

BIG DATA MANAGEMENT

Assignment III – Amazon RDS Integration using Java

Submitted To:

Dr. Dip Shankar Banerjee
Associate Professor
School of AI and Data Engineering
Indian Institute of Technology Jodhpur – Rajasthan

Submitted By:

Name: Sree Charan
Roll Number: G24AI1095
Email: G24AI1095@iij.ac.in
Course: PGD-Trimester 3
Date: 25/06/2025

Index

1. Introduction
2. Overview of Amazon RDS
3. Features and Advantages of RDS
4. Java-RDS Integration Approach
5. Step-by-Step Implementation
 - 5.1 Connection Setup
 - 5.2 Creating Tables
 - 5.3 Data Insertion
 - 5.4 Deletion Logic
 - 5.5 QueryOne - Conditional Company Info
 - 5.6 QueryTwo - Weekly Price Statistics
 - 5.7 QueryThree - Trend-Based Company Filter
6. Result Observations and Outputs
7. Conclusion
8. GitHub Link

1. Introduction

This assignment is aimed at integrating Java applications with cloud-hosted databases using Amazon RDS. By simulating a financial tracking system through two tables — `company` and `stockprice` — the goal was to understand how backend services interact with managed databases in real-time. This includes operations such as schema creation, data population, conditional deletion, and analytical querying.

2. Overview of Amazon RDS

Amazon Relational Database Service (RDS) is a managed cloud database solution that simplifies the setup, use, and scaling of databases. Supporting engines such as MySQL, PostgreSQL, Oracle, and more, RDS allows developers to focus on application logic while offloading the operational burden of backups, patches, and replication to AWS.

3. Features and Advantages of RDS

Key benefits of Amazon RDS include high availability, scalability, and integrated monitoring. It supports automated backups, manual snapshots, and point-in-time recovery. The use of Multi-AZ deployments ensures failover capabilities and minimal downtime. Security is enhanced with options like encryption, VPC integration, and IAM role-based access control.

