

SERENITY BDD con CUCUMBER

6

Trabajando con bases de datos DB2

(Tiempo ejecución 2 horas)

Objetivos:

Suministrar unas clases genéricas que permitirán a los analistas realizar una conexión a un sistema de bases de datos DB2 y posteriormente ejecutar cualquier query.

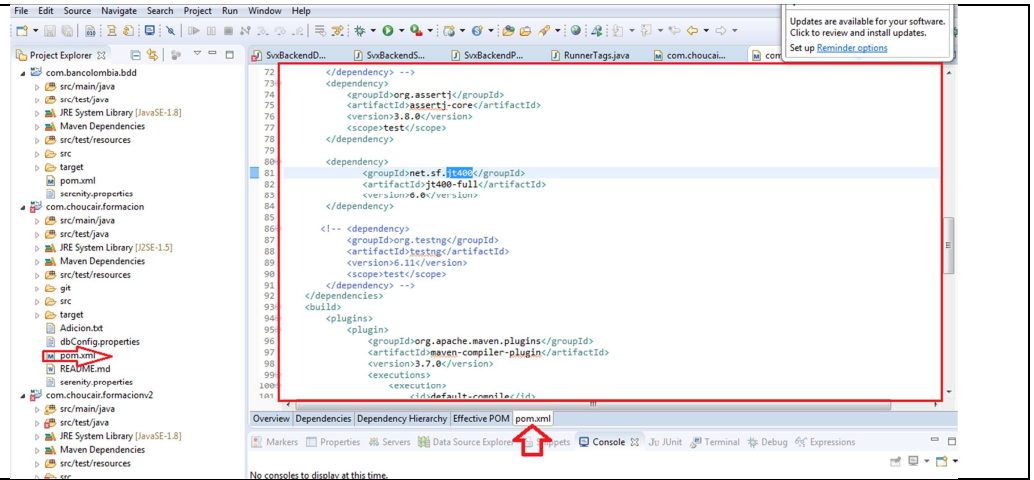
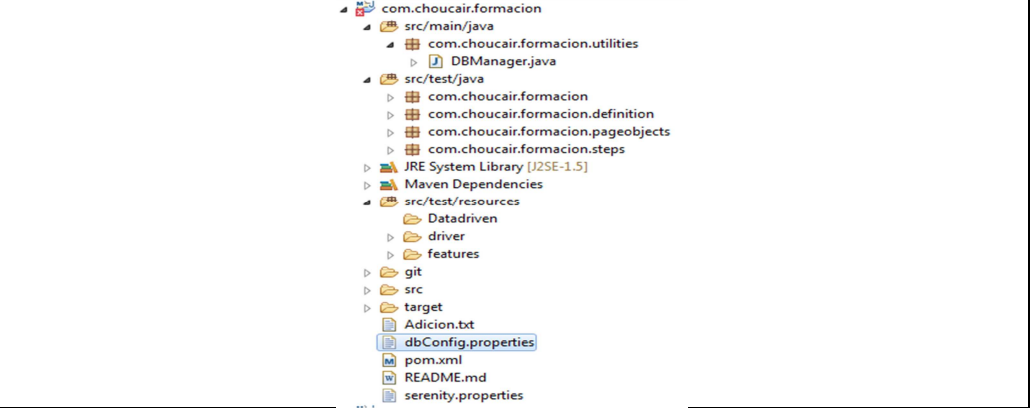

Los analistas podrán con la información recibida como resultado de la ejecución del Query, recorrer y realizar las respectivas verificaciones.

Se pondrá en práctica otra forma de utilizar los datos en Cucumber, como lo es a través del Scenario Outline.

Preparación del Entorno

Antes de iniciar el proceso de consulta, es necesario realizar algunas configuraciones.

Archivo pom.xml, como vimos anteriormente, en los proyectos maven se cuenta con un archivo “pom.xml” el cual contiene datos de configuración de nuestro proyecto, como dependencias con otros jar, tipos de informes que queremos en la página web de nuestro proyecto, etc.. dentro de este archivo encontrará una sección que inicia como “<dependencies>” y finaliza con “</dependencies>”, en medio de estas se debe agregar la fracción de xml que maven requiere para descargar el **jar** que necesitamos para nuestro proyecto.

