How to develop a JDBC Form Load Binder

In this tutorial, we will be following the guideline for developing a plugin to develop our JDBC Form Load Binder plugin. Please also refer to this tutorial, How to develop a Bean Shell Hash Variable and also a JDBC related plugin How to develop a JDBC Options Binder for more details steps.

1. What is the problem?

For integration purpose, we would like to load our form data from a different database table instead of Joget form data table.

2. How to solve the problem?

Joget Workflow has provided a plugin type called Form Load Binder Plugin. We will refer to the former to develop a plugin to support JDBC connection and custom query to load form data.

3. What is the input needed for your plugin?

To develop a JDBC Load binder, we will need the JDBC connection setting and also the custom query to populate the form data based on primary key or foreign key (for Grid).

1. Datasource: Using custom datasource or Joget default datasource
2. Custom JDBC Driver: The JDBC driver for custom datasource
3. Custom JDBC URL: The JDBC connection URL for custom datasource
4. Custom JDBC Username: The username for custom datasource
5. Custom JDBC Password: The password for custom datasource
6. SQL Query: The query to populate form data.

The query should also support a syntax to inject the primary key (For Form/Section) or foreign key (For Grid element)

Example:

1. SELECT * from app_fd_sample where id = ?

4. What is the output and expected outcome of your plugin?

All retrieved columns are auto mapped to the fields with column name equal to field id.

5. Are there any resources/API that can be reused?

We can refer to the implementation of other available Form Load Binder plugins. Joget default datasource can be retrieve with AppUtil.getApplicationContext().getBean("setupDataSource").

6. Prepare your development environment

We need to always have our Joget Workflow Source Code ready and builded by following this guideline.

The following tutorial is prepared with a Macbook Pro and Joget Source Code version 5.0.0. Please refer to Guideline for developing a plugin for other platform command.

Let say our folder directory is as follows.

```
- Home
  - joget
    - plugins
    - jw-community
      -5.0.0
```

The "plugins" directory is the folder we will create and store all our plugins and the "jw-community" directory is where the Joget Workflow Source code stored.

Run the following command to create a maven project in "plugins" directory.

```
cd joget/plugins/
~/joget/jw-community/5.0.0/wflow-plugin-archetype/create-plugin.sh org.joget.tutorial jdbc_load_binder 5.0.0
```

Then, the shell script will ask us to key in a version for your plugin and ask us for confirmation before generate the maven project.

```
Define value for property 'version':  1.0-SNAPSHOT: : 5.0.0
[INFO] Using property: package = org.joget.tutorial
Confirm properties configuration:
groupId: org.joget.tutorial
artifactId: jdbc_load_binder
version: 5.0.0
package: org.joget.tutorial
Y: : y
```

We should get "BUILD SUCCESS" message shown in our terminal and a "jdbc_load_binder" folder created in "plugins" folder.

Open the maven project with your favour IDE. I will be using NetBeans.

7. Just code it!

a. Extending the abstract class of a plugin type

Create a "JdbcLoadBinder" class under "org.joget.tutorial" package. Then, extend the class with **org.joget.apps.form.model.FormBinder** abstract class.

To make it work as a Form Load Binder, we will need to implement **org.joget.apps.form.model.FormLoadBinder** interface. Then, we need to implement **org.joget.apps.form.model.FormLoadElementBinder** interface to make this plugin show as a selection in load binder select box and implement **org.joget.apps.form.model.FormLoadMultiRowElementBinder** interface to list it under the load binder select box of grid element.

Please refer to Form Load Binder Plugin.

b. Implement all the abstract methods

As usual, we have to implement all the abstract methods. We will using AppPluginUtil.getMessage method to support i18n and using constant variable MESSAGE_PATH for message resource bundle directory.

