Write a sample program to demonstrate java.sql.ResultSetMetaData

Solution Code:

```
import java.sql.*;

public class ResultSetMetaData1 {
    public static void main(String[] args) {
        // Database connection details (replace with your actual database
credentials)
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";  // Your MySQL username
        String password = "Ishan@1234";  // Your MySQL password

        // SQL query to execute
        String query = "SELECT * FROM employees";  // Replace "yourtable"
with your actual table name

        // Declare connection, statement, and result set objects
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        try {
            // 1. Establish a connection to the database
            connection = DriverManager.getConnection(url, username,
password);

            // 2. Create a statement object to execute the query
            statement = connection.createStatement();

            // 3. Execute the query and obtain the result set
            resultSet = statement.executeQuery(query);

            // 4. Get ResultSetMetaData from the result set
            ResultSetMetaData rsMetaData = resultSet.getMetaData();

            // 5. Get the number of columns in the ResultSet
            int columnCount = rsMetaData.getColumnCount();
            System.out.println("Number of columns: " + columnCount);
```

```java
            // 6. Loop through each column and print metadata information
            for (int i = 1; i <= columnCount; i++) {
                // Get column name
                String columnName = rsMetaData.getColumnName(i);
                // Get column type (SQL type code)
                int columnType = rsMetaData.getColumnType(i);
                // Get column type name (e.g., VARCHAR, INT)
                String columnTypeName = rsMetaData.getColumnTypeName(i);
                // Get the size of the column
                int columnSize = rsMetaData.getColumnDisplaySize(i);

                // Print the metadata for each column
                System.out.println("Column " + i + ": " + columnName);
                System.out.println("  Type: " + columnTypeName + " (SQL
Type Code: " + columnType + ")");
                System.out.println("  Size: " + columnSize + "
characters");
            }

        } catch (SQLException e) {
            // Handle SQL exceptions
            e.printStackTrace();
        } finally {
            try {
                // 7. Close the resources to avoid memory leaks
                if (resultSet != null) resultSet.close();
                if (statement != null) statement.close();
                if (connection != null) connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Step by Step Explanation:

1. Importing Required Classes

```
import java.sql.*;
```

- **Explanation**: This line imports all the necessary classes from the `java.sql` package that are required for handling database connections, statements, queries, and result sets. It includes classes such as `Connection`, `Statement`, `ResultSet`, `SQLException`, and `ResultSetMetaData`.

2. Class Declaration

```
public class ResultSetMetaData1 {
```

- **Explanation**: This is the class declaration. The class `ResultSetMetaData1` is the main class of the program that contains the code for connecting to a MySQL database, executing a query, and working with `ResultSetMetaData`.

3. Main Method Declaration

```
public static void main(String[] args) {
```

- **Explanation**: This is the main entry point of the Java application. The `main` method is executed when the program is run, and inside it, we perform all the operations for database connection, query execution, and metadata retrieval.

4. Database Connection Details

```
String url = "jdbc:mysql://localhost:3306/mydatabase";
String username = "root";  // Your MySQL username
String password = "Ishan@1234";  // Your MySQL password
```

- **Explanation**: These lines define the connection details to your MySQL database:
  - `url`: The connection URL specifies the location of the database (in this case, the `localhost` server, port `3306`, and the database name `mydatabase`).
  - `username`: Your MySQL database username (in this case, it's `root`).
  - `password`: Your MySQL password (here, it's `Ishan@1234`).
- These values will be used to authenticate the connection to the MySQL database.

5. SQL Query

```
String query = "SELECT * FROM employees";
```

- **Explanation**: This line specifies the SQL query you want to execute. Here, the query `SELECT * FROM employees` fetches all columns and all rows from the `employees` table in the database.

6. Declare Connection, Statement, and ResultSet Objects

```
Connection connection = null;
Statement statement = null;
ResultSet resultSet = null;
```

- **Explanation**: These lines declare three objects:
  - `Connection`: Represents the database connection.
  - `Statement`: Used to execute SQL queries.
  - `ResultSet`: Holds the results returned by the SQL query.

7. Try Block to Establish Database Connection and Execute Query

