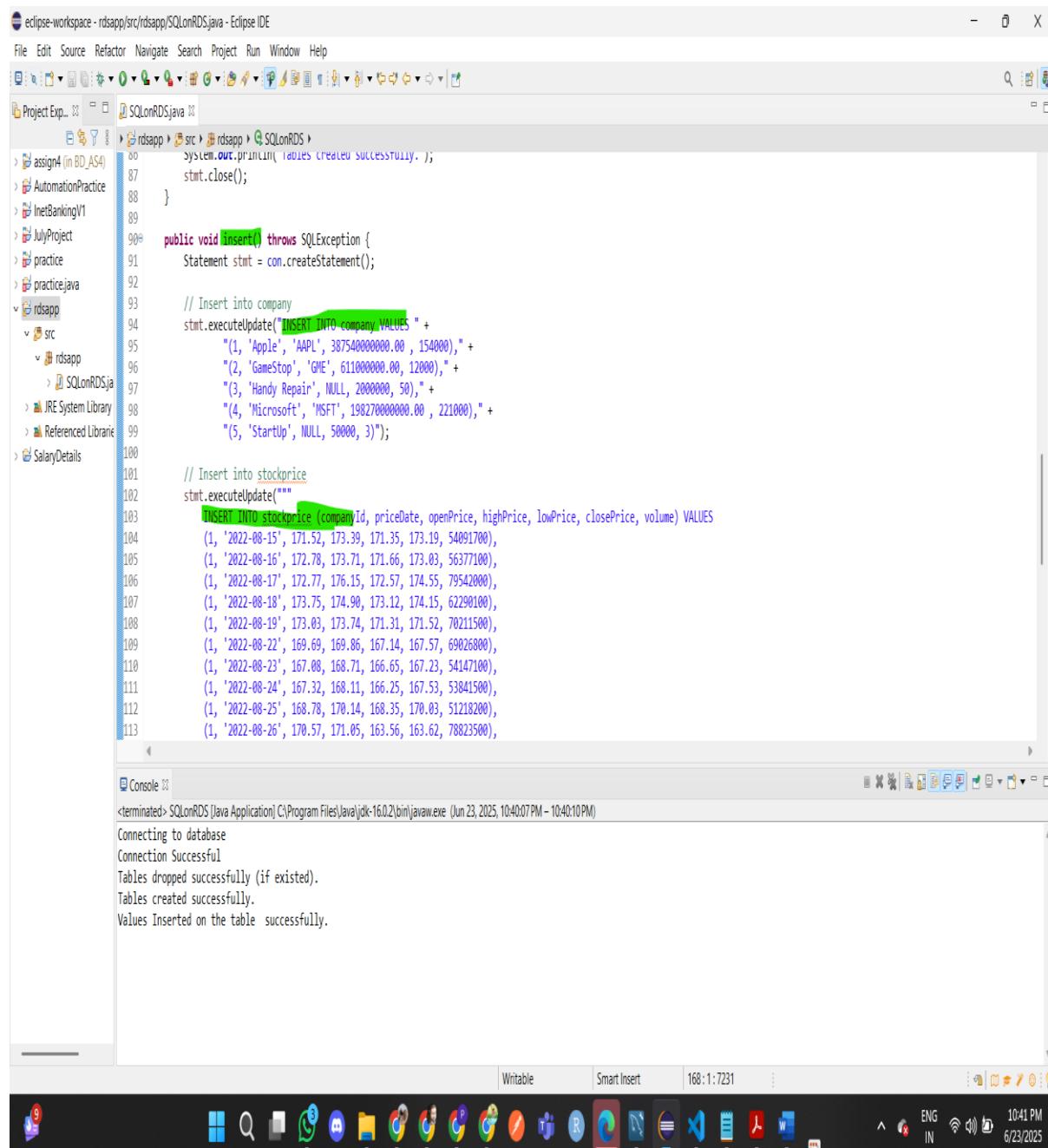
System.out.println("Values Inserted on the table successfully.");

stmt.close();

}

```

On Script



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like `SQLonRDS.java`, `assign4 (in BD_A54)`, `AutomationPractice`, `InetBankingV1`, `JulyProject`, `practice`, and `practice.java`.
- Code Editor:** Displays the `SQLonRDS.java` file containing Java code for database operations. The code includes methods for creating tables, inserting data into the `company` and `stockprice` tables, and closing statements.
- Console:** Shows the output of the application's execution. It includes messages like "Connecting to database", "Connection Successful", "Tables dropped successfully (if existed).", "Tables created successfully.", and "Values Inserted on the table successfully.".
- Bottom Bar:** Includes standard Eclipse icons for file operations, search, and help, along with system status indicators for battery, signal, and date/time (10:41 PM, 6/23/2025).