Implementation of all basic abstract methods

```
package org.joget.tutorial;

import org.joget.apps.app.service.AppPluginUtil;
import org.joget.apps.app.service.AppUtil;
import org.joget.apps.form.model.Element;
import org.joget.apps.form.model.FormBinder;
import org.joget.apps.form.model.FormData;
import org.joget.apps.form.model.FormLoadBinder;
import org.joget.apps.form.model.FormLoadElementBinder;
import org.joget.apps.form.model.FormLoadMultiRowElementBinder;
import org.joget.apps.form.model.FormRowSet;

public class JdbcLoadBinder extends FormBinder implements FormLoadBinder, FormLoadElementBinder,
FormLoadMultiRowElementBinder {

    private final static String MESSAGE_PATH = "messages/JdbcLoadBinder";

    public String getName() {
        return "JDBC Load Binder";
    }

    public String getVersion() {
        return "5.0.0";
    }

    public String getClassName() {
        return getClass().getName();
    }

    public String getLabel() {
        //support i18n
        return AppPluginUtil.getMessage("org.joget.tutorial.JdbcLoadBinder.pluginLabel", getClassName(),
MESSAGE_PATH);
    }

    public String getDescription() {
        //support i18n
        return AppPluginUtil.getMessage("org.joget.tutorial.JdbcLoadBinder.pluginDesc", getClassName(),
MESSAGE_PATH);
    }

    public String getPropertyOptions() {
        return AppUtil.readPluginResource(getClassName(), "/properties/jdbcLoadBinder.json", null, true,
MESSAGE_PATH);
    }

    public FormRowSet load(Element element, String primaryKey, FormData formData) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods,
choose Tools | Templates.
    }
}
```

Then, we have to do a UI for admin user to provide inputs for our plugin. In getPropertyOptions method, we already specify our Plugin Properties Options definition file is locate at "/properties/jdbcLoadBinder.json". Let us create a directory "resources/properties" under "jdbc_load_binder/src/main" directory. After create the directory, create a file named "jdbcLoadBinder.json" in the "properties" folder.

In the properties definition options file, we will need to provide options as below. Please note that we can use "@@message.key@@" syntax to support i18n in our properties options.

```
[{
    title : '@@form.jdbcLoadBinder.config@@',
    properties : [{
        name : 'jdbcDatasource',
        label : '@@form.jdbcLoadBinder.datasource@@',
        type : 'selectbox',
        options : [{
            value : 'custom',
            label : '@@form.jdbcLoadBinder.customDatasource@@'
        },{
            value : 'default',
            label : '@@form.jdbcLoadBinder.defaultDatasource@@'
        }],
        value : 'default'
    },{
        name : 'jdbcDriver',
        label : '@@form.jdbcLoadBinder.driver@@',
        description : '@@form.jdbcLoadBinder.driver.desc@@',
        type : 'textfield',
        value : 'com.mysql.jdbc.Driver',
        control_field: 'jdbcDatasource',
        control_value: 'custom',
        control_use_regex: 'false',
        required : 'true'
    },{
        name : 'jdbcUrl',
        label : '@@form.jdbcLoadBinder.url@@',
        type : 'textfield',
        value : 'jdbc:mysql://localhost/jwdb?characterEncoding=UTF8',
        control_field: 'jdbcDatasource',
        control_value: 'custom',
        control_use_regex: 'false',
        required : 'true'
    },{
        name : 'jdbcUser',
        label : '@@form.jdbcLoadBinder.username@@',
        type : 'textfield',
        control_field: 'jdbcDatasource',
        control_value: 'custom',
        control_use_regex: 'false',
        value : 'root',
        required : 'true'
    },{
        name : 'jdbcPassword',
        label : '@@form.jdbcLoadBinder.password@@',
        type : 'password',
        control_field: 'jdbcDatasource',
        control_value: 'custom',
        control_use_regex: 'false',
        value : ''
    },{
        name : 'sql',
        label : '@@form.jdbcLoadBinder.sql@@',
        description : '@@form.jdbcLoadBinder.sql.desc@@',
        type : 'codeeditor',
        mode : 'sql',
        required : 'true'
    }],
    buttons : [{
        name : 'testConnection',
        label : '@@form.jdbcLoadBinder.testConnection@@',
        ajax_url : '[CONTEXT_PATH]/web/json/app[APP_PATH]/plugin/org.joget.tutorial.JdbcLoadBinder/service?
action=testConnection',
        fields : ['jdbcDriver', 'jdbcUrl', 'jdbcUser', 'jdbcPassword'],
        control_field: 'jdbcDatasource',
        control_value: 'custom',
        control_use_regex: 'false'
    }]
}]
```

Same as JDBC Options Binder, we will need to add a test connection button for custom JDBC setting. Please refer to How to develop a JDBC Options Binder on the Web Service Plugin implementation.