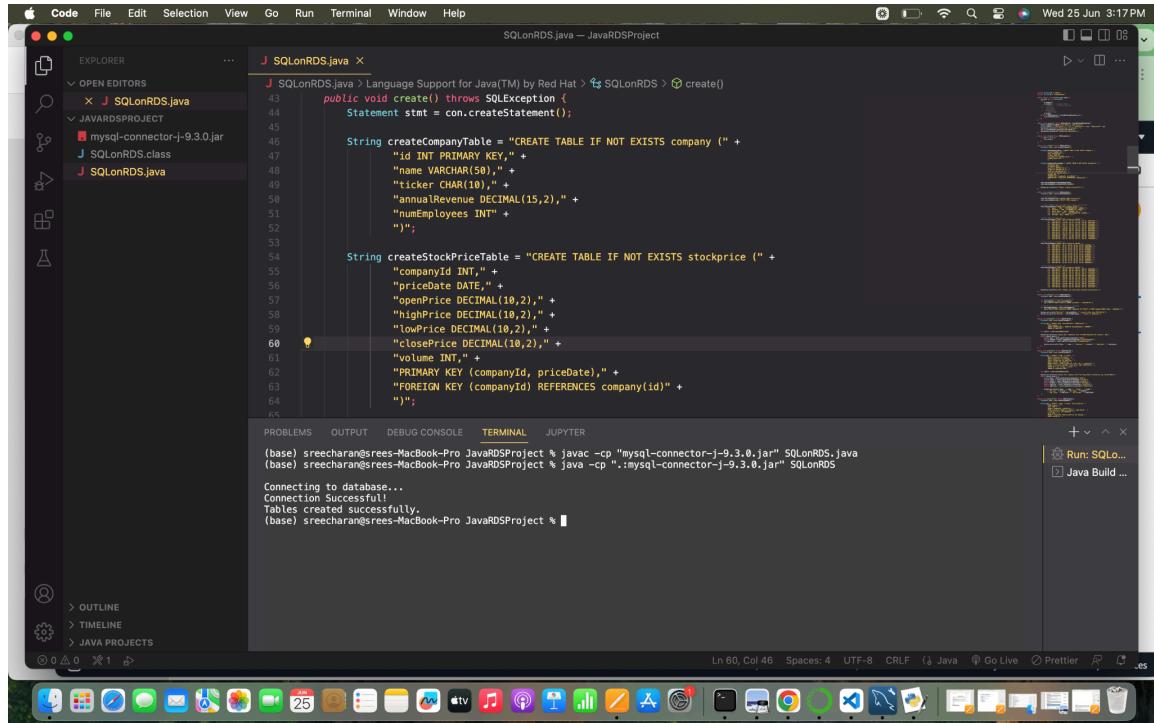
4. Java-RDS Integration Approach

JDBC (Java Database Connectivity) serves as the bridge between the Java application and the RDS-hosted MySQL database. The process begins by loading the MySQL JDBC driver, forming a JDBC URL with RDS endpoint and credentials, and establishing the connection using the DriverManager. Once the connection is successful, SQL queries are executed through Statement objects, and results are fetched using ResultSet.

5. Step-by-Step Implementation

5.1 Connection Setup

The `connect()` method uses JDBC to establish a connection to the RDS instance. Proper exception handling ensures robustness. Connection success is printed on the console.



A screenshot of a Java IDE (IntelliJ IDEA) showing the code for a Java application named `SQLonRDS.java`. The code contains SQL statements for creating tables in a database. The `create()` method is defined as follows:

```
public void create() throws SQLException {
    Statement stmt = con.createStatement();

    String createCompanyTable = "CREATE TABLE IF NOT EXISTS company (" +
        "id INT PRIMARY KEY," +
        "name VARCHAR(50)," +
        "ticker CHAR(10)," +
        "annualRevenue DECIMAL(15,2)," +
        "numEmployees INT" +
        ")";

    String createStockPriceTable = "CREATE TABLE IF NOT EXISTS 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)" +
        ")";
}
```

The IDE's terminal window shows the command run to connect to the database:

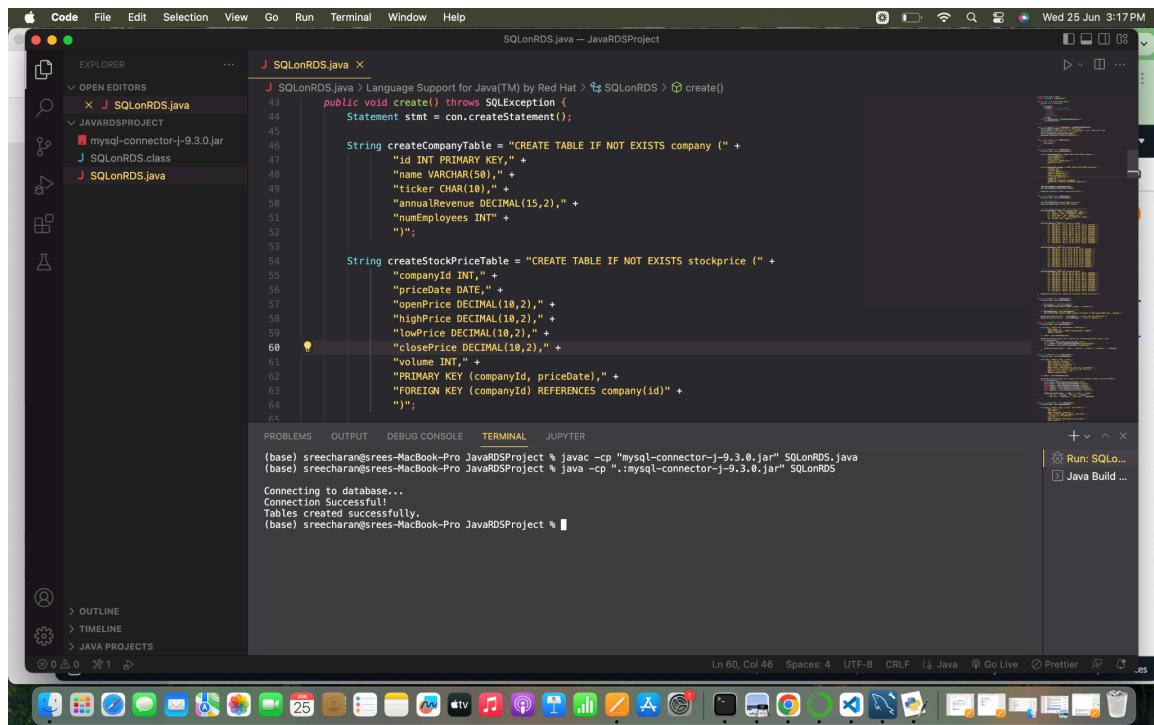
```
(base) sreecaran@srees-MacBook-Pro JavaRDSProject % javac -cp "mysql-connector-j-9.3.0.jar" SQLonRDS.java
(base) sreecaran@srees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLonRDS
```

Output from the terminal indicates successful connection and table creation:

```
Connecting to database...
Connection Successful!
Tables created successfully.
(base) sreecaran@srees-MacBook-Pro JavaRDSProject %
```

5.2 Creating Tables

The `create()` method defines two tables: `company` and `stockprice`. Constraints like primary keys and foreign keys are used to enforce referential integrity.