Editar el archivo pom.xml Doble clic al archivo pom.xml > clic a la pestaña pom.xml	
Agregar la dependencia Al interior de las etiquetas <dependencies> </dependencies> Agregue el siguiente xml agregará el jar para la conexión a bases de datos DB2.	<pre><dependency> <groupId>net.sf.jt400</groupId> <artifactId>jt400-full</artifactId> <version>6.0</version> </dependency></pre>
Copiar el archivo dbconfig.properties en una ruta local. Clic derecho al proyecto > New > File > asignar un nombre	
El archivo dbconfig.properties contiene información necesaria para la conexión. Los campos db.user y db.password hacen referencia al usuario y clave para ingresar a iseries, reemplácelos con la información de su usuario.	<pre>db.driver=com.ibm.as400.access.AS400JDBCDriver db.url=jdbc:as400:10.9.2.221 db.user=CXXXXXXX db.password=????????</pre>
Crear un paquete de utilidades Para efectos de organización crearemos un repositorio para las clases que consideremos utilitarias. en la ruta src/main/java, agregue un paquete con el siguiente nombre: “com.choucair.formacion.utilities”	

<p>Crear la clase DBManager</p> <p>Agregué una clase al paquete de utilities con el nombre DBManager.</p> <p>Edite la clase y agregue el código adjunto.</p> <p>Reemplace el código resaltado por la ruta completa de la ubicación de su archivo en su máquina.</p> <p>Más adelante podemos optimizar el código para que utilice una ruta relativa.</p>	<pre> package com.choucair.formacion.utilities; import java.io.FileReader; import java.io.IOException; import java.sql.Connection; import java.sql.DriverManager; import java.sql.SQLException; import java.util.Properties; public class DBManager { private static DBManager instance; private String url; private String user; private String password; /** Creates a new instance of DBManager */ private DBManager() { inicialice(); } public void inicialice() { try { Properties prop = new Properties(); prop.load(new FileReader("C:/rutacompleta/dbConfig.properties")); this.url = prop.getProperty("db.url"); this.user = prop.getProperty("db.user"); this.password = prop.getProperty("db.password"); Class.forName(prop.getProperty("db.driver")); } catch (IOException e) { // TODO Auto-generated catch block e.printStackTrace(); } catch (ClassNotFoundException e) { // TODO Auto-generated catch block e.printStackTrace(); } } //aplicando Singleton public static DBManager getInstance() { if (instance == null) instance = new DBManager(); return instance; } public Connection getConeccion() throws SQLException { Connection con = DriverManager.getConnection(this.url, this.user, this.password); return con; } } </pre>
<p>Crear la clase Sql_Execute</p> <p>Agregué una clase al paquete de utilities con el nombre Sql_Execute.</p> <p>Edite la clase y agregue el código adjunto.</p>	<pre> package com.choucair.formacion.utilities; import java.sql.Connection; import java.sql.PreparedStatement; import java.sql.ResultSet; import java.sql.SQLException; import com.choucair.formacion.utilities.DBManager; public class Sql_Execute { DBManager manager = null; /** Creates a new instance of DAOResultSet */ public Sql_Execute() { manager = DBManager.getInstance(); } public ResultSet sql_Execute (String Query) throws SQLException { manager = DBManager.getInstance(); Connection con = manager.getConeccion(); PreparedStatement ps = con.prepareStatement(Query); ResultSet rs = ps.executeQuery(); return rs; } } </pre>

Preparación datos de prueba

Para efectos del taller necesitamos contar con el acceso a una **tabla** de pruebas en la base de datos DB2, con los permisos necesarios para realizar consultas por ODBC a las diferentes tablas y librerías (en el caso del Banco es necesario colocar un ticket solicitando permisos *Change o *Use).

Si al momento de realizar este taller no contamos con ese tipo de permiso sobre alguna tabla, podemos realizar la siguiente actividad: ingresar a iseries con nuestro usuario y copiar la información (o parte de ella) de una tabla ya existente a otra tabla temporal, de esta forma la tabla creada tendrá los permisos necesarios para ser accedida de forma externa por el mismo usuario que la creo.

Para efectos del taller usaremos como referencia la tabla **VISIONR.CNAME**, de la cual vamos a extraer 4 campos y 10 registros y los llevaremos a la tabla **GPLILIBRA.TMPCNAME**.

Los campos a extraer son:

CNNOSS = Numero de documento del cliente

CNCDTI = Tipo de identificación

CNNAME = Nombre del Cliente

CNCDCC = Control de terceros o tipo de persona

Query de ejemplo, para la creación de una tabla temporal de la visionr.cname.

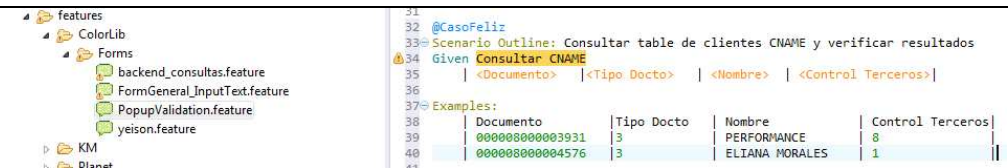
```
> create table gplilibra.tmpcname as (  
select CNNOSS, CNCDTI, CNNAME, CNCDCC from visionr.cname  
where cnnoSS like '000008%'  
fetch first 10 row only) with data
```

Crear feature y agregar un escenario.