Once we done the properties option to collect input and the web service to test the connection, we can work on the main method of the plugin which is load method.

```
public FormRowSet load(Element element, String primaryKey, FormData formData) {
    FormRowSet rows = new FormRowSet();
    rows.setMultiRow(true);
    //Check the sql. If require primary key and primary key is null, return empty result.
    String sql = getPropertyString("sql");
    if ((primaryKey == null || primaryKey.isEmpty()) && sql.contains("?")) {
        return rows;
    }

    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;

    try {
        DataSource ds = createDataSource();
        con = ds.getConnection();

        pstmt = con.prepareStatement(sql);

        //set query parameters
        if (sql.contains("?") && primaryKey != null && !primaryKey.isEmpty()) {
            pstmt.setObject(1, primaryKey);
        }

        rs = pstmt.executeQuery();
        ResultSetMetaData rsmd = rs.getMetaData();
        int columnsNumber = rsmd.getColumnCount();

        // Set retrieved result to Form Row Set
        while (rs.next()) {
            FormRow row = new FormRow();

            //get the name of each column as field id
            for (int i = 1; i <= columnsNumber; i++) {
                String name = rsmd.getColumnName(i);
                String value = rs.getString(name);

                if (FormUtil.PROPERTY_ID.equals(name)) {
                    row.setId(value);
                } else {
                    row.setProperty(name, (value != null)?value:"");
                }
            }

            rows.add(row);
        }
    } catch (Exception e) {
        LogUtil.error(getClassName(), e, "");
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (pstmt != null) {
                pstmt.close();
            }
            if (con != null) {
                con.close();
            }
        } catch (Exception e) {
            LogUtil.error(getClassName(), e, "");
        }
    }

    return rows;
}
```

```
    /**
     * To creates data source based on setting
     * @return
     * @throws Exception
     */
    protected DataSource createDataSource() throws Exception {
        DataSource ds = null;
        String datasource = getPropertyString("jdbcDatasource");
        if ("default".equals(datasource)) {
            // use current datasource
             ds = (DataSource)AppUtil.getApplicationContext().getBean("setupDataSource");
        } else {
            // use custom datasource
            Properties dsProps = new Properties();
            dsProps.put("driverClassName", getPropertyString("jdbcDriver"));
            dsProps.put("url", getPropertyString("jdbcUrl"));
            dsProps.put("username", getPropertyString("jdbcUser"));
            dsProps.put("password", getPropertyString("jdbcPassword"));
            ds = BasicDataSourceFactory.createDataSource(dsProps);
        }
        return ds;
    }
```

c. Manage the dependency libraries of your plugin

Our plugin is using dbcp, javax.servlet.http.HttpServletRequest and javax.servlet.http.HttpServletResponse class, so we will need to add jsp-api and commons-dbcp library to our POM file.

```
<!-- Change plugin specific dependencies here -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.0</version>
</dependency>
<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.3</version>
</dependency>
<!-- End change plugin specific dependencies here -->
```

d. Make your plugin internationalization (i18n) ready

We are using i18n message key in getLabel and getDescription method. We also used i18n message key in our properties options definition as well. So, we will need to create a message resource bundle properties file for our plugin.

Create directory "resources/messages" under "jdbc_load_binder/src/main" directory. Then, create a "JdbcLoadBinder.properties" file in the folder. In the properties file, let add all the message keys and its label as below.

```
org.joget.tutorial.JdbcLoadBinder.pluginLabel=JDBC Binder
org.joget.tutorial.JdbcLoadBinder.pluginDesc=Used to load form data using JDBC
form.jdbcLoadBinder.config=Configure JDBC Binder
form.jdbcLoadBinder.datasource=Datasource
form.jdbcLoadBinder.customDatasource=Custom Datasource
form.jdbcLoadBinder.defaultDatasource=Default Datasource
form.jdbcLoadBinder.driver=Custom JDBC Driver
form.jdbcLoadBinder.driver.desc=Eg. com.mysql.jdbc.Driver (MySQL), oracle.jdbc.driver.OracleDriver (Oracle),
com.microsoft.sqlserver.jdbc.SQLServerDriver (Microsoft SQL Server)
form.jdbcLoadBinder.url=Custom JDBC URL
form.jdbcLoadBinder.username=Custom JDBC Username
form.jdbcLoadBinder.password=Custom JDBC Password
form.jdbcLoadBinder.sql=SQL SELECT Query
form.jdbcLoadBinder.sql.desc=Use question mark (?) in your query to represent primary key or foreign key
form.jdbcLoadBinder.testConnection=Test Connection
form.jdbcLoadBinder.connectionOk=Database connected
form.jdbcLoadBinder.connectionFail=Not able to establish connection.
```