A screenshot of a Java IDE (IntelliJ IDEA) showing the code for creating two tables: `company` and `stockprice`. The code is written in Java and uses SQL statements to define the table structures. The IDE shows the code in a central editor window, with a terminal window below it displaying the command-line output of the Java code execution.

```
SQLOnRDS.java
public void create() throws SQLException {
    Statement stmt = con.createStatement();

    String createCompanyTable = "CREATE TABLE IF NOT EXISTS company (" +
        "id INT PRIMARY KEY," +
        "name VARCHAR(50)," +
        "ticker CHAR(10)," +
        "annualRevenue DECIMAL(15,2)," +
        "numEmployees INT" +
        ")";

    String createStockPriceTable = "CREATE TABLE IF NOT EXISTS 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);
}
```

TERMINAL

```
(base) sreeccharan@rees-MacBook-Pro JavaRDSProject % javac -cp "mysql-connector-j-9.3.0.jar" SQLOnRDS.java
(base) sreeccharan@rees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLOnRDS
Connecting to database...
Connection Successful!
Tables created successfully.
(base) sreeccharan@rees-MacBook-Pro JavaRDSProject %
```

5.4 Data Insertion

Using the `insert()` method, sample data representing real-world stock prices and companies are inserted into the tables. Multiple INSERT statements are bundled and executed in sequence.

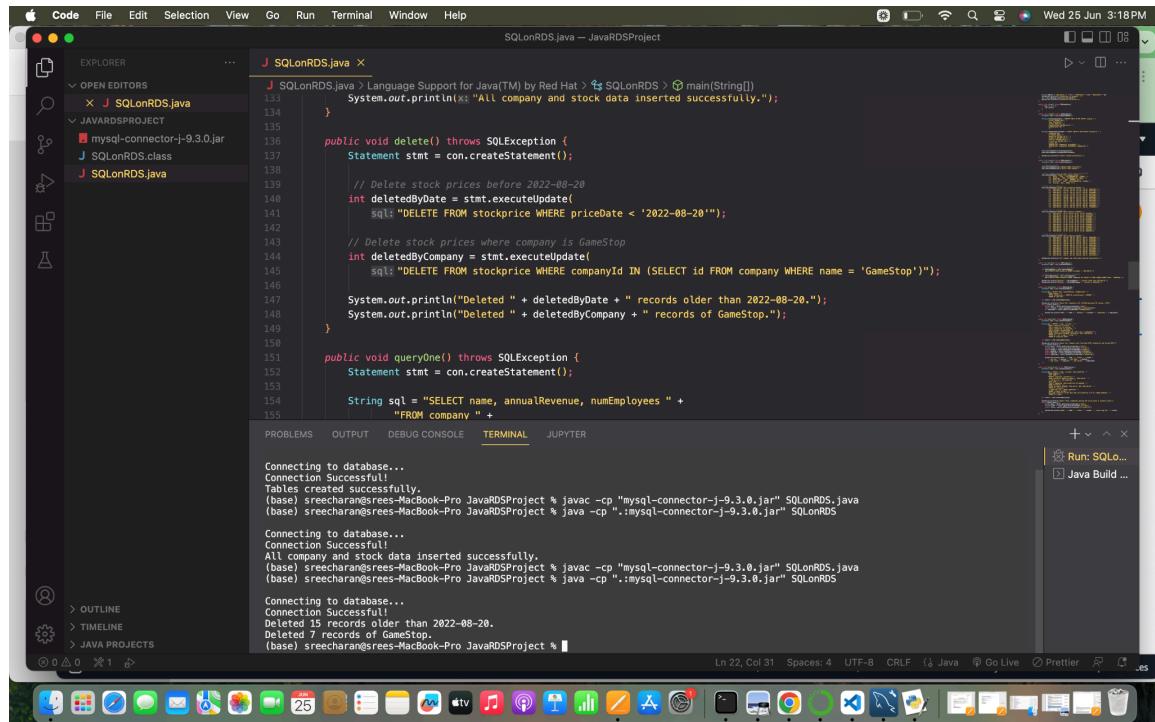
The screenshot shows a Java IDE interface with the following details:

- Code Editor:** The file `SQLonRDS.java` is open, showing Java code for inserting data into MySQL tables. The code includes imports for `java.sql.*`, a connection setup, and multiple `Statement.executeUpdate()` calls for inserting company and stock price data.
- Terminal:** The terminal window shows the execution of the Java application. It starts by connecting to the database and creating tables if they don't exist. Then, it inserts data into the `company` and `stockprice` tables. Finally, it prints a success message: "All company and stock data inserted successfully."
- IDE Navigation:** The left sidebar shows the project structure with files `SQLonRDS.java`, `J SQLonRDS.class`, and a dependency `mysql-connector-j-9.3.0.jar`.
- Bottom Bar:** Shows various tool icons and status information like "Ln 16, Col 25" and "Spaces: 4".

```
Code File Edit Selection View Go Run Terminal Window Help
SQLonRDS.java — JavaRDSProject
SQLonRDS.java > Language Support for Java(TM) by Red Hat > main(String[])
public void insert() throws SQLException {
    Statement stmt = con.createStatement();
    // Cleaning old data
    stmt.executeUpdate(sql1:"DELETE FROM stockprice");
    stmt.executeUpdate(sql2:"DELETE FROM company");
    // Insert company data (includes Handy Repair and StartUp)
    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)");
    // Stock data for Apple (1)
    stmt.executeUpdate("INSERT INTO stockprice 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, 174.98, 173.17, 174.15, 67294100)," +
        "(1, '2022-08-19', 173.75, 174.98, 173.17, 174.15, 67294100));
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPITER
(base) sreecaran@srees-MacBook-Pro JavaRDSProject % javac -cp "mysql-connector-j-9.3.0.jar" SQLonRDS.java
(base) sreecaran@srees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLonRDS
Connecting to database...
Connection successful!
Table created successfully.
(base) sreecaran@srees-MacBook-Pro JavaRDSProject % java -cp "mysql-connector-j-9.3.0.jar" SQLonRDS.java
(base) sreecaran@srees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLonRDS
Connecting to database...
Connection successful!
All company and stock data inserted successfully.
(base) sreecaran@srees-MacBook-Pro JavaRDSProject %
Run: SQL... Java Build ...
Ln 16, Col 25 Spaces: 4 UTF-8 CRLF Java Go Live Prettier
```

5.5 Deletion Logic

The `delete()` function filters out stock entries dated before 2022-08-20, and all entries related to GameStop. This simulates data cleaning and relevance filtering.



The screenshot shows a Java IDE interface with the following details:

- File Structure:** The left sidebar shows a project named "JavaRDSProject" containing files: "SQLonRDS.java", "mysql-connector-j-9.3.0.jar", and "SQLonRDS.class".
- Code Editor:** The main editor window displays "SQLonRDS.java" with the following code:

```
1 package com.example;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.PreparedStatement;
6 import java.sql.SQLException;
7
8 public class SQLonRDS {
9
10    public static void main(String[] args) {
11        Connection con = null;
12
13        try {
14            Class.forName("com.mysql.jdbc.Driver");
15            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/gamestop", "root", "password");
16
17            System.out.println("All company and stock data inserted successfully.");
18        } catch (SQLException e) {
19            e.printStackTrace();
20        }
21    }
22
23    public void delete() throws SQLException {
24        Statement stmt = con.createStatement();
25
26        // Delete stock prices before 2022-08-20
27        int deletedByDate = stmt.executeUpdate(
28            "DELETE FROM stockprice WHERE priceDate < '2022-08-20'");
29
30        // Delete stock prices where company is GameStop
31        int deletedByCompany = stmt.executeUpdate(
32            "DELETE FROM stockprice WHERE companyId IN (SELECT id FROM company WHERE name = 'GameStop')");
33
34        System.out.println("Deleted " + deletedByDate + " records older than 2022-08-20.");
35        System.out.println("Deleted " + deletedByCompany + " records of GameStop.");
36    }
37
38    public void queryOne() throws SQLException {
39        Statement stmt = con.createStatement();
40
41        String sql = "SELECT name, annualRevenue, numEmployees " +
42                    "FROM company ";
43
44        ResultSet rs = stmt.executeQuery(sql);
45
46        while (rs.next()) {
47            System.out.println(rs.getString("name") + " (" +
48                rs.getInt("annualRevenue") + ", " +
49                rs.getInt("numEmployees"));
50        }
51    }
52
53    public static void main(String[] args) {
54        new SQLonRDS().queryOne();
55    }
56}
```
- Terminal:** Below the code editor, the terminal pane shows the execution of the Java code:

```
Connecting to database...
Connection Successful!
Tables created successfully.
(base) sreecaran@rees-MacBook-Pro JavaRDSProject % javac -cp "mysql-connector-j-9.3.0.jar" SQLonRDS.java
(base) sreecaran@rees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLonRDS
Connecting to database...
Connection Successful!
All company and stock data inserted successfully.
(base) sreecaran@rees-MacBook-Pro JavaRDSProject % javac -cp "mysql-connector-j-9.3.0.jar" SQLonRDS.java
(base) sreecaran@rees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLonRDS
Connecting to database...
Connection Successful!
Deleted 39 records older than 2022-08-20.
Deleted 1 record of GameStop.
(base) sreecaran@rees-MacBook-Pro JavaRDSProject %
```