```

eclipse-workspace - rdsapp/src/rdsapp/SQLonRDS.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help
Project Exp... SQLonRDS.java
1 assign4 (in BD_A54)
2 AutomationPractice
3 InetBankingV1
4 JulyProject
5 practice
6 practice.java
7 rdsapp
8   src
9     rdsapp
10    > SQLonRDS.java
11 JRE System Library
12 Referenced Libraries
13 SalaryDetails

public class SQLonRDS {
    public void createTables() throws SQLException {
        Statement stmt = con.createStatement();
        // Create company table
        stmt.executeUpdate("CREATE TABLE company (id INT, name VARCHAR(50), revenue DECIMAL(10,2), profit DECIMAL(10,2), marketCap DECIMAL(10,2), stockPrice DECIMAL(10,2), volume INT);");
        // Create stockprice table
        stmt.executeUpdate("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);");
    }

    public void insert() throws SQLException {
        Statement stmt = con.createStatement();
        // Insert into company
        stmt.executeUpdate("INSERT INTO company VALUES " +
                           "(1, 'Apple', 'AAPL', 387540000000.00, 154000), " +
                           "(2, 'GameStop', 'GME', 611000000.00, 12800), " +
                           "(3, 'Handy Repair', NULL, 2000000, 50), " +
                           "(4, 'Microsoft', 'MSFT', 19827000000.00, 221000), " +
                           "(5, 'StartUp', NULL, 50000, 3)");
        // Insert into stockprice
        stmt.executeUpdate("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, '2022-08-20', 169.69, 169.86, 167.14, 167.51, 69026800),
                           (1, '2022-08-21', 167.08, 168.71, 166.65, 167.23, 54147100),
                           (1, '2022-08-22', 167.32, 168.11, 166.25, 167.53, 53841500),
                           (1, '2022-08-23', 168.78, 170.14, 168.35, 170.03, 51218200),
                           (1, '2022-08-24', 170.57, 171.05, 163.56, 163.62, 78823500),
                           (1, '2022-08-25', 171.05, 171.85, 163.56, 163.62, 78823500),
                           (1, '2022-08-26', 171.85, 172.57, 163.56, 163.62, 78823500),
                           (1, '2022-08-27', 172.57, 173.39, 171.35, 173.19, 54091700),
                           (1, '2022-08-28', 173.39, 174.90, 173.12, 174.15, 62290100),
                           (1, '2022-08-29', 174.90, 176.15, 173.75, 174.55, 79542000),
                           (1, '2022-08-30', 176.15, 177.78, 175.32, 177.51, 62290100),
                           (1, '2022-08-31', 177.78, 179.41, 176.65, 179.03, 79542000));
    }
}

Console
<terminated> SQLonRDS [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\java.exe (Jun 23, 2025, 10:40:07 PM - 10:40:10 PM)
Connecting to database
Connection Successful
Tables dropped successfully (if existed).
Tables created successfully.
Values Inserted on the table successfully.


```

Verification on the MySQL Work Bench

The screenshot shows the MySQL Workbench interface. In the Navigator pane, under the 'company' schema, there is a 'Tables' section containing 'company'. The 'Query' tab is active, displaying a result grid for the 'company' table. The table has columns: id, name, ticker, annualRevenue, and numEmployees. The data shows five rows: Apple (AAPL), Google (GOOGL), Amazon (AMZN), Microsoft (MSFT), and Starbucks (SBUX). The 'Output' pane at the bottom shows the execution history of SQL commands.

id	name	ticker	annualRevenue	numEmployees
1	Apple	AAPL	38754000000.00	154000
2	Google	GOOGL	61300000000.00	12000
3	Amazon	AMZN	182700000000.00	52000
4	Microsoft	MSFT	198270000000.00	221000
5	Starbucks	SBUX	50000.00	3

Action Output:

#	Time	Action	Message	Duration / Fetch
1	11:08:22	SHOW DATABASES	4 row(s) returned	0.078 sec / 0.000 sec
2	11:37:48	SHOW DATABASES	5 row(s) returned	0.094 sec / 0.000 sec
3	22:32:35	Select * from company LIMIT 0, 1000	0 row(s) returned	0.110 sec / 0.000 sec
4	22:40:18	Select * from company LIMIT 0, 1000	5 row(s) returned	0.078 sec / 0.000 sec

Delete Handling

`delete()`: Deletes records based on specific conditions, such as date thresholds and company identity.

Script

```
public static void main(String[] args) {  
    SQLonRDS q = new SQLonRDS();  
  
    try {  
        q.connect();  
  
        q.drop();  
  
        q.create();  
  
        q.insert();  
  
        q.delete();  
  
    } catch (SQLException | ClassNotFoundException e) {  
        e.printStackTrace();  
    }  
}
```

```
public void delete() throws SQLException {  
    Statement stmt = con.createStatement();  
  
    stmt.executeUpdate(""  
        DELETE FROM stockprice  
        WHERE priceDate < '2022-08-20'  
        OR companyId = (SELECT id FROM company WHERE name = 'GameStop')  
        "");  
  
    System.out.println("Given Values Deleted on the stockprice table successfully.");  
  
    stmt.close();  
}
```

On Script

eclipse-workspace - rdsapp/src/rdsapp/SQLonRDS.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Exp... SQLonRDS.java

```
1 assignA (in BD,AS4)
2 AutomationPractice
3 netBankingV1
4 JulyProject
5 practice
6 practice.java
7 rdsapp
8 src
9 rdsapp
10 SQLonRDS.java
11 JRE System Library
12 Referenced Libraries
13 SalaryDetails

