Ismael Calvo Villalvilla

51689141E

alaricoi@gmail.com

TRABAJO 2 DE EVALUACIÓN CONTINUA

Generación Automática de Código

Contenido

1 Introducción 2

2 Desarrollo de la práctica 3

2.1 Recursos de programación utilizados 3

2.2 Descripción (especificación) detallada de la solución 3

2.3 Alcance y limitaciones de la solución propuesta 10

2.4 Descripción de los casos de Prueba 10

3 Ubicación de los programas fuente y explicación del resto de directorios/ficheros anexados 12

4 Manuales de uso/instalación 13

5 Conclusiones. 16

# Introducción

En el marco de la asignatura de Generación Automática de Código del máster universitario Investigación en Ingeniería de Software y Sistemas Informáticos, a continuación, se describe la solución para resolver el trabajo 2.

Se trata de exportar el contenido de cualquier tabla de un SGBD relacional (o un subconjunto de registros de las mismas) a los siguientes formatos:

* CSV
* HTML
* XML
* JSON

El generador tendrá como entrada el nombre de la tabla (y eventualmente un filtro “where”, si la tabla tiene muchos registros)

El programa generador mostrará el contenido seleccionado en los formatos indicados.

Se ha decidido, como gestores de base de datos, MySQL y SQLite, las razones principales:

* Son de fácil instalación y muy flexibles a la hora de acceder desde distintos lenguajes y sistemas. SQLite es un SGBD contenido en un fichero fácilmente gestionable lo que nos proporciona mucha flexibilidad para ejemplos académicos.
* Ser de código abierto.

La forma de obtener la conexión se realiza de forma genérica por jdbc y cargando el conector en tiempo de ejecución por lo que ampliar a otros SGDB sería asumible.

Por otra parte, en cuanto al desarrollo del documento, empezamos por describir los recursos empleados para llevar a cabo el trabajo, así como el lenguaje de programación seleccionado, seguimos con la descripción detallada de la solución, incluyendo las pruebas realizadas. Por último, se darán las indicaciones oportunas para su instalación y uso para acabar con la conclusión.

# Desarrollo de la práctica

## Recursos de programación utilizados

Al igual que el trabajo 1, el lenguaje elegido para el desarrollo es java (jdk 1.8). Esta elección se ha basado principalmente la comodidad a la hora de gestionar las conexiones a SGDB y su flexibilidad a la hora de trabajar con ficheros, por otra parte es un lenguaje de programación ampliamente difundido y como de trabajar en cualquier sistema operativo. Las herramientas utilizadas para la construcción del programa son:

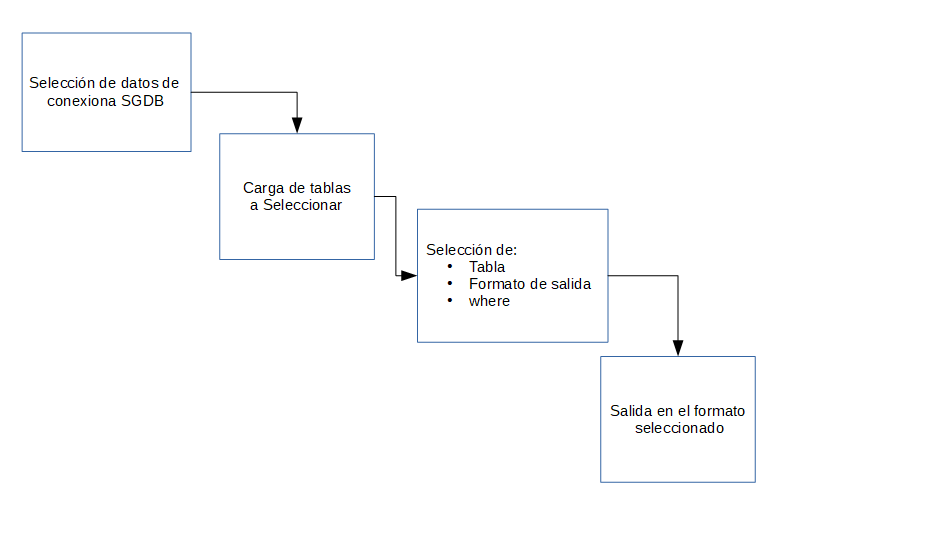
* Sistema operativo Windows 10.
* Como IDE (Integrated Development Environment) he utilizado Eclipse Oxigen en su instalación de Java.
* Librerías JAVA de terceros:
  + librería de SWT (Standard Widget Toolkit), de eclipse y openSource, nos permite crear de aplicaciones de ventanas y que crea la interfaz de usuario a partir de componentes operativos de cada Sistema operativo.
  + Librerías de conexión a SGDB. Para el trabajo hemos empleado SQLite y MySQL, sus conectores son los ficheros:
    - * mysql-connector-java-5.1.45-bin.jar
      * sqlite-jdbc-3.21.0.jar

## Descripción (especificación) detallada de la solución

Se ha realizado una aplicación de escritorio en entorno gráfico que de manera sencilla nos va a permitir realizar todas las acciones. Al igual que en el trabajo 1, todo el código está contenido en la clase t2App, el entorno gráfico se crea a partir de la librería SWT de eclipse.