Para el ejemplo vamos a utilizar el Scenario Outline:

En este caso la tabla de datos va después del ítem "Examples:" y el escenario se ejecutaría de acuerdo a la cantidad de filas de la tabla.

Para trasladar datos a una línea gherkin se hace utilizando el nombre del campo entre <>.



The screenshot shows a project structure on the left with folders like 'ColorLib' and 'Forms'. The main editor displays a Gherkin scenario outline for '@CasoFeliz' with a 'Given Consultar_CNAME' step. Below the step is an 'Examples:' table with 2 rows of data.

<Documento>	<Tipo Docto>	<Nombre>	<Control Terceros>
00000800003931	3	PERFORMANCE	8
00000800004576	3	ELIANA MORALES	1

Preparamos el Runner

```
@CucumberOptions (features = "src/test/resources/features/ColorLib/Forms/backend_consultas.feature", tags = "@CasoFeliz")  
public class RunnerTags {  
  
}
```

Ejecutar para obtener el método "Definition" propuesto

You can implement missing steps with the snippets below:

```
@Given("^Consultar CNAME$")  
public void consultar_CNAME(DataTable arg1) throws Throwable {  
    // Write code here that turns the phrase above into concrete actions  
    // For automatic transformation, change DataTable to one of  
    // List<YourType>, List<List<E>>, List<Map<K,V>> or Map<K,V>.  
    // E,K,V must be a scalar (String, Integer, Date, enum etc)  
    throw new PendingException();  
}
```

Implementar en **definition**
-Crear la clase definition
"BackendAs400db2Definition"
-Agregar el método propuesto

Nota:

Como en la feature utilizamos una tabla debajo de la línea gherkin, en la clase se crea un parámetro tipo DataTable para recibir los datos.

```
11 public class BackendAs400db2Definition {  
12  
13     @Steps  
14     BackendAs400db2Steps backendAs400db2Steps;  
15  
16     @Given("^Consultar CNAME$")  
17     public void consultar_CNAME(DataTable dtDatosPrueba) throws Throwable {  
18         List<List<String>> data = dtDatosPrueba.raw();  
19         backendAs400db2Steps.consultar_CNAME(data);  
20     }  
21 }
```

Como vemos, se ha creado la clase "BackendAs400Steps" y el método "Consultar_CNAME" para desarrollar posteriormente el paso a paso

<p>Implementar Pageobject</p> <ul style="list-style-type: none"> -Crear la clase pageobjects "BackendAs400db2Page" -Crear un método para cada uno de los siguientes pasos: <ul style="list-style-type: none"> ➤ Crear Query ➤ Ejecutar Query ➤ Verificar resultados 	<p>Crear Query</p> <p>Es indispensable conocer previamente el query requerido para la consulta y los datos necesarios, en este caso el dato requerido para armar la consulta es el documento del cliente:</p> <p>Query a implementar:</p> <p>"SELECT * FROM GLIBRA.TMPCNAME WHERE CNNOSS = '<documento>'"</p> <p>Lo que haremos es crear un texto con el query y utilizaremos una palabra clave para el documento, el cual posteriormente será reemplazado por el dato de prueba recibido previamente por parámetro.</p> <p>El método es definido tipo String, para que podamos retornar el query ya organizado</p> <pre> public String Armar_Query_Consulta_CNAME(String strDocumento) { String strQuery = "SELECT * FROM GPLILIBRA.TMPCNAME2 WHERE CNNOSS = '<documento>'"; strQuery = strQuery.replace("<documento>", strDocumento); return strQuery; } </pre> <p>Ejecutar Query</p> <p>Ahora debemos crear un método genérico que nos permita ejecutar de forma genérica cualquier query recibido y retorne el resultado (resultset) de la consulta.</p> <p>Para esto es necesario:</p> <ul style="list-style-type: none"> -recibir como parámetro el query organizado previamente. -Ejecutar la consulta consumiendo la clase Sql_Execute -retornar el resultset respectivo <pre> public ResultSet Ejecutar_Query(String Query) throws SQLException { Sql_Execute DAO = new Sql_Execute(); ResultSet rs = DAO.sql_Execute(Query); return rs; } </pre> <p><i>Nota: este método es genérico, por lo que puede seguir siendo utilizado en posteriores consultas.</i></p> <p>Verificar Resultados</p> <p>Por último crearemos el método encargado de recorrer el resultset y realizar las respectivas verificaciones.</p> <p>Para esto es necesario contar con el resultset y con los datos de prueba enviados desde la feature.</p> <pre> public void Verificar_Consulta_CNAME(ResultSet rs, List<List<String>> data) throws SQLException { while (rs.next()) { String Documento_Recibido = rs.getString(1); String Documento_Esperado = data.get(0).get(0); assertThat(Documento_Recibido, equalTo(Documento_Esperado)); String TipoDoc_Recibido = rs.getString(2); String TipoDoc_Esperado = data.get(0).get(1); assertThat(TipoDoc_Recibido, equalTo(TipoDoc_Esperado)); String Nombre_Recibido = rs.getString(3); String Nombre_Esperado = data.get(0).get(2); assertThat(Nombre_Recibido.trim(), equalTo(Nombre_Esperado.trim())); String CtrlTercero_Recibido = rs.getString(4); String CtrlTercero_Esperado = data.get(0).get(3); assertThat(CtrlTercero_Recibido.trim(), equalTo(CtrlTercero_Esperado.trim())); } } </pre> <p>Si observamos el código anterior, estamos comparando el valor recibido en el resultset con el recibido desde la feature</p> <p>rs.getString(1) //hace referencia al primer campo del resultset.</p> <p>data.get(fila).get(columna) //hace referencia a una celda de la tabla de datos.</p> <p>assertThat //permite la comparación entre los campos (ver la ayuda en la siguiente línea).</p> <p>Recuerde importar las librerías de hamcrest necesarias para el uso del assertThat.</p>
<p>Ayuda : hamcrest</p>	<p>hamcrest usa el método assertThat como una expresión de emparejamiento para determinar si la prueba fue exitosa.</p> <pre> import static org.hamcrest.MatcherAssert.assertThat; import static org.hamcrest.Matchers.is; import static org.hamcrest.Matchers.equalTo; boolean a; boolean b; // all statements test the same assertThat(a, equalTo(b)); assertThat(a, is(equalTo(b))); assertThat(a, is(b)); </pre> <p>Ver más: http://www.vogella.com/tutorials/Hamcrest/article.html</p>