14 public class SQLonRDS {
15     // Method to delete data from stockprice table
16     public void delete() throws SQLException {
17         Statement stmt = con.createStatement();
18         stmt.executeUpdate("DELETE FROM stockprice
19             WHERE priceDate < '2022-08-20'
20             OR companyId = (SELECT id FROM company WHERE name = 'GameStop')
21             ");
22         System.out.println("Given Values Deleted on the stockprice table successfully.");
23         stmt.close();
24     }
25
26     // Method to query one record
27     public ResultSet queryOne() throws SQLException {
28         Statement stmt = con.createStatement();
29         ResultSet rs = stmt.executeQuery("SELECT * FROM stockprice
30             WHERE priceDate > '2022-08-20'
31             AND companyId = (SELECT id FROM company WHERE name = 'GameStop')
32             ");
33         return rs;
34     }
35
36     // Method to query two records
37     public ResultSet queryTwo() throws SQLException {
38         Statement stmt = con.createStatement();
39         ResultSet rs = stmt.executeQuery("SELECT * FROM stockprice
40             WHERE priceDate > '2022-08-20'
41             AND companyId = (SELECT id FROM company WHERE name = 'GameStop')
42             ");
43         return rs;
44     }
45
46     // Method to query three records
47     public ResultSet queryThree() throws SQLException {
48         Statement stmt = con.createStatement();
49         ResultSet rs = stmt.executeQuery("SELECT * FROM stockprice
50             WHERE priceDate > '2022-08-20'
51             AND companyId = (SELECT id FROM company WHERE name = 'GameStop')
52             ");
53         return rs;
54     }
55
56     // Method to close connection
57     public void close() throws SQLException {
58     }
59
60     // Method to drop tables if they exist
61     public void drop() throws SQLException {
62     }
63
64     // Method to create tables
65     public void create() throws SQLException {
66     }
67
68     // Method to insert data
69     public void insert() throws SQLException {
70     }
71
72     // Method to delete data
73     public void delete() throws SQLException {
74         Statement stmt = con.createStatement();
75         stmt.executeUpdate("DELETE FROM stockprice
76             WHERE priceDate < '2022-08-20'
77             OR companyId = (SELECT id FROM company WHERE name = 'GameStop')
78             ");
79         System.out.println("Given Values Deleted on the stockprice table successfully.");
80         stmt.close();
81     }
82
83     // Method to query one record
84     public ResultSet queryOne() throws SQLException {
85         Statement stmt = con.createStatement();
86         ResultSet rs = stmt.executeQuery("SELECT * FROM stockprice
87             WHERE priceDate > '2022-08-20'
88             AND companyId = (SELECT id FROM company WHERE name = 'GameStop')
89             ");
90         return rs;
91     }
92
93     // Method to query two records
94     public ResultSet queryTwo() throws SQLException {
95         Statement stmt = con.createStatement();
96         ResultSet rs = stmt.executeQuery("SELECT * FROM stockprice
97             WHERE priceDate > '2022-08-20'
98             AND companyId = (SELECT id FROM company WHERE name = 'GameStop')
99             ");
100        return rs;
101    }
102
103    // Method to query three records
104    public ResultSet queryThree() throws SQLException {
105        Statement stmt = con.createStatement();
106        ResultSet rs = stmt.executeQuery("SELECT * FROM stockprice
107            WHERE priceDate > '2022-08-20'
108            AND companyId = (SELECT id FROM company WHERE name = 'GameStop')
109            ");
110        return rs;
111    }
112
113    // Method to print stack trace
114    public void printStackTrace() {
115        System.out.println("Stack Trace:");
116        StackTraceElement[] stackTrace = Thread.currentThread().getStackTrace();
117        for (StackTraceElement element : stackTrace) {
118            System.out.println(element);
119        }
120    }
121
122    // Main method
123    public static void main(String[] args) {
124        SQLonRDS obj = new SQLonRDS();
125        obj.delete();
126        obj.queryOne();
127        obj.queryTwo();
128        obj.queryThree();
129        obj.printStackTrace();
130    }
131 }
```

Console

```
<terminated> SQLonRDS [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (Jun 23, 2025, 10:40:07 PM - 10:40:10 PM)
Connecting to database
Connection Successful
Tables dropped successfully (if existed).
Tables created successfully.
```

Verification on MySQL Work Bench after the delete the data based on the condition

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database is set to 'MySql@itrigbigdata.cshkmu06'. The main area displays a query results grid for the 'stockprice' table, which contains 36 rows of historical price data. Below the grid is an 'Output' panel showing the execution history of the query.

companyid	priceDate	openPrice	highPrice	lowPrice	closePrice	volume
1	2022-08-16	172.78	172.71	171.66	172.03	56377100
1	2022-08-17	172.77	176.15	172.57	174.55	79540200
1	2022-08-18	173.75	174.99	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

Action Output:

#	Time	Action	Message	Duration / Fetch
1	11:08:22	SHOW DATABASES	4 row(s) returned	0.078 sec / 0.000 sec
2	11:37:48	SHOW DATABASES	5 row(s) returned	0.094 sec / 0.000 sec
3	22:32:35	Select * from company LIMIT 0, 1000	0 row(s) returned	0.110 sec / 0.000 sec
4	22:40:18	Select * from company LIMIT 0, 1000	5 row(s) returned	0.078 sec / 0.000 sec
5	22:50:20	Select * from stockprice LIMIT 0, 1000	36 row(s) returned	0.098 sec / 0.000 sec

QueryOne: QueryOne() 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.

Script

