

BIG DATA MANAGEMENT

POST GRADUATE DIPLOMA IN DATA ENGINEERING

ASSIGNMENT 3

SUBMITTED BY:

NIRAJ BHAGCHANDANI [G23AI2087]



SUBMISSION DATE: 10th December, 2024

**DEPARTMENT OF AIDE
INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR**

ASSIGNMENT - 3

Full Code:

BigDataAssign.java

```
import java.sql.*;

class BigDataAssign{
    private Connection connection;
    private String url = "jdbc:mysql://g23ai2087-instance.cojx0qdcaqlm.us-east-1.rds.amazonaws.com";
    //private String url = "jdbc:mysql://localhost/db_g23ai2087";
    private String user = "admin";
    private String password = "z8vm6IFzsMwcgSj0mtto";
    //private String user = "root";
    //private String password = "";
    private String db = "db_g23ai2087";
    private Statement stmt;
    private ResultSet rs;

    public void resultSetMetaDataToString(String table){
        String query = "SELECT table_schema, table_name, column_name, ordinal_position, data_type, numeric_precision, column_type, column_default, is_nullable, column_comment FROM information_schema.columns WHERE table_name = '"+table+"' order by ordinal_position";
        try{
            ResultSet rs = this.executeMe(query, "Query");
            this.print(rs, "Metadata: " + table);
        } catch (Exception e){
            System.out.println("Error: Generating Metadata - " + e.getMessage());
        }
    }

    public void queryOne(){
        String query = ""
            SELECT name, annualRevenue, numEmployees
            FROM company
            WHERE numEmployees > 10000 OR annualRevenue < 1000000
            ORDER BY name ASC
            "";
        try{
            this.rs = this.executeMe(query, "Query");
            this.print(rs, "Query One");
        } catch (Exception e){
            System.out.println("Error Executing Query: " + e.getMessage());
        }
    }

    public void queryTwo(){
        String query = ""
            SELECT c.name, c.ticker,
            MIN(s.lowPrice) AS lowestPrice,
            MAX(s.highPrice) AS highestPrice,
```

```

        AVG(s.closePrice) AS avgClosingPrice,
        AVG(s.volume) 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, c.name, c.ticker
    ORDER BY avgVolume DESC
    """;
}
try{
    this.rs = this.executeMe(query, "Query");
    this.print(rs, "Query Two");
}catch(Exception e){
    System.out.println("Error Executing Query: " + e.getMessage());
}
}
public void queryThree(){
    String query = """
    SELECT c.name, c.ticker, s.closePrice
    FROM company c
    LEFT JOIN stockprice s ON c.id = s.companyId
    AND s.priceDate = '2022-08-30'
    WHERE (s.closePrice IS NULL OR s.closePrice >= 0.9 * (
        SELECT AVG(closePrice)
        FROM stockprice
        WHERE priceDate BETWEEN '2022-08-15' AND '2022-08-19'
    ))
    ORDER BY c.name ASC;
    """;
    try{
        this.rs = this.executeMe(query, "Query");
        this.print(rs, "Query Two");
    }catch(Exception e){
        System.out.println("Error Executing Query: " + e.getMessage());
    }
}
public void createDatabase(){
    String sql = "CREATE DATABASE IF NOT EXISTS db_g23ai2087";
    this.executeMe(sql, "Update");
    System.out.println("createDatabase() executed successfully");
}

public ResultSet executeMe(String query, String flag){
    try{
        if(flag == "Update"){
            int status = this.stmt.executeUpdate(query);
        }else if(flag == "Query"){
            this.rs = this.stmt.executeQuery(query);
        }
    }catch (SQLException e) {
        System.out.println("Error Executing Query: " + e.getMessage());
    }
    return this.rs;
}

```

```

    }
    public void createTable(){
        String select = "USE "+db;
        String createCompany = ""
            CREATE TABLE company (
                id INT PRIMARY KEY,
                name VARCHAR(50),
                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) ON DELETE
CASCAD
            )
            "";
        try{
            this.executeMe(select, "Update");
            this.executeMe(createCompany, "Update");
            this.executeMe(createStockPrice, "Update");
            System.out.println("\nTable Created Successfully");
        }catch(Exception e){
            System.out.println("Error: Creating Database - "
+e.getMessage());
        }
    }

    public void dropTables(){
        try{
            String query = "DROP TABLE stockprice";
            this.executeMe(query, "Update");
            query = "DROP TABLE company";
            this.executeMe(query, "Update");
            System.out.println("Tables Deleted Successfully");
        }catch(Exception e){
            System.out.println("Error: Creating Database - " +e.getMessage());
        }
    }

    public void print(ResultSet rs, String table){
        int count = 0;
        String str = "";