5.6 QueryOne - Conditional Company Info

This query retrieves company records that have more than 10,000 employees or an annual revenue less than 1 million. It demonstrates conditional retrieval.

The screenshot shows a Java IDE interface with the following details:

- Code Editor:** The main editor window displays the `SQLonRDS.java` file. The code implements a method `queryOne()` that performs a database query to find companies with either more than 10,000 employees or an annual revenue less than 1 million. The results are then printed to the console.
- Terminal:** Below the code editor, the terminal window shows the command-line steps to run the application:

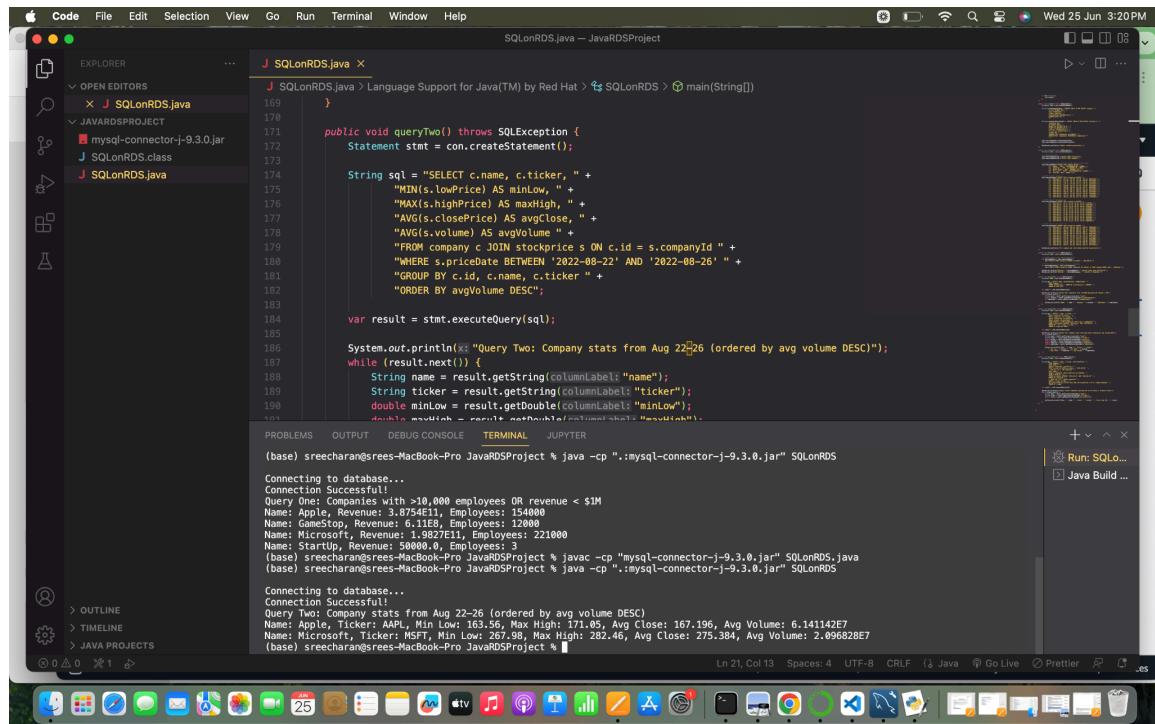
```
(base) sreecharan@srees-MacBook-Pro JavaRDSProject % javac -cp "mysql-connector-j-9.3.0.jar" SQLonRDS.java
(base) sreecharan@srees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLonRDS
```