```
public static void main(String[] args) {
    SQLonRDS q = new SQLonRDS();

    try {
        q.connect();
        q.drop();
        q.create();
        q.insert();
        q.delete();
        q.queryOne();
        //        q.queryTwo();
        //        q.queryThree();
        //        q.close();
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

```

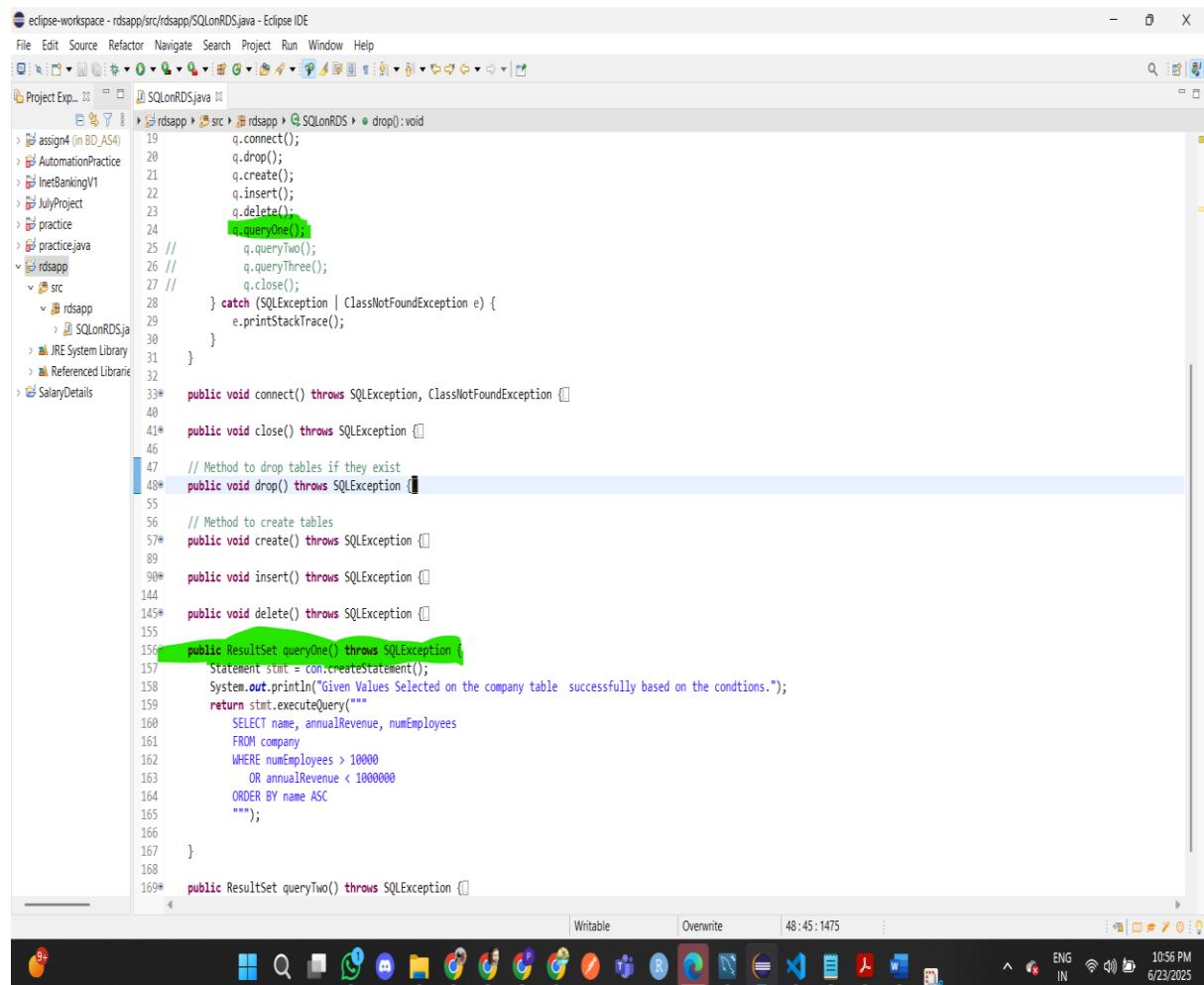
    }

}

public ResultSet queryOne() throws SQLException {
    Statement stmt = con.createStatement();
    System.out.println("Given Values Selected on the company table successfully based on the conditions.");
    return stmt.executeQuery(""""
        SELECT name, annualRevenue, numEmployees
        FROM company
        WHERE numEmployees > 10000
        OR annualRevenue < 1000000
        ORDER BY name ASC
    """);
}