```

```

        try{
            str +=
"\n=====\\n";
            ResultSetMetaData metaData = rs.getMetaData();
            int columnCount = metaData.getColumnCount();
            str += table + ": Columns Found: " +columnCount;
            str +=
"\n=====\\n";
            for(int i = 1; i <= columnCount; i++){
                str += metaData.getColumnName(i);
                if(i == columnCount) continue;
                str += ",";
            }
            str +=
"\n=====\\n";
            while(rs.next()){
                for(int i = 1; i <= columnCount; i++){
                    str += rs.getString(i);
                    if(i == columnCount) continue;
                    str += ",";
                }
                count++;
                str += "\\n";
            }
        }catch(Exception e){
            System.out.println("Error: Result Set to String Conversion: - "
+e.getMessage());
        }
        str +=
"=====\\n";
        str += "Records found in table: "+count+"\\n";
        str +=
"=====\\n";
        System.out.println(str);
    }
    public void resultSetToString(String table){
        String query = "SELECT * FROM "+table;
        try{
            ResultSet rs = this.executeMe(query,"Query");
            this.print(rs,"Table - "+table);
        }catch(Exception e){
            System.out.println("Error: Result Set to String Conversion: - "
+e.getMessage());
        }
    }
    public void dropDB(){
        try{
            String query = "DROP database "+ this.db;
            this.executeMe(query,"Update");
            System.out.println("Database Dropped Successfully");
        }catch(Exception e){
            System.out.println("Error: Creating Database - " +e.getMessage());

```

```

    }
}
public void insertData(){
    String company = ""
        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, 50),
        (4, 'Microsoft', 'MSFT', '198270000000.00' , 221000),
        (5, 'StartUp', null, 50000, 3)
    "";
    String stockprices = ""
        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{
        this.executeMe(company, "Update");
        this.executeMe(stockprices, "Update");
    }catch(Exception e){
        System.out.println("Error: Inserting Data: "+e.getMessage());
    }
}
public void connect(){
    try {
        Class.forName("com.mysql.jdbc.Driver");
        connection = DriverManager.getConnection(this.url, this.user,
this.password);
        System.out.println("Connected to the Server.");
        this.stmt = connection.createStatement();
    } catch (Exception e) {
        System.out.println("Error connecting to the database: " +
e.getMessage());
    }
}
public Connection getConnection(){
    return this.connection;
}
public Statement getStatement(){
    return this.stmt;
}

public void disconnect() {
    try {
        if (this.connection != null && !this.connection.isClosed()) {
            this.connection.close();
            System.out.println("Disconnected from the database.");
        }
    } catch (SQLException e) {
        System.out.println("Error disconnecting from the database: " +
e.getMessage());
    }
}
}
}

```

ExecuteAssignment.java

```

import java.util.Scanner;
import java.lang.*;

/*
D:\IIT - Jodhpur\BigData>javac -cp mysql.jar;. *.java
D:\IIT - Jodhpur\BigData>java -cp mysql.jar;. ExecuteAssignment
*/
class ExecuteAssignment{
    private BigDataAssign bda;

```

```

public BigDataAssign getBda(){
    return this.bda;
}

public ExecuteAssignment(){
    System.out.println("Hello World");
    this.bda = new BigDataAssign();
    bda.connect();
    bda.createDatabase();
}

public static void main(String args[]){
    ExecuteAssignment e = new ExecuteAssignment();

    Scanner scanner = new Scanner(System.in);
    boolean exit = false;
    while (!exit) {
        System.out.println("\nMenu:");
        System.out.println("1. Connect to Database");
        System.out.println("2. Create Database");
        System.out.println("3. Create Table");
        System.out.println("4. Insert into tables");
        System.out.println("5. queryOne");
        System.out.println("6. queryTwo");
        System.out.println("7. queryThree");
        System.out.println("8. Drop Tables");
        System.out.println("9. Delete Database");
        System.out.println("10. Result Set ToString");
        System.out.println("11. Result Set Meta Data To String");
        System.out.println("12. Exit");
        System.out.print("Enter your choice: ");

        int choice = scanner.nextInt();
        switch (choice) {
            case 1 :
                System.out.println("Connecting the Database...\n");
                e.bda.connect();
                break;
            case 2 :
                e.getBda().createDatabase();
                System.out.println("Database Created Successfully ...\n");
                break;
            case 3 :
                System.out.println("Creating the Tables...\n");
                e.getBda().createTable();
                break;
            case 4 :
                System.out.println("Inserting Data...");
                e.getBda().insertData();
                break;
            case 5 :
                System.out.println("Executing Simple One Query");
                e.getBda().queryOne();
                break;

