JDBC Day 3: ResultSet, DAO Pattern, layered architecture (Presentation layer, service layer, Data access layer)

ResultSet

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A `ResultSet` object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The `next` method moves the cursor to the next row, and because it returns `false` when there are no more rows in the `ResultSet` object, it can be used in a `while` loop to iterate through the result set.

A default `ResultSet` object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce `ResultSet` objects that are scrollable and/or updatable. The following code fragment, in which `con` is a valid `Connection` object, illustrates how to make a result set that is scrollable and insensitive to updates by others, and that is updatable. See `ResultSet` fields for other options.

```
       Statement stmt = con.createStatement();
       ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
```

The

```
ResultSet
```

interface provides

getter

methods (

```
getBoolean
```

,

```
getLong
```

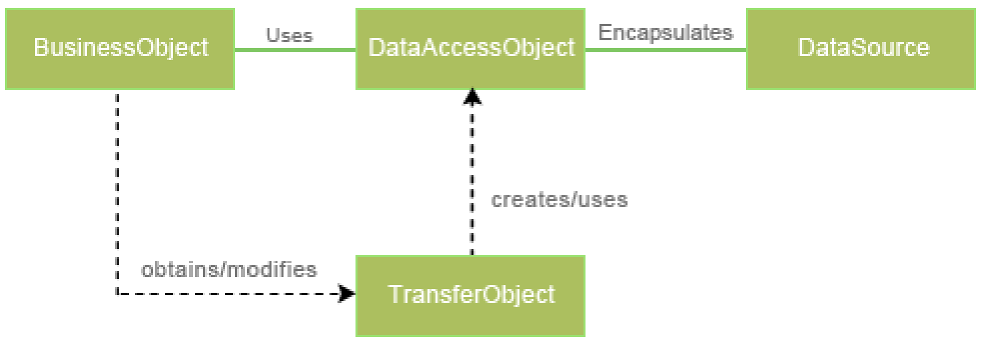
, and so on) for retrieving column values from the current row. Values can be retrieved using either the index number of the column or the name of the column. In general, using the column index will be more efficient. Columns are numbered from 1. For maximum portability, result set columns within each row should be read in left-to-right order, and each column should be read only once.

For the getter methods, a JDBC driver attempts to convert the underlying data to the Java type specified in the getter method and returns a suitable Java value. The JDBC specification has a table showing the allowable mappings from SQL types to Java types that can be used by the `ResultSet` getter methods.

Column names used as input to getter methods are case insensitive. When a getter method is called with a column name and several columns have the same name, the value of the first matching column will be returned. The column name option is designed to be used when column names are used in the SQL query that generated the result set. For columns that are NOT explicitly named in the query, it is best to use column numbers. If column names are used, the programmer should take care to guarantee that they uniquely refer to the intended columns, which can be assured with the SQL *AS* clause.

Data Access Object Pattern

Data Access Layer has proven good in separate business logic layer and persistent layer. The DAO design pattern completely hides the data access implementation from its clients. The interfaces given to client does not changes when the underlying data source mechanism changes. this is the capability which allows the DAO to adopt different access scheme without affecting to business logic or its clients. generally it acts as a adapter between its components and database. The DAO design pattern consists of some factory classes, DAO interfaces and some DAO classes to implement those interfaces.



The Data Access object is the primary object of this design pattern. This object abstract the data access implementations for the other object to enable transparently access to the database.

An example given below which illustrates the Data Access Design Pattern.

At first create table named student in MySql database and inset values into it as.

At first create table named student in MySql database and inset values into it as.

CREATE TABLE student (RollNo int(9) PRIMARY KEY NOT NULL, Name tinytext NOT NULL, Course varchar(25) NOT NULL, Address text);

ConnectionFactory.java

```
package roseindia.net;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionFactory {
  String driverClassName = "com.mysql.jdbc.Driver";
  String connectionUrl = "jdbc:mysql://localhost:3306/student";
  String dbUser = "root";
  String dbPwd = "root";

  private static ConnectionFactory connectionFactory = null;

  private ConnectionFactory() {
    try {
      Class.forName(driverClassName);
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  public Connection getConnection() throws SQLException {
    Connection conn = null;
    conn = DriverManager.getConnection(connectionUrl, dbUser, dbPwd);
    return conn;
  }

  public static ConnectionFactory getInstance() {
    if (connectionFactory == null) {
      connectionFactory = new ConnectionFactory();
    }
    return connectionFactory;
  }
}
```

StudentBean.java

```
package roseindia.net;

import java.io.Serializable;

public class StudentBean implements Serializable {
  int rollNo;
  String name;
  String course;
  String address;

  public StudentBean() {

  }

  public StudentBean(int roll, String name, String course, String address) {
    this.rollNo = roll;
    this.name = name;
    this.course = course;
    this.address = address;
  }

  public int getRollNo() {
    return rollNo;
  }

  public void setRollNo(int rollNo) {
    this.rollNo = rollNo;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getCourse() {
    return course;
  }
  public void setCourse(String course) {
    this.course = course;
  }

  public String getAddress() {
    return address;
  }

  public void setAddress(String address) {
    this.address = address;
  }

}
```