```
try {
   // Establish a connection to the database
   connection = DriverManager.getConnection(url, username, password);
```

- **Explanation**: Inside the `try` block, the connection to the database is established using the `DriverManager.getConnection()` method, passing in the `url`, `username`, and `password` values. This connects your program to the MySQL database.

8. Create Statement and Execute Query

```
  statement = connection.createStatement();
   resultSet = statement.executeQuery(query);
```

- **Explanation**:
  - `statement = connection.createStatement();`: Creates a `Statement` object that allows you to execute SQL queries against the database.
  - `resultSet = statement.executeQuery(query);`: Executes the SQL query (`SELECT * FROM employees`) and stores the result in a `ResultSet` object.

9. Get ResultSetMetaData

```
  ResultSetMetaData rsMetaData = resultSet.getMetaData();
```

- **Explanation**: The `getMetaData()` method is called on the `ResultSet` object to retrieve metadata about the result set. The returned `ResultSetMetaData` object contains information about the columns in the result set, such as column names, types, sizes, etc.

10. Get the Number of Columns in the ResultSet

```
int columnCount = rsMetaData.getColumnCount();
 System.out.println("Number of columns: " + columnCount);
```

- **Explanation**: The `getColumnCount()` method returns the number of columns in the result set. This value is then printed to the console to let the user know how many columns are in the result set.

11. Loop Through Each Column and Print Metadata

```
for (int i = 1; i <= columnCount; i++) {
    // Get column name
    String columnName = rsMetaData.getColumnName(i);
    // Get column type (SQL type code)
    int columnType = rsMetaData.getColumnType(i);
    // Get column type name (e.g., VARCHAR, INT)
    String columnTypeName = rsMetaData.getColumnTypeName(i);
    // Get the size of the column
    int columnSize = rsMetaData.getColumnDisplaySize(i);

    // Print the metadata for each column
    System.out.println("Column " + i + ": " + columnName);
     System.out.println("  Type: " + columnTypeName + " (SQL Type Code: " + columnType +
")");
    System.out.println("  Size: " + columnSize + " characters");
 }
```

- **Explanation**:
  - This is a `for` loop that runs from 1 to `columnCount` (inclusive), which represents the total number of columns in the result set.
  - Inside the loop, several methods are called on the `ResultSetMetaData` object to retrieve metadata for each column:
    - `getColumnName(i)`: Returns the name of the column at index `i`.
    - `getColumnType(i)`: Returns the SQL type code of the column (e.g., `Types.INTEGER`, `Types.VARCHAR`).

        - `getColumnTypeName(i)`: Returns the type name of the column (e.g., `INT`, `VARCHAR`).
        - `getColumnDisplaySize(i)`: Returns the display size (maximum length) of the column.
    - These values are printed to the console for each column, showing the column name, type, and size.

12. Catch SQL Exceptions

```
} catch (SQLException e) {
    // Handle SQL exceptions
    e.printStackTrace();
}
```

- **Explanation**: If there is an error while connecting to the database or executing the query, it will throw a `SQLException`. This `catch` block catches and prints the stack trace of the exception.

13. Close Database Resources in the Finally Block

```
finally {
    try {
        // Close the resources to avoid memory leaks
        if (resultSet != null) resultSet.close();
        if (statement != null) statement.close();
        if (connection != null) connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- **Explanation**: The `finally` block ensures that all database resources (e.g., `ResultSet`, `Statement`, and `Connection`) are properly closed, even if an exception occurs. This prevents resource leaks and ensures that connections to the database are properly released.

Summary:

This Java program connects to a MySQL database, executes a SQL query to retrieve all rows from the `employees` table, and then uses `ResultSetMetaData` to extract and print metadata about each column in the result set. The program outputs:

- The number of columns in the result set.

- The name, type, and size of each column.

By doing this, you can gain insight into the structure of the data returned by your SQL query.