```



```

        case 6 :
            System.out.println("Executing Simple Two Query");
            e.getBda().queryTwo();
            break;
        case 7 :
            System.out.println("Executing Simple Three Query");
            e.getBda().queryThree();
            break;
        case 8 :
            System.out.println("Drop Tables");
            e.getBda().dropTables();
            break;
        case 9 :
            System.out.println("Dropping Database");
            e.getBda().dropDB();
            break;
        case 10 :
            System.out.println("Result Set to Data String");
            e.getBda().resultSetToString("company");
            e.getBda().resultSetToString("stockprice");
            break;
        case 11 :
            e.getBda().resultSetMetaDataToString("company");
            e.getBda().resultSetMetaDataToString("stockprice");
            break;
        case 12 :
            exit = true;
            break;
        default :
            System.out.println("Invalid choice. Try again.");
            break;
    }
    //scanner.close();
}
}
}

```

Extend the Amazon RDS connector code in java to do the following in each of the respective functions.

Step 1: Choose a database creation method.

You can select either:

- **Standards create:** Allows detailed configuration of availability, security, backups, and maintenance.
- **Easy create:** Provides recommended best-practice configurations for a simplified setup.


Choose a database creation method


☒ **Standard create**
You set all of the configuration options, including ones for availability, security, backups, and maintenance.


☐ **Easy create**
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type [Info](#)

☐ Aurora (MySQL Compatible) 

☐ Aurora (PostgreSQL Compatible) 

☒ **MySQL** 


☐ PostgreSQL 

Fig.3.1 RDS database creation interface showing database creation methods (Standard create and Easy create) and engine options (Aurora MySQL/PostgreSQL compatible, MySQL, and PostgreSQL).

Step 2: Select the database edition and engine version.

- Choose the **MySQL Community** edition.
- Optionally, apply filters like Multi-AZ DB clusters or Amazon RDS Optimized Writes.
- Select the required **engine version** (e.g., MySQL 8.0.39).
- You can also enable **RDS Extended Support** if needed.

Edition

☒ **MySQL Community**

Engine version [Info](#)
View the engine versions that support the following database features.

▼ **Hide filters**

☒ **Show only versions that support the Multi-AZ DB cluster** [Info](#)
Create a Multi-AZ DB cluster with one primary DB instance and two readable standby DB instances. Multi-AZ DB clusters provide up to 2x faster transaction commit latency and automatic failover in typically under 35 seconds.

☒ **Show only versions that support the Amazon RDS Optimized Writes** [Info](#)
Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

Engine version

MySQL 8.0.39 ▼

☐ **Enable RDS Extended Support** [Info](#)
Amazon RDS Extended Support is a [paid offering](#). By selecting this option, you consent to being charged for this offering if you are running your database major version past the RDS end of standard support date for that version. Check the end of standard support date for your major version in the [RDS for MySQL documentation](#).

Fig.3.2 Configuration options for selecting the database edition, engine version, filters, and enabling RDS Extended Support in the RDS database creation process.

Step 3: Select a template (Production, Dev/Test, or Free Tier) and configure deployment options (Multi-AZ DB Cluster, Multi-AZ DB Instance, or Single DB Instance) based on availability and durability needs.

The screenshot shows the 'Create database' page in the AWS RDS console. It is divided into two main sections: 'Templates' and 'Availability and durability'.

Templates: This section asks the user to 'Choose a sample template to meet your use case.' There are three options:

- Production:** Use defaults for high availability and fast, consistent performance.
- Dev/Test:** This instance is intended for development use outside of a production environment.
- Free tier:** Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS. (This option is selected with a blue border and a radio button).

Availability and durability: This section shows 'Deployment options' with a note: 'The deployment options below are limited to those supported by the engine you selected above.' There are three options:

- Multi-AZ DB Cluster:** Creates a DB cluster with a primary DB instance and two readable standby DB instances, with each DB instance in a different Availability Zone (AZ). Provides high availability, data redundancy and increases capacity to serve read workloads.
- Multi-AZ DB instance (not supported for Multi-AZ DB cluster snapshot):** Creates a primary DB instance and a standby DB instance in a different AZ. Provides high availability and data redundancy, but the standby DB instance doesn't support connections for read workloads.
- Single DB instance (not supported for Multi-AZ DB cluster snapshot):** Creates a single DB instance with no standby DB instances. (This option is selected with a radio button).