e. Register your plugin to Felix Framework

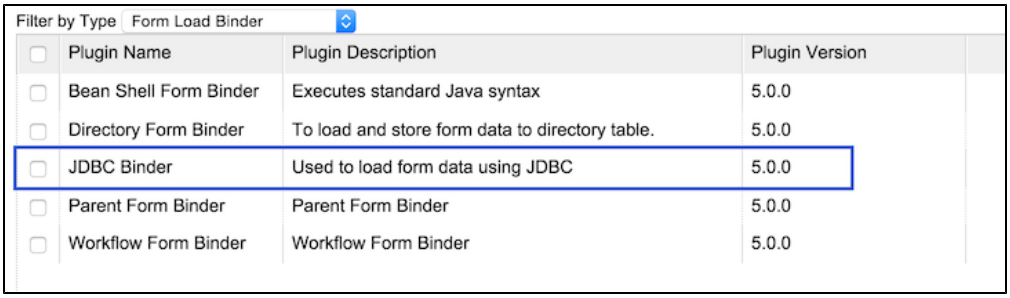
We will have to register our plugin class in Activator class (Auto generated in the same class package) to tell Felix Framework that this is a plugin.

```
    public void start(BundleContext context) {
        registrationList = new ArrayList<ServiceRegistration>();
        //Register plugin here
        registrationList.add(context.registerService(JdbcLoadBinder.class.getName(), new JdbcLoadBinder(),
null));
    }
```

f. Build it and testing

Let build our plugin. Once the building process is done, we will found a "jdbc_load_binder-5.0.0.jar" file is created under "jdbc_load_binder/target" directory.

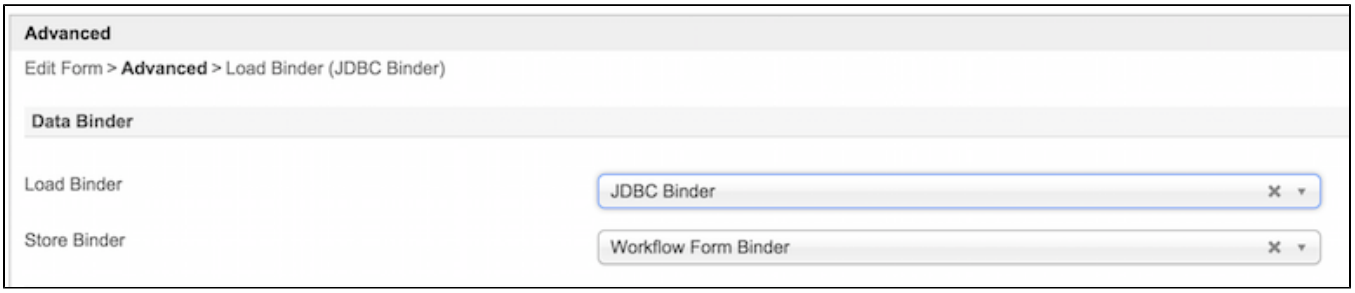
Then, let upload the plugin jar to Manage Plugins. After upload the jar file, double check the plugin is uploaded and activated correctly.