- Crear la clase *steps*
"BackendAs400db2Steps"
- Agregar los pasos correspondientes:
 - Crear Query
 - Ejecutar Query
 - Verificar resultados

Notas:

- Se instancia la clase page creada previamente
- El método “Consultar_CNAME” recibe como parámetro la lista con los datos de prueba.
- La variable **strDocumento**, recibe el valor de la primera celda de la tabla de datos.
- Para el paso “Crear query”, se requiere invocar el método “Armar_Query_Consulta_CNAME” pasando el número de documento del cliente y el resultado será el query a ejecutar, éste lo asignamos a la variable **query**
- El paso Ejecutar consulta, requiere del uso del método “Ejecutar_Query” pasando por parámetro el **query** organizado previamente. Y el resultado lo asignamos a la variable **rs** tipo ResultSet.
- El paso final de verificación, lo hacemos, invocando el método “Verificar_Consulta_CNAME”, el cual recibe como parámetros el resultset (**rs**) y los datos de prueba (**data**).

C:\VertranPOC\com.choucair.formacionv2\target\site\serenity\title-of-your-feature_cor

Acceso Remoto Bancolombia Consultar tabla de clientes ...

Archivo Edición Ver Favoritos Herramientas Ayuda

Galería de Web Slice Sitios sugeridos

Title Of Your Feature

I want to use this template for my feature file

Tag (tag)
Casofeliz (tag)
Forms (capability)

Consultar Tabla De Clientes CNAME Y Verificar Resultados

Scenario Outline

Given Consultar CNAME

<Documento> <Tipo Docto> <Nombre> <Control Terceros>

Examples:

Show	entries	Search:		
#	Documento	Tipo Docto	Nombre	Control Terceros
1	000008000003931	3	PERFORMANCE	8
2	000008000004576	3	ELIANA MORALES	1

Showing 1 to 2 of 2 entries Previous Next

Steps	Outcome	Duration
Consultar tabla de clientes CNAME y verificar resultados #1: (Documento=000008000003931, Tipo Docto=3, Nombre=PERFORMANCE, Control Terceros=8)	SUCCESS	1.84s
Given Consultar CNAME 000008000003931 3 PERFORMANCE 8	SUCCESS	1.83s
Consultar CNAME: [[000008000003931, 3, PERFORMANCE, 8]]	SUCCESS	1.62s
Consultar tabla de clientes CNAME y verificar resultados #2: (Documento=000008000004576, Tipo Docto=3, Nombre=ELIANA MORALES, Control Terceros=1)	SUCCESS	0.9s
Given Consultar CNAME 000008000004576 3 ELIANA MORALES 1	SUCCESS	0.9s
Consultar CNAME: [[000008000004576, 3, ELIANA MORALES, 1]]	SUCCESS	0.89s