StudentJDBCDAO.java

```
package roseindia.net;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class StudentJDBCDAO {
  Connection connection = null;
  PreparedStatement ptmt = null;
  ResultSet resultSet = null;

  public StudentJDBCDAO() {

  }

  private Connection getConnection() throws SQLException {
    Connection conn;
    conn = ConnectionFactory.getInstance().getConnection();
    return conn;
  }

  public void add(StudentBean studentBean) {
    try {
      String queryString = "INSERT INTO student(RollNo, Name, Course, Address) VALUES
(?,?,?,?)";
      connection = getConnection();
      ptmt = connection.prepareStatement(queryString);
      ptmt.setInt(1, studentBean.getRollNo());
      ptmt.setString(2, studentBean.getName());
      ptmt.setString(3, studentBean.getCourse());
      ptmt.setString(4, studentBean.getAddress());
      ptmt.executeUpdate();
      System.out.println("Data Added Successfully");
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      try {
        if (ptmt != null)
          ptmt.close();
        if (connection != null)
          connection.close();
      } catch (SQLException e) {
        e.printStackTrace();
      } catch (Exception e) {
        e.printStackTrace();
      }

    }

  }

  public void update(StudentBean studentBean) {

    try {
      String queryString = "UPDATE student SET Name=? WHERE RollNo=?";
      connection = getConnection();
      ptmt = connection.prepareStatement(queryString);
      ptmt.setString(1, studentBean.getName());
      ptmt.setInt(2, studentBean.getRollNo());
      ptmt.executeUpdate();
      System.out.println("Table Updated Successfully");
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      try {
        if (ptmt != null)
          ptmt.close();
        if (connection != null)
          connection.close();
      }

      catch (SQLException e) {
        e.printStackTrace();
      } catch (Exception e) {
        e.printStackTrace();

      }
    }
  }

  public void delete(int rollNo) {

    try {
      String queryString = "DELETE FROM student WHERE RollNo=?";
      connection = getConnection();
      ptmt = connection.prepareStatement(queryString);
      ptmt.setInt(1, rollNo);
      ptmt.executeUpdate();
      System.out.println("Data deleted Successfully");
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      try {
        if (ptmt != null)
          ptmt.close();
        if (connection != null)
          connection.close();
      } catch (SQLException e) {
        e.printStackTrace();
      } catch (Exception e) {
        e.printStackTrace();
      }

    }

  }

  public void findAll() {
    try {
      String queryString = "SELECT * FROM student";
      connection = getConnection();
      ptmt = connection.prepareStatement(queryString);
      resultSet = ptmt.executeQuery();
      while (resultSet.next()) {
        System.out.println("Roll No " + resultSet.getInt("RollNo")
            + ", Name " + resultSet.getString("Name") + ", Course "
            + resultSet.getString("Course") + ", Address "
            + resultSet.getString("Address"));
      }
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      try {
        if (resultSet != null)
          resultSet.close();
        if (ptmt != null)
          ptmt.close();
        if (connection != null)
          connection.close();
      } catch (SQLException e) {
        e.printStackTrace();
      } catch (Exception e) {
        e.printStackTrace();
      }

    }
  }
}
```

MainClaz.java

```
package roseindia.net;

public class MainClaz {
  public static void main(String[] args) {
    StudentJDBCDAO student = new StudentJDBCDAO();
    StudentBean alok = new StudentBean();
    alok.setName("Alok");
    alok.setRollNo(8);
    alok.setCourse("MBA");
```

```
      alok.setAddress("Ranchi");
      StudentBean tinkoo = new StudentBean();
      tinkoo.setName("Arvind");
      tinkoo.setRollNo(6);
      // Adding Data
      student.add(alok);
      // Deleting Data
      student.delete(7);
      // Updating Data
      student.update(tinkoo);
      // Displaying Data
      student.findAll();
    }
  }
```

When you run this application it will display message as shown below:

```
 Data Added Successfully
Data deleted Successfully
Table Updated Successfully
Roll No 1, Name Java, Course MCA, Address Motihari
Roll No 2, Name Ravi, Course BCA, Address Patna
Roll No 3, Name Mansukh, Course  M.Sc , Address Katihar
Roll No 4, Name Raman, Course B.Tech, Address Betiah
Roll No 5, Name Kanhaiya, Course M.Tech, Address Delhi
Roll No 6, Name Arvind, Course MBA, Address Alligarh
Roll No 8, Name Alok, Course MBA, Address Ranchi
```

References:

https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html
https://www.javatpoint.com/java-jdbc

https://www.baeldung.com/java-statement-preparedstatement

https://www.roseindia.net/tutorial/java/jdbc/dataaccessobjectdesignpattern.html

https://www.oracle.com/java/technologies/data-access-object.html