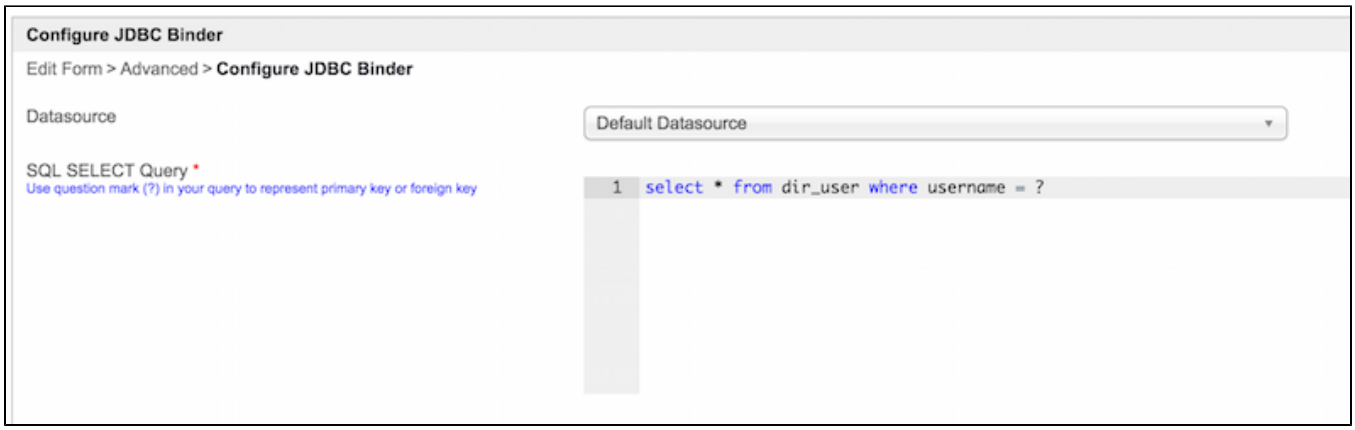


Let create a form to load firstName, lastName and email from dir_user table based on username to test the load binder.

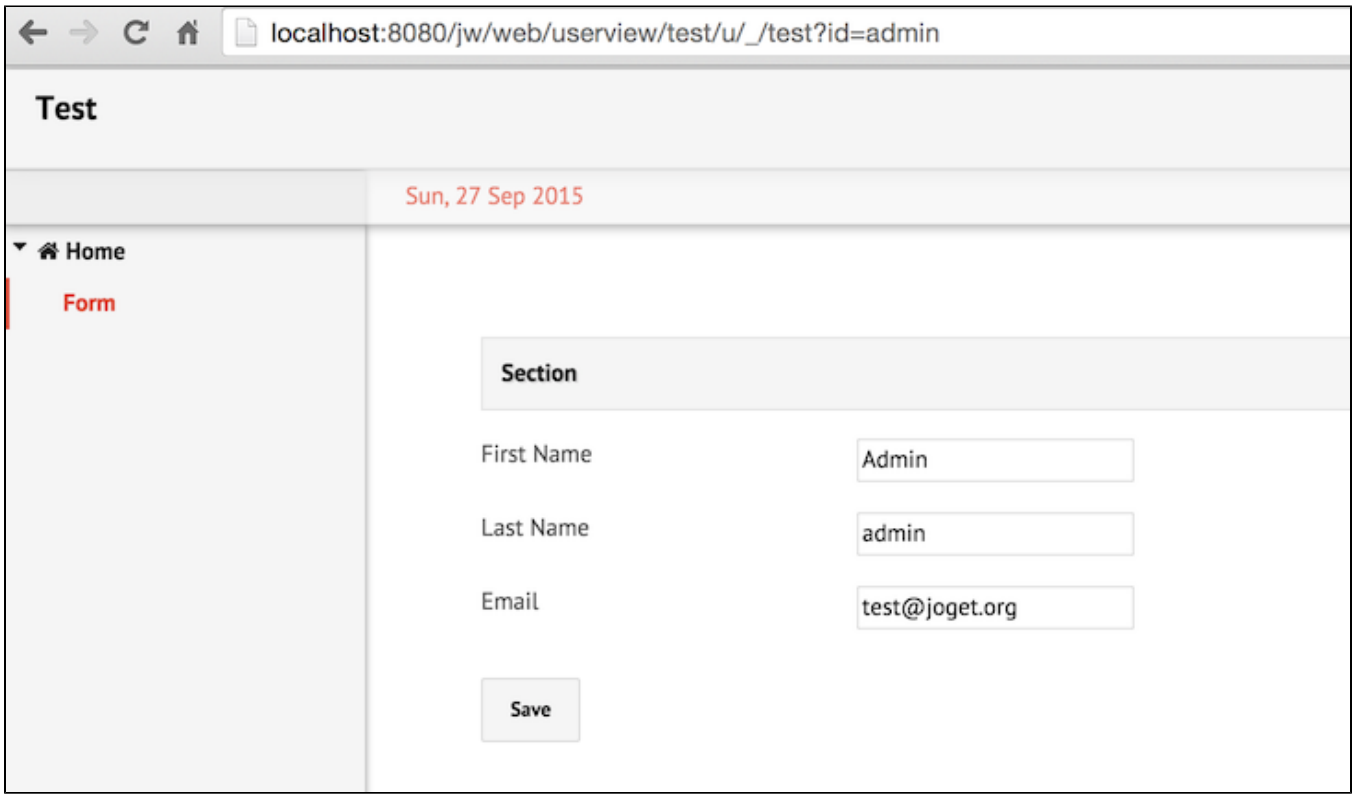
Then, configure the load binder of the form to use JDBC Binder with the following query.

```
select * from dir_user where username = ?
```