Fig.3.3 Template and deployment option configurations in the RDS database creation process.

Step 4: Configure the database settings by specifying the DB instance identifier, master username, and password. Choose credentials management as either **Self-managed** or **Managed in AWS Secrets**

The screenshot shows the 'Settings' page in the AWS RDS console. It contains the following configuration fields and options:

- DB instance identifier:** A text field containing 'g23ai2087-instance'. A note below states: 'The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 63 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.'
- Credentials Settings:**
 - Master username:** A text field containing 'admin'. A note below states: 'Type a login ID for the master user of your DB instance. 1 to 16 alphanumeric characters. The first character must be a letter.'
 - Credentials management:**
 - Managed in AWS Secrets Manager - most secure:** RDS generates a password for you and manages it throughout its lifecycle using AWS Secrets Manager. (This option is selected with a radio button).
 - Self managed:** Create your own password or have RDS create a password that you manage. (This option is also selected with a radio button).
 - Auto generate password:** Amazon RDS can generate a password for you, or you can specify your own password. (This option is unchecked).

Manager.

Fig. 3.4 Settings configuration for DB instance identifier, master username, and credentials management in the RDS database creation process.

Step 5: Set the master password and confirm it, ensuring it meets the required constraints. Configure the instance settings by selecting the DB instance class and optional features such as Amazon RDS Optimized Writes.

The screenshot displays the 'Create database' page in the Amazon RDS console. The 'Master password' field is highlighted with a red border and a red error message: 'The Master password is invalid. Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / ' " @'. The 'Password strength' indicator shows 'Very strong'. The 'Confirm master password' field is also visible. Below the password fields, the 'Instance configuration' section is shown, which includes a 'DB instance class' dropdown and a 'Hide filters' button. Two filters are listed: 'Show instance classes that support Amazon RDS Optimized Writes' (which is selected) and 'Include previous generation classes' (which is not selected). An 'Info' link is provided for the first filter, stating that Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

Fig.3.5 Configuration of master password and DB instance class with optional settings for Amazon RDS Optimized Writes in the RDS database creation process.

Step 6: Configure instance type and storage. Select the DB instance class (e.g., db.t4g.micro) and define storage settings, including storage type (e.g., General Purpose SSD) and allocated storage size.

Fig.3.6 Configuration of DB instance type and storage settings, including storage type and allocated storage

The screenshot shows the 'Create database' page in the AWS RDS console. The instance type is set to 'db.t4g.micro' (2 vCPUs, 1 GiB RAM, Network: Up to 2,085 Mbps). Under the 'Storage' section, the storage type is 'General Purpose SSD (gp2)' with a baseline performance determined by volume size. The allocated storage is set to '20' GiB. A note states: 'Allocated storage value must be 20 GiB to 6,144 GiB'. There is a link for 'Additional storage configuration'.

size, in the RDS database creation process

Step 7: Configure connectivity by selecting the VPC, DB subnet group, and whether to connect to an EC2 compute resource.

The screenshot shows the 'Connectivity' section of the 'Create database' page. Under 'Compute resource', there are two radio buttons: 'Don't connect to an EC2 compute resource' (selected) and 'Connect to an EC2 compute resource'. Under 'Virtual private cloud (VPC)', the 'Default VPC (vpc-08a4598365eb86770)' is selected. A note states: 'Only VPCs with a corresponding DB subnet group are listed.' Below this, a message says: 'After a database is created, you can't change its VPC.' Under 'DB subnet group', the 'default-vpc-08a4598365eb86770' is selected. A note states: 'Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.'

Fig.3.7 Connectivity settings for VPC, DB subnet group, and EC2 compute resource in the RDS database creation process.

Step 8: Set public access, select or create a VPC security group, and choose an Availability Zone.

The screenshot shows the 'Create database' page in the Amazon RDS console. The 'Public access' section has the 'Yes' radio button selected, indicating that a public IP address will be assigned. The 'VPC security group (firewall)' section has the 'Choose existing' radio button selected. Below this, a dropdown menu for 'Existing VPC security groups' shows 'default' as the selected option. The 'Availability Zone' section has a dropdown menu with 'No preference' selected.

Fig.3.8 Public access and VPC security group settings in RDS database setup.

Step 9: Configure additional settings by specifying the database name, parameter group, option group, and enabling automated backups.