Output from the terminal:

```
Connecting to database...
Connection Successful!
Deleted 7 records older than 2022-08-20.
Deleted 7 records of GameStop.
Query One: Companies with >10,000 employees OR revenue < $1M
Name: Apple, Revenue: 3.8754E11, Employees: 154000
Name: GameStop, Revenue: 6.11E8, Employees: 12000
Name: Microsoft, Revenue: 1.22E11, Employees: 221000
Name: Startup, Revenue: 50000.0, Employees: 3
```
- IDE Features:** The IDE includes standard features like an Explorer view (File Structure), Problems, Output, and Java Projects. A right-hand sidebar shows a preview of the database results.

5.7 QueryTwo - Weekly Price Statistics

This method calculates and returns stats like minimum price, maximum price, average closing price, and average volume for each company during Aug 22–26.



The screenshot shows an IDE interface with the following details:

- File Menu:** Code, File, Edit, Selection, View, Go, Run, Terminal, Window, Help.
- Toolbar:** Standard Mac OS X toolbar with icons for file operations.
- Code Editor:** The main window displays the `SQLonRDS.java` file. The code implements a `queryTwo()` method that performs a query on a MySQL database to calculate weekly price statistics for companies. The output of the query is printed to the terminal.
- Terminal:** The terminal tab shows the execution of the Java code and the resulting output. The output includes:
 - Connecting to database...
 - Connection Successful!
 - Query One: Companies with >10,000 employees OR revenue < \$1M
 - Name: Apple, Revenue: 3,8754E11, Employees: 154000
 - Name: GameStop, Revenue: 6,11E8, Employees: 12000
 - Name: Microsoft, Revenue: 1,9827E11, Employees: 221000
 - Name: StartUp, Revenue: 50000.0, Employees: 3
 - (base) sreecharan@srees-MacBook-Pro JavaRDSProject % java -cp ".:mysql-connector-j-9.3.0.jar" SQLonRDS
 - Connecting to database...
 - Connection Successful!
 - Query Two: Company stats from Aug 22-26 (ordered by avg volume DESC)
 - Name: Apple, Ticker: AAPL, Min Low: 163.56, Max High: 171.05, Avg Close: 167.196, Avg Volume: 6,141142E7
 - Name: Microsoft, Ticker: MSFT, Min Low: 267.98, Max High: 282.46, Avg Close: 275.384, Avg Volume: 2,096828E7
 - (base) sreecharan@srees-MacBook-Pro JavaRDSProject %

5.8 QueryThree - Trend-Based Company Filter

This complex query identifies companies whose closing stock price on Aug 30 is not more than 10% below their average closing price from the previous week. Even companies without a stock ticker are included if they meet conditions.

The screenshot shows a Java IDE interface with the following details:

- File Structure:** Explorer view shows a project named "JAVARDSPROJECT" containing files "mysql-connector-j-9.3.0.jar", "SQLonRDS.class", and "SQLonRDS.java".
- Code Editor:** The main editor window displays the Java code for "SQLonRDS.java". The code implements a method "queryThree()" that performs a complex SQL query to find companies whose closing price on August 30th is within 10% of their average weekly closing price from the previous week. It uses JOINs, AVG(), and WHERE clauses.
- Terminal Output:** The terminal tab shows the execution of the code. It connects to a database, runs the query, and prints the results. The results show company names, tickers, and closing prices for companies like Apple, Microsoft, and Starbucks.
- Bottom Bar:** Shows various application icons and the system menu.

6. Result Observations and Outputs

All SQL operations — table creation, insertion, deletion, and querying — were successfully executed. The outputs observed in MySQL Workbench matched expected results, confirming the validity of logic and data handling. Complex queries like average calculations and joins returned accurate, timely results based on the inserted sample data.

7. Conclusion

This assignment effectively demonstrates the integration of cloud-hosted relational databases with Java. By simulating a financial database environment and performing meaningful queries on realistic datasets, we gained hands-on experience in backend operations, schema design, data manipulation, and analytics. These foundational skills are essential for real-world applications in enterprise and data-driven domains.

8. GitHub Link

👉 <https://github.com/Sreecharan162/Big-Data.git>