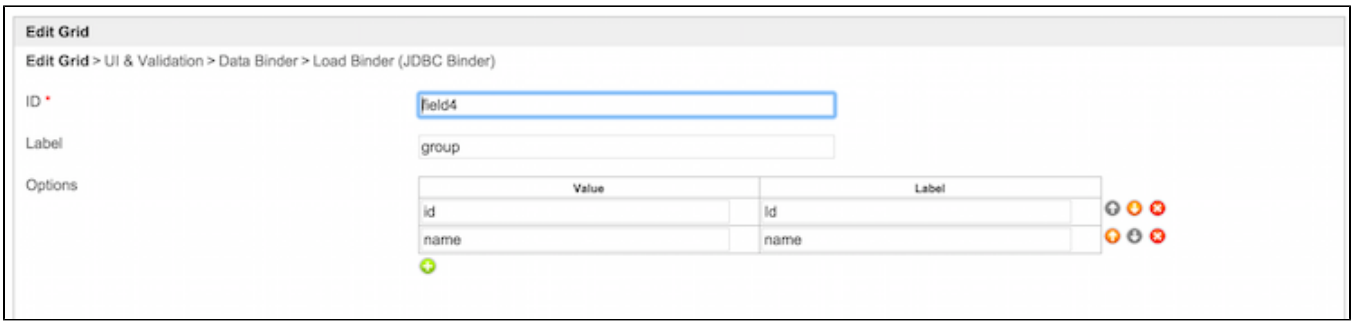


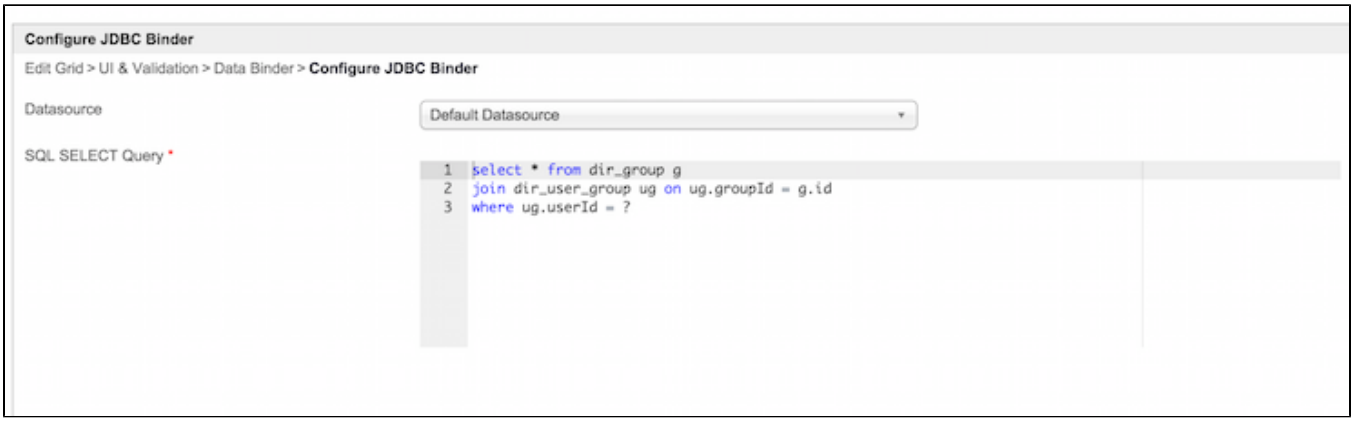
Let create an userview and drag a Form Menu element to display our form. Then, publish it and test our form with an "id" parameter.