```

On Script



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with a file named `SQLonRDS.java` selected.
- Code Editor:** Displays the `SQLonRDS.java` file content. The code implements methods for connecting to a database, dropping tables, creating tables, inserting data, deleting data, and executing queries. The `queryOne()` method is highlighted with a green selection bar.
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and run.
- Bottom Status Bar:** Shows the status "Writable", the current time "48:45:1475", and system information like battery level, network, and date/time "1056 PM IN 6/23/2025".

```

eclipse-workspace - rdsapp/src/rdsapp/SQLonRDS.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Exp... SQLonRDS.java
  rdsapp src rdsapp SQLonRDS.java drop():void
19     q.connect();
20     q.drop();
21     q.create();
22     q.insert();
23     q.delete();
24     q.queryOne();
25 //     q.queryTwo();
26 //     q.queryThree();
27 //     q.close();
28 } catch (SQLException | ClassNotFoundException e) {
29     e.printStackTrace();
30 }
31
32
33 public void connect() throws SQLException, ClassNotFoundException {}
34
35 public void close() throws SQLException {}
36
37 // Method to drop tables if they exist
38 public void drop() throws SQLException {}
39
40 // Method to create tables
41 public void create() throws SQLException {}
42
43 public void insert() throws SQLException {}
44
45 public void delete() throws SQLException {}
46
47
48 public ResultSet queryOne() throws SQLException {
49     Statement stmt = con.createStatement();
50     System.out.println("Given Values Selected on the company table successfully based on the conditions.");
51     return stmt.executeQuery(""""
52         SELECT name, annualRevenue, numEmployees
53         FROM company
54         WHERE numEmployees > 10000
55         OR annualRevenue < 1000000
56         ORDER BY name ASC
57     """);
58 }
59
60 public ResultSet queryTwo() throws SQLException {}

```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with files like `SQLOnRDS.java`, `rsdapp/src/SQLOnRDS.java`, and `rsdapp/src/rsdapp.java`.
- Code Editor:** Displays the `SQLOnRDS.java` file containing Java code for database operations using JDBC.
- Console:** Shows the terminal output of the application's execution, indicating successful database connection, table creation, data insertion, and deletion.

```
<terminated> SQLOnRDS [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\java.exe (Jun 23, 2025, 10:56:39 PM - 10:56:42 PM)
Connecting to database
Connection Successful
Tables dropped successfully (if existed).
Tables created successfully.
Values Inserted on the table successfully.
Given values Deleted on the stockprice table successfully.
Given values Selected on the company table successfully based on the conditions.
```

Verify in MySQL Work Bench

The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** The main window displays a query in the "Query 1" tab:

```
1 SELECT name, annualRevenue, numEmployees
2   FROM company
3  WHERE numEmployees > 10000
4      OR annualRevenue < 1000000
5 ORDER BY name ASC
```
- Results Grid:** Below the query editor, the results are shown in a grid format:

name	annualRevenue	numEmployees
Apple	387540000000.00	154000
GameStop	6110000000.00	12000
Microsoft	198270000000.00	221000
StartUp	50000.00	3
- Table Definition:** On the left, the "Tables" section of the Navigator shows the "company" table with columns: id (int PK), name (varchar(50)), ticker (char(10)), annualRevenue (decimal(15,2)), and numEmployees (int).
- History:** At the bottom, the "Output" pane shows the execution history of the session:| # | Action | Message | Duration / Fetch |
| --- | --- | --- | --- |
| 1 | SHOW DATABASES | 4 row(s) returned | 0.078 sec / 0.000 sec |
| 2 | SHOW DATABASES | 5 row(s) returned | 0.094 sec / 0.000 sec |
| 3 | Select * from company LIMIT 0, 1000 | 0 row(s) returned | 0.110 sec / 0.000 sec |
| 4 | Select * from company LIMIT 0, 1000 | 5 row(s) returned | 0.078 sec / 0.000 sec |
| 5 | Select * from stockprice LIMIT 0, 1000 | 36 row(s) returned | 0.094 sec / 0.000 sec |
| 6 | SELECT name, annualRevenue, numEmployees FROM company WHERE numEmployees > 10... 4 row(s) returned | | 0.156 sec / 0.000 sec |

QueryTwo()for returning ResultSet 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

Script

```
public static void main(String[] args) {
    SQLonRDS q = new SQLonRDS();

    try {
        q.connect();
        q.drop();
        q.create();
        q.insert();
        q.delete();
        q.queryOne();
        q.queryTwo();
        //    q.queryThree();
        //    q.close();
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}

public ResultSet queryTwo() throws SQLException {
    Statement stmt = con.createStatement();

    System.out.println("Given Values Selected and done the Joining operation on the company table and stockprice Table successfully based on the conditions.");
    return stmt.executeQuery("""
        SELECT c.name, c.ticker,
               MIN(s.lowPrice) AS minPrice,
               MAX(s.highPrice) AS maxPrice,
               ROUND(AVG(s.closePrice), 2) AS avgClose,
               ROUND(AVG(s.volume), 2) AS avgVolume
        FROM company c
        JOIN stockprice s ON c.id = s.companyId
    """);
}
```

WHERE s.priceDate BETWEEN '2022-08-22' AND '2022-08-26'