The screenshot shows the 'Additional configuration' section of the Amazon RDS console. It includes the following settings:

- Database options:** Initial database name is 'db_g23ai2087'.
- DB parameter group:** Set to 'default.mysql8.0'.
- Option group:** Set to 'default:mysql-8-0'.
- Backup:** The 'Enable automated backups' checkbox is checked, with a note that it 'Creates a point-in-time snapshot of your database'.

Fig. 3.9 Additional configuration for database options and backup settings in the RDS database setup.

Step 10: Set backup and encryption options. Configure the backup retention period, window, and enable encryption with an AWS KMS key.

The screenshot shows the 'Create database' page in the AWS RDS console. At the top, there is a warning message: 'Please note that automated backups are currently supported for InnoDB storage engine only. If you are using MyISAM, refer to details [here](#).' Below this, the 'Backup retention period' is set to 1 day. The 'Backup window' is set to 'No preference'. The 'Copy tags to snapshots' checkbox is checked. The 'Backup replication' section has the 'Enable replication in another AWS Region' checkbox unchecked. The 'Encryption' section has the 'Enable encryption' checkbox checked. The 'AWS KMS key' is set to '(default) aws/rds'.

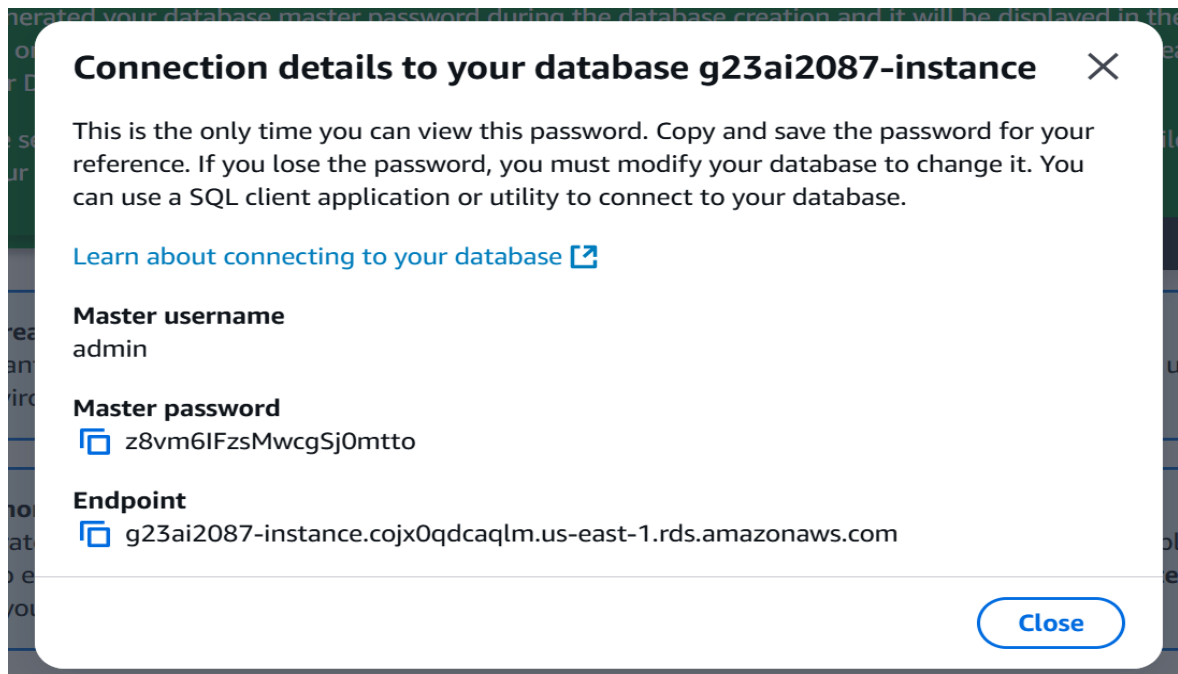
Fig.3.10 Backup and encryption settings, including retention period, backup replication, and encryption options in the RDS database setup.

Step 11: Review the estimated monthly costs, enable deletion protection if needed, and click **Create database** to finalize the setup.

The screenshot shows the 'Create database' page in the AWS RDS console, focusing on the 'Deletion protection' and 'Estimated monthly costs' sections. The 'Deletion protection' section has the 'Enable deletion protection' checkbox unchecked. The 'Estimated monthly costs' section lists the free tier benefits: 750 hrs of Amazon RDS in a Single-AZ db.t2.micro, db.t3.micro or db.t4g.micro Instance, 20 GB of General Purpose Storage (SSD), and 20 GB for automated backup storage and any user-initiated DB Snapshots. At the bottom, there is a disclaimer: 'You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.' The 'Create database' button is highlighted in orange.