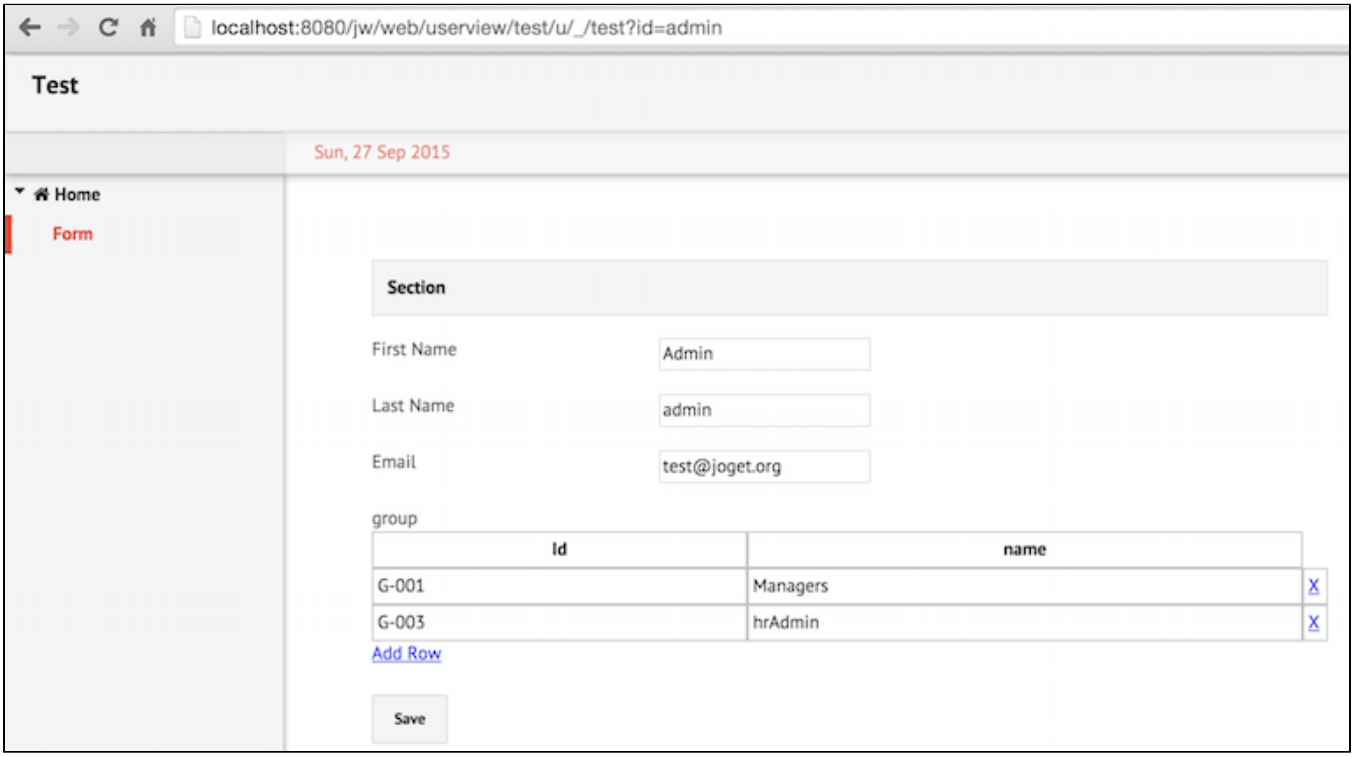
Next, let add a grid element and test with the following query.

```
select * from dir_group g
join dir_user_group ug on ug.groupId = g.id
where ug.userId = ?
```





Let check our result again.



It works! Please remember test the other features of the plugin as well. 😂

8. Take a step further, share it or sell it

You can download the source code from jdbc_load_binder_src.zip.

To download the ready-to-use plugin jar, please find it in http://marketplace.joget.org/.