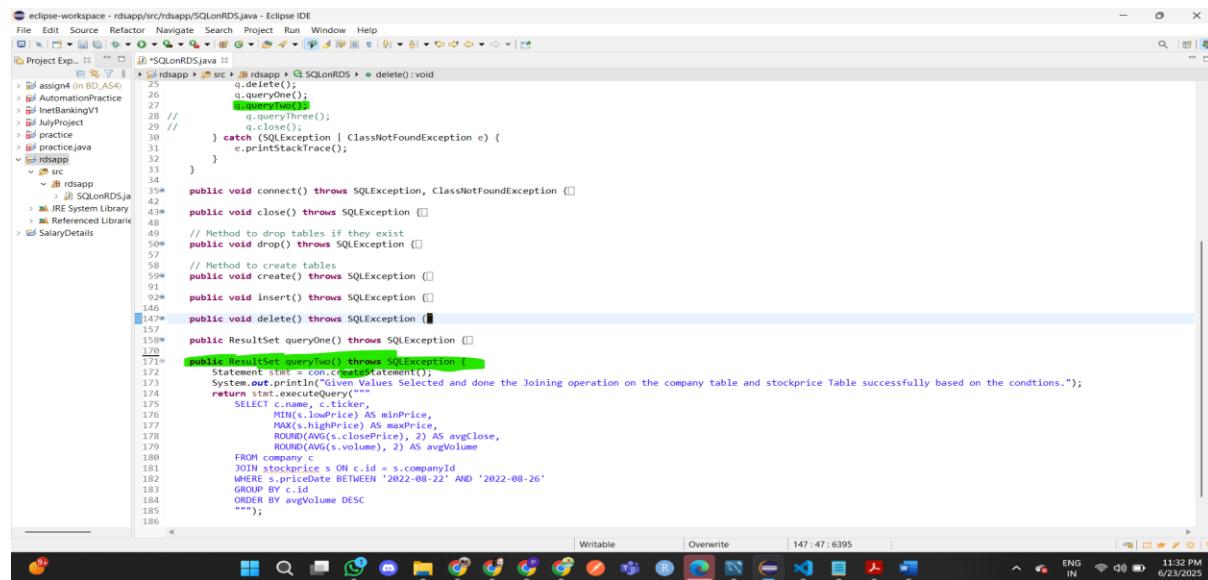
GROUP BY c.id

ORDER BY avgVolume DESC

"");