Fig.3.11 Final review of estimated monthly costs and deletion protection before creating the RDS database.

Step 12: Save the connection details, including the master username, password, and endpoint. This information is displayed only once and is required to connect to your database.



3.12 Connection details, including username, password, and endpoint, for the newly created RDS database instance.

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

```
public void connect(){
    try {
        Class.forName("com.mysql.jdbc.Driver");
        connection = DriverManager.getConnection(this.url, this.user,
this.password);
        System.out.println("Connected to the Server.");
        this.stmt = connection.createStatement();
    } catch (Exception e) {
        System.out.println("Error connecting to the database: " +
e.getMessage());
    }
}
```



```

D:\iitj\tri-3\BDM\iitj-bdm-assignment\Assignment - 3>javac -cp mysql.jar;. *.java

D:\iitj\tri-3\BDM\iitj-bdm-assignment\Assignment - 3>java -cp mysql.jar;. ExecuteAssignment
Hello World
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.
Connected to the Server.
createDatabase() executed successfully

Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 1
Connecting the Database...

Connected to the Server.

```

Fig3.13 Command prompt displaying database operations menu and successful execution of a Java program for MySQL interactions.

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

```

public void dropTables(){
    try{
        String query = "DROP TABLE stockprice";
        this.executeMe(query,"Update");
        query = "DROP TABLE company";
        this.executeMe(query,"Update");
        System.out.println("Tables Deleted Successfully");
    }catch(Exception e){
        System.out.println("Error: Creating Database - " +e.getMessage());
    }
}

```

```

D:\iitj\tri-3\BDM\iitj-bdm-assignment\Assignment - 3>java -cp mysql.jar;. ExecuteAssignment
Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 8
Drop Tables
Tables Deleted Successfully

```

Fig.3.14 Command prompt showing successful deletion of tables using a Java program for MySQL database operations.

```

java
Command Prompt - java -cp mysql.jar, ExecuteAssignment

Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 9
Dropping Database
Database Dropped Successfully

```

Fig.3.15 Command prompt showing successful database deletion using a Java program for MySQL database operations.

3. create() : creates the following table in the database: [10] * Creates the table in the database. * Table name: company * Fields: * - id - integer, must be primary key * - name - variable character field up to size 50 * - ticker - character field always of size 10 * - annualRevenue - must hold up to 999,999,999.99 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 createTable(){
    String createCompany = ""
        CREATE TABLE company (
            id INT PRIMARY KEY,
            name VARCHAR(50),
            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) ON DELETE CASCADE
        )
        "";
}

```

```

try{
    this.executeMe(createCompany,"Update");
    this.executeMe(createStockPrice,"Update");
    System.out.println("\nTable Created Successfully");
}catch(Exception e){
    System.out.println("Error: Creating Database - " +e.getMessage());
}
}

```

```

Command Prompt - java -cp mysql.jar;. ExecuteAssignment
Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 2
createDatabase() executed successfully
Database Created Successfully ...

```

Fig.3.16 Command prompt showing successful database creation using a Java program for MySQL database operations.

```

Command Prompt - java -cp mysql.jar;. ExecuteAssignment
Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 3
Creating the Tables...
Table Created Successfully

```

Fig.3.17 Command prompt showing successful table creation using a Java program for MySQL database operations.

4. insert() : Inserts test records in the database: [10] * 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

Code:

```
public void insertData(){
    String company = ""
    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, 50),
    (4, 'Microsoft', 'MSFT', '198270000000.00' , 221000),
    (5, 'StartUp', null, 50000, 3)
    "";
    String stockprices = ""
    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{
    this.executeMe(company,"Update");
    this.executeMe(stockprices,"Update");
}catch(Exception e){
    System.out.println("Error: Inserting Data: "+e.getMessage());
}
}

```

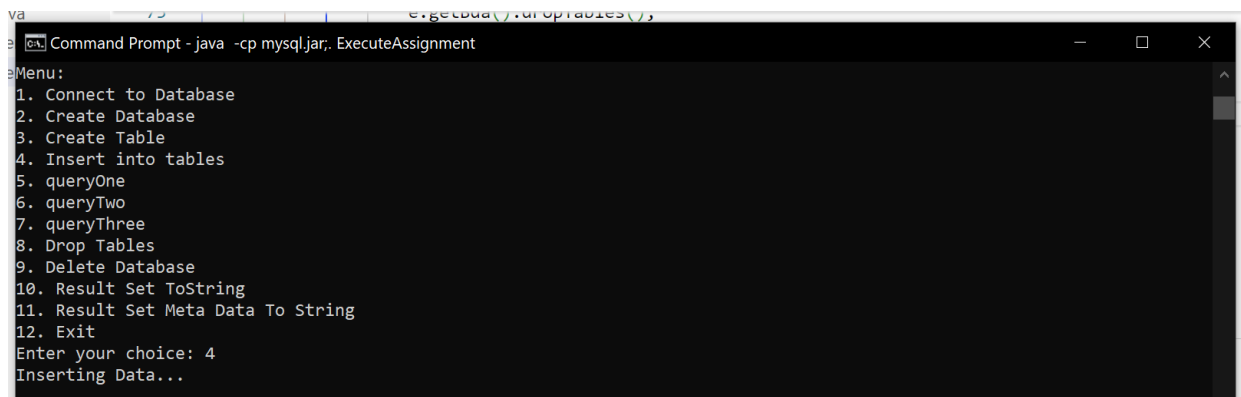


Fig.3.18 Command prompt showing the initiation of data insertion into tables using a Java program for MySQL database operations.

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

```

public void dropTables(){
    try{
        String query = "DROP TABLE stockprice";
        this.executeMe(query,"Update");
        query = "DROP TABLE company";
        this.executeMe(query,"Update");
    }
}

```

```

        System.out.println("Tables Deleted Successfully");
    } catch (Exception e) {
        System.out.println("Error: Creating Database - " + e.getMessage());
    }
}

```

```

java
Command Prompt - java -cp mysql.jar, ExecuteAssignment
Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 8
Drop Tables
Tables Deleted Successfully

```

Fig.3.19 Command prompt showing successful deletion of tables using a Java program for MySQL database management.

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(){
    String query = ""
        SELECT name, annualRevenue, numEmployees
        FROM company
        WHERE numEmployees > 10000 OR annualRevenue < 1000000
        ORDER BY name ASC
        "";
    try{
        this.rs = this.executeMe(query, "Query");
        this.print(rs, "Query One");
    } catch (Exception e) {
        System.out.println("Error Executing Query: " + e.getMessage());
    }
}

```

```

C:\Program Files\Java\jdk-11.0.10\bin> java -cp mysql.jar;. ExecuteAssignment
Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 5
Executing Simple One Query

=====
Query One: Columns Found: 3
=====
name,annualRevenue,numEmployees
=====
Apple,38754000000.00,154000
GameStop,611000000.00,12000
Microsoft,19827000000.00,221000
Startup,50000.00,3
=====
Records found in table: 4
=====

```

Fig.3.20 Command prompt displaying the execution of a query to retrieve and display table data, including company names, annual revenue, and number of employees.

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(){
    String query = ""
        SELECT c.name, c.ticker,
            MIN(s.lowPrice) AS lowestPrice,
            MAX(s.highPrice) AS highestPrice,
            AVG(s.closePrice) AS avgClosingPrice,
            AVG(s.volume) 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, c.name, c.ticker
        ORDER BY avgVolume DESC
        "";
    try{
        this.rs = this.executeMe(query,"Query");
        this.print(rs, "Query Two");
    }catch(Exception e){
        System.out.println("Error Executing Query: " + e.getMessage());
    }
}

```

```

e.g. getBoard().getProperties();
e. Command Prompt - java -cp mysql.jar, ExecuteAssignment
Enter your choice: 6
Executing Simple Two Query

=====
Query Two: Columns Found: 6
=====
name,ticker,lowestPrice,highestPrice,avgClosingPrice,avgVolume
=====
Apple,AAPL,163.56,171.05,167.196000,61411420.0000
Microsoft,MSFT,267.98,282.46,275.384000,20968280.0000
GameStop,GME,30.63,36.20,32.686000,5054200.0000
=====
Records found in table: 3
=====

```

Fig.3.21 Command prompt displaying the execution of a query to retrieve stock data, including company name, ticker, price details, and average trading volume.

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(){
    String query = ""
    SELECT c.name, c.ticker, s.closePrice
    FROM company c
    LEFT JOIN stockprice s ON c.id = s.companyId
    AND s.priceDate = '2022-08-30'
    WHERE (s.closePrice IS NULL OR s.closePrice >= 0.9 * (
        SELECT AVG(closePrice)
        FROM stockprice
        WHERE priceDate BETWEEN '2022-08-15' AND '2022-08-19'
    ))
    ORDER BY c.name ASC;
    "";
    try{
        this.rs = this.executeMe(query, "Query");
        this.print(rs, "Query Two");
    }catch(Exception e){
        System.out.println("Error Executing Query: " + e.getMessage());
    }
}

