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

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());

        }

    }

1. 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());

        }

    }

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

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());

            }

    }

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());

        }

    }

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());

        }

    }

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

public void queryOne(){

        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());

        }

    }

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());

        }

    }

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());

        }

    }

9. resultSetToString(): converts a ResultSet obtained front he 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());

        }

    }

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());

        }

    }