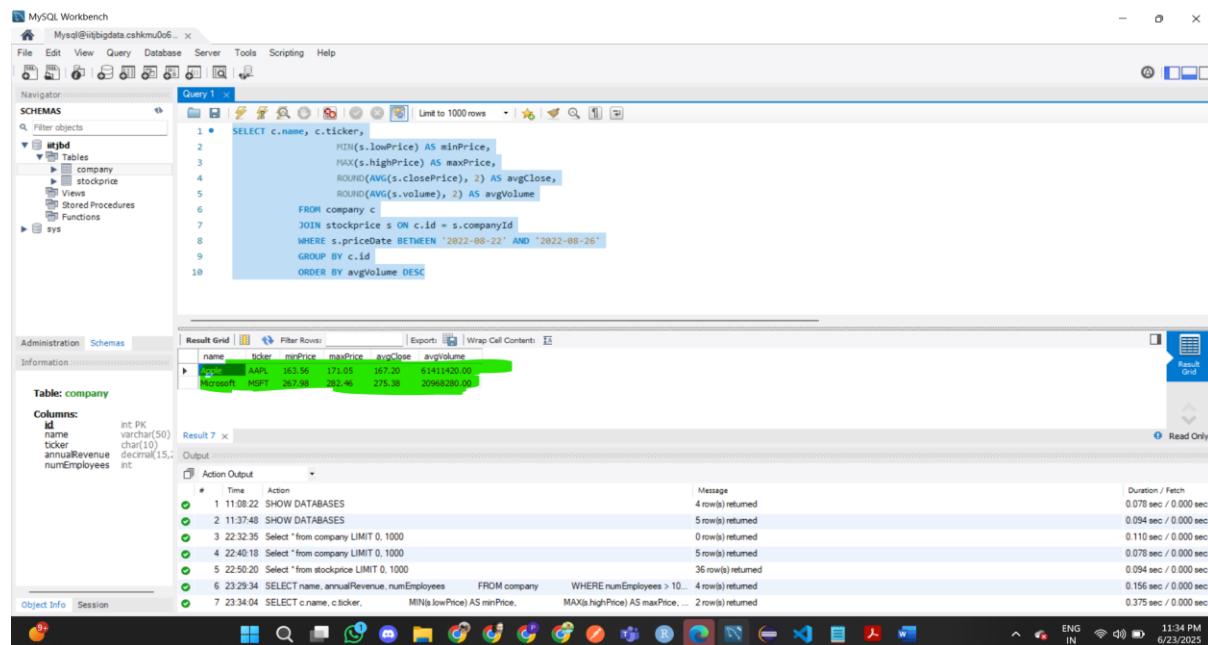
}

On Script



```
eclipse-workspace - rdsapp/src/rdsapp/SQLonRDS.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer src SQLonRDS.java
1 package rdsapp;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.sql.Statement;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class SQLonRDS {
12     // Method to drop tables if they exist
13     public void drop() throws SQLException {
14         String query = "DROP TABLE IF EXISTS company, stockprice";
15         Statement stmt = conn.createStatement();
16         stmt.executeUpdate(query);
17     }
18
19     // Method to create tables
20     public void create() throws SQLException {
21         String query = "CREATE TABLE company (c_id INT PRIMARY KEY, name VARCHAR(50), ticker CHAR(10), annualRevenue DECIMAL(15, 2), numEmployees INT);";
22         String query2 = "CREATE TABLE stockprice (s_id INT PRIMARY KEY, c_id INT, date DATE, lowPrice DECIMAL(15, 2), highPrice DECIMAL(15, 2), closePrice DECIMAL(15, 2), volume INT);";
23         Statement stmt = conn.createStatement();
24         stmt.executeUpdate(query);
25         stmt.executeUpdate(query2);
26     }
27
28     // Method to insert data
29     public void insert() throws SQLException {
30         String query = "INSERT INTO company (name, ticker, annualRevenue, numEmployees) VALUES ('AMAZON', 'AMZN', 100000000000.0, 1000000);";
31         Statement stmt = conn.createStatement();
32         stmt.executeUpdate(query);
33     }
34
35     // Method to delete data
36     public void delete() throws SQLException {
37         String query = "DELETE FROM company WHERE name = 'AMAZON';";
38         Statement stmt = conn.createStatement();
39         stmt.executeUpdate(query);
40     }
41
42     // Method to connect to database
43     public void connect() throws SQLException, ClassNotFoundException {
44         Class.forName("com.mysql.cj.jdbc.Driver");
45         conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/bigdata", "root", "password");
46     }
47
48     // Method to close connection
49     public void close() throws SQLException {
50         conn.close();
51     }
52
53     // Method to drop tables if they exist
54     public void drop() throws SQLException {
55         String query = "DROP TABLE IF EXISTS company, stockprice";
56         Statement stmt = conn.createStatement();
57         stmt.executeUpdate(query);
58     }
59
60     // Method to create tables
61     public void create() throws SQLException {
62         String query = "CREATE TABLE company (c_id INT PRIMARY KEY, name VARCHAR(50), ticker CHAR(10), annualRevenue DECIMAL(15, 2), numEmployees INT);";
63         String query2 = "CREATE TABLE stockprice (s_id INT PRIMARY KEY, c_id INT, date DATE, lowPrice DECIMAL(15, 2), highPrice DECIMAL(15, 2), closePrice DECIMAL(15, 2), volume INT);";
64         Statement stmt = conn.createStatement();
65         stmt.executeUpdate(query);
66         stmt.executeUpdate(query2);
67     }
68
69     // Method to insert data
70     public void insert() throws SQLException {
71         String query = "INSERT INTO company (name, ticker, annualRevenue, numEmployees) VALUES ('AMAZON', 'AMZN', 100000000000.0, 1000000);";
72         Statement stmt = conn.createStatement();
73         stmt.executeUpdate(query);
74     }
75
76     // Method to delete data
77     public void delete() throws SQLException {
78         String query = "DELETE FROM company WHERE name = 'AMAZON';";
79         Statement stmt = conn.createStatement();
80         stmt.executeUpdate(query);
81     }
82
83     // Method to query data
84     public ResultSet queryOne() throws SQLException {
85         String query = "SELECT * FROM company WHERE name = 'AMAZON';";
86         Statement stmt = conn.createStatement();
87         ResultSet rs = stmt.executeQuery(query);
88         return rs;
89     }
90
91     // Method to query data
92     public ArrayList<Company> queryAll() throws SQLException {
93         String query = "SELECT * FROM company";
94         Statement stmt = conn.createStatement();
95         ResultSet rs = stmt.executeQuery(query);
96         ArrayList<Company> companies = new ArrayList<Company>();
97         while (rs.next()) {
98             Company company = new Company();
99             company.setC_id(rs.getInt("c_id"));
100            company.setName(rs.getString("name"));
101            company.setTicker(rs.getString("ticker"));
102            company.setAnnualRevenue(rs.getBigDecimal("annualRevenue"));
103            company.setNumEmployees(rs.getInt("numEmployees"));
104            companies.add(company);
105        }
106        return companies;
107    }
108
109    // Method to query data
110    public ArrayList<StockPrice> queryStockPrice() throws SQLException {
111        String query = "SELECT * FROM stockprice";
112        Statement stmt = conn.createStatement();
113        ResultSet rs = stmt.executeQuery(query);
114        ArrayList<StockPrice> stockPrices = new ArrayList<StockPrice>();
115        while (rs.next()) {
116            StockPrice stockPrice = new StockPrice();
117            stockPrice.setS_id(rs.getInt("s_id"));
118            stockPrice.setC_id(rs.getInt("c_id"));
119            stockPrice.setDate(rs.getDate("date"));
120            stockPrice.setLowPrice(rs.getBigDecimal("lowPrice"));
121            stockPrice.setHighPrice(rs.getBigDecimal("highPrice"));
122            stockPrice.setClosePrice(rs.getBigDecimal("closePrice"));
123            stockPrice.setVolume(rs.getInt("volume"));
124            stockPrices.add(stockPrice);
125        }
126        return stockPrices;
127    }
128
129    // Method to query data
130    public ArrayList<Company> queryCompany() throws SQLException {
131        String query = "SELECT * FROM company";
132        Statement stmt = conn.createStatement();
133        ResultSet rs = stmt.executeQuery(query);
134        ArrayList<Company> companies = new ArrayList<Company>();
135        while (rs.next()) {
136            Company company = new Company();
137            company.setC_id(rs.getInt("c_id"));
138            company.setName(rs.getString("name"));
139            company.setTicker(rs.getString("ticker"));
140            company.setAnnualRevenue(rs.getBigDecimal("annualRevenue"));
141            company.setNumEmployees(rs.getInt("numEmployees"));
142            companies.add(company);
143        }
144        return companies;
145    }
146
147    // Method to query data
148    public ArrayList<StockPrice> queryStockPriceByCompanyID(int c_id) throws SQLException {
149        String query = "SELECT * FROM stockprice WHERE c_id = ?";
```