```



```

Command Prompt - java -cp mysql.jar, ExecuteAssignment
Executing Simple Three Query

=====
Query Two: Columns Found: 3
=====
name,ticker,closePrice
=====
Apple,AAPL,158.91
Handy Repair,null,null
Microsoft,MSFT,262.97
StartUp,null,null
=====
Records found in table: 4
=====

```

Fig.3.22 Command prompt displaying the execution of a query retrieving company names, tickers, and closing prices, with some missing values in the dataset.

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

```

public void resultSetToString(String table){
    String query = "SELECT * FROM "+table;
    try{
        ResultSet rs = this.executeMe(query,"Query");
        this.print(rs,"Table - "+table);
    }catch(Exception e){
        System.out.println("Error: Result Set to String Conversion: - "
+e.getMessage());
    }
}

```

```

Command Prompt - java -cp mysql.jar, ExecuteAssignment
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 10
Result Set to Data String

=====
Table - company: Columns Found: 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,19827000000.00,221000
5,StartUp,null,50000.00,3
=====
Records found in table: 5
=====

```

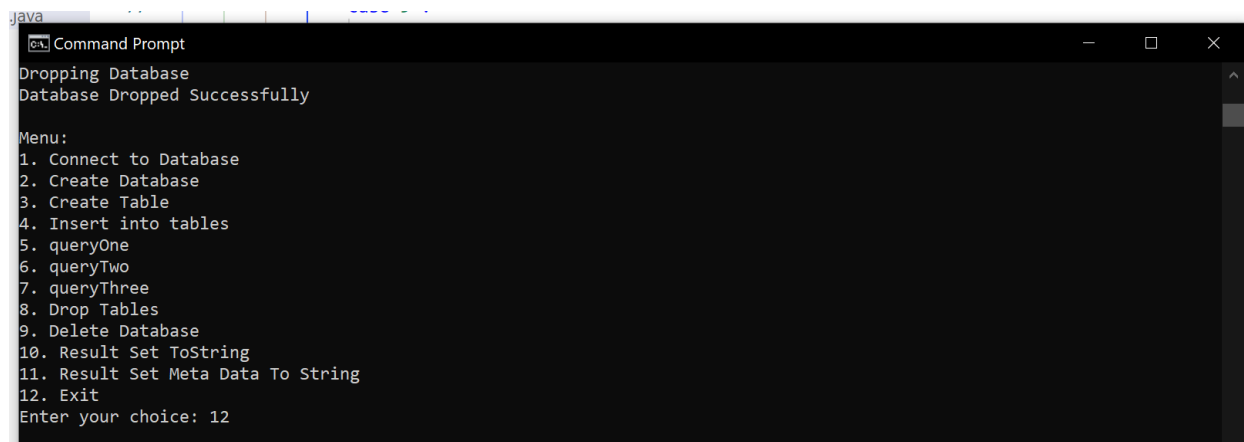
Fig.3.23 Command prompt displaying the result set converted to a data string, showing company details including ID, name, ticker, annual revenue, and number of employees.

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

```
public void resultSetMetaDataToString(String table){
    String query = "SELECT table_schema, table_name, column_name,
ordinal_position, data_type,numeric_precision, column_type, column_default,
is_nullable, column_comment FROM information_schema.columns WHERE table_name =
'"+table+"' order by ordinal_position";
    try{
        ResultSet rs = this.executeMe(query,"Query");
        this.print(rs,"Metadata: "+table);
    }catch(Exception e){
        System.out.println("Error: Generating Metadata - " +e.getMessage());
    }
}
```

```
=====
tcompanyId,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
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
=====
Records found in table: 36
```

Fig.3.24 Command prompt displaying table data with stock price details, including company ID, date, open price, high price, low price, close price, and trading volume.



```
java
C:\> Command Prompt
Dropping Database
Database Dropped Successfully

Menu:
1. Connect to Database
2. Create Database
3. Create Table
4. Insert into tables
5. queryOne
6. queryTwo
7. queryThree
8. Drop Tables
9. Delete Database
10. Result Set ToString
11. Result Set Meta Data To String
12. Exit
Enter your choice: 12
```

Fig.3.25 Command prompt showing the successful deletion of a database and the program menu with the option to exit.