Verification on the MySQL Work Bench



The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** Contains the following SQL query:

```
1 SELECT c.name, c.ticker,
2        MIN(s.lowPrice) AS minPrice,
3        MAX(s.highPrice) AS maxPrice,
4        ROUND( AVG(s.closePrice), 2 ) AS avgClose,
5        ROUND( AVG(s.volume), 2 ) AS avgVolume
6
7        FROM company c
8        JOIN stockprice s ON c.c_id = s.companyId
9        WHERE s.priceDate BETWEEN '2022-08-22' AND '2022-08-26'
10       GROUP BY c.c_id
11       ORDER BY avgVolume DESC
```
- Result Grid:** Displays the results of the query, showing a single row for the company AMZN.

name	ticker	minPrice	maxPrice	avgClose	avgVolume
AMAZON	AMZN	163.56	171.05	167.20	61411420.00
- Information Schema:** Shows the structure of the company table.
- Object Info:** Shows the columns of the company table.
- Session:** Shows the execution history of the query.

QueryThree(): 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.

Script

```
public static void main(String[] args) {  
    SQLonRDS q = new SQLonRDS();  
  
    try {  
        q.connect();  
        q.drop();  
        q.create();  
        q.insert();  
        q.delete();  
        q.queryOne();  
        q.queryTwo();  
        q.queryThree();  
  
        //      q.close();  
    } catch (SQLException | ClassNotFoundException e) {  
        e.printStackTrace();  
    }  
  
    public ResultSet queryThree() throws SQLException {  
        Statement stmt = con.createStatement();  
        return stmt.executeQuery("""  
            SELECT c.name, c.ticker, s1.closePrice AS closeOn30  
            FROM company c  
            LEFT JOIN stockprice s1 ON c.id = s1.companyId AND s1.priceDate = '2022-08-30'  
            LEFT JOIN (  
                SELECT companyId, AVG(closePrice) AS avgClose  
                FROM stockprice  
            )  
            WHERE c.name IS NOT NULL  
            ORDER BY c.name  
        """);  
    }  
}
```

WHERE priceDate BETWEEN '2022-08-15' AND '2022-08-19'

GROUP BY companyId

) s2 ON c.id = s2.companyId

WHERE c.ticker IS NULL

OR (s1.closePrice IS NOT NULL AND s1.closePrice >= 0.9 * s2.avgClose)

ORDER BY c.name ASC

"");

}

On Script

The screenshot shows the Eclipse IDE interface with the SQLOnRDS.java file open in the editor. The code implements a query method that performs a complex join and filtering operation between the company and stockprice tables. The Java console below shows the execution of the script and the resulting output, which includes connection details, table creation, value insertion, and the successful execution of the query.

```
public ResultSet queryTwo() throws SQLException {
    public ResultSet queryOne() throws SQLException {
        Statement stmt = con.createStatement();
        return stmt.executeQuery(""
            + "SELECT c.name, c.ticker, s1.closePrice AS closeOn30
            FROM company c
            LEFT JOIN stockprice s1 ON c.id = s1.companyId AND s1.priceDate = '2022-08-30'
            GROUP BY companyId
            ) s2 ON c.id = s2.companyId
            WHERE c.ticker IS NULL
            OR (s1.closePrice IS NOT NULL AND s1.closePrice >= 0.9 * s2.avgClose)
            ORDER BY c.name ASC
        """);
}
```

```
<terminated> SQLOnRDS [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (Jun 23, 2025, 11:38:52 PM - 11:39:01 PM)
Connected to database.
Connection Successful.
Tables dropped successfully (if existed).
Tables created successfully.
Values inserted on the table successfully.
Given Values Deleted on the stockprice table successfully.
Given Values Selected on the company table successfully based on the conditions.
Given Values Selected and done the Joining operation on the company table and stockprice Table successfully based on the conditions.
```

Verification on the MySQL Work Bench

The screenshot shows the MySQL Workbench interface with a query editor containing the same SQL code as the Eclipse script. Below the editor is a result grid displaying the data from the company table. The bottom pane shows the execution history of the query, detailing the time taken for each step of the execution.

Action	Time	Message
1	11:37:48	SHOW DATABASES
2	11:37:48	5 rows(s) returned
3	22:32:35	Select * from company LIMIT 0, 1000
4	22:40:18	5 rows(s) returned
5	22:50:20	Select * from stockprice LIMIT 0, 1000
6	23:29:34	36 rows(s) returned
7	23:34:04	SELECT name, annualRevenue, numEmployees FROM company WHERE numEmployees > ... 4 rows(s) returned
8	23:42:41	MAX(s.highPrice) AS maxPrice, MIN(s.lowPrice) AS minPrice, LEFT JOIN stockp... 2 rows(s) returned

Conclusion

Through this assignment, students gain practical experience in integrating Java applications with cloud-based relational databases using JDBC. By designing and implementing structured queries and managing relational data, the exercise helps solidify concepts of SQL operations, schema design, and data analysis within a cloud environment. It also reinforces best practices in data manipulation and extraction essential for real-world analytics and financial data systems.

Git Link: <https://github.com/PurushothamanShanmugam/Big-Data>