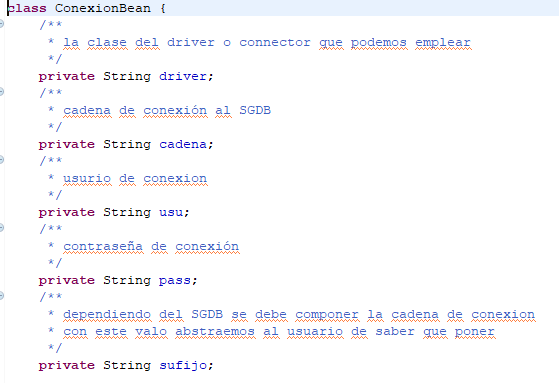
El proceso se divide en dos partes de entrada de datos diferencias:

1. Una primera parte de introducción donde se deberá seleccionar y el tipo de SGDB y los parámetros de conexión. Tendrá cómo salida la lista de tablas que podremos utilizar para la exportación.
2. La segunda parte tiene como entrada la selección de la tabla a exportar, el posible filtro “where” y el tipo de exportación de datos.



Para la generación de los formatos de salida se ha optado por realizarlo mediante programación del formato de salida a partir de las especificaciones de dichos formatos. Otra posibilidad barajada ha sido la utilización de platillas, sobre todo teniendo en cuenta el formato HTML. pero tanto la homogeneidad de la entrada siempre será una lista de registros, como la rigidez de la especificación de formato de los otros tipos de exportación me han hecho decantarme por la solución programática. A continuación, se detalla cada uno de los procesos:

1. Seleccionar un SGDB, conectarnos a él. Para ello, se ha creado la clase **ConexionBean** que nos permite guardar los datos introducidos por pantalla en un objeto y así tratar los parámetros más fácilmente.



A partir de los datos introducidos por pantalla se genera la conexón y se carga el comboBox con las tablas, los datos solicitados son:

* Tipo de conexión
* Cadena
* Usuario
* Contraseña

Desde el Botón de conectar llamamos al método **rellenaComboTablas**, que realiza:

/\*\*

\* Rellena el combo de tablas a partir de la base de datos seleccionada

\*/

**private** **void** rellenaComboTablas() {

// Es MySQL

String driver = **null**;

String sufijo = **null**;

**if** (comboTipo.getSelectionIndex() == 1) {

driver = "com.mysql.jdbc.Driver";

sufijo = "jdbc:mysql://";

}

// Es SQLite

**if** (comboTipo.getSelectionIndex() == 0) {

driver = "org.sqlite.JDBC";

sufijo = "jdbc:sqlite:";

}

// creamos el objeto conexión con los datos seleccionados

conexionBean = **new** ConexionBean(driver, tCadenaConex.getText(), tUsu.getText(), tPass.getText(), sufijo);

// conexion a bese de datos

Connection conn = dameConexion(conexionBean);

// Si no obtenemos conexión salimos del método

**if** (conn == **null**) {

**return**;

}

ResultSet rs = **null**;

**try** {

// extraccion del catalogo de datos del SGDB

DatabaseMetaData metaDatos = conn.getMetaData();

rs = metaDatos.getTables(**null**, **null**, "%", **null**);

comboTablas.removeAll();

**while** (rs.next()) {

// El contenido de cada columna del ResultSet se puede ver

// en la API, en el metodo getTables() de DataBaseMetaData.

// La columna 1 es TABLE\_CAT

// y la 3 es TABLE\_NAME

// String catalogo = rs.getString(1);

String tabla = rs.getString(3);

comboTablas.add(tabla);

}

muestraDialogoModal(SWT.***ICON\_INFORMATION*** | SWT.***OK***, "Información", "Creada la conexión");

comboTablas.setFocus();

} **catch** (SQLException e1) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error", "No se ha podido crear la conexión");

***logger***.error(e1);

// liberamos los objetos de de conexion

} **finally** {

**try** {

**if** (rs != **null**)

rs.close();

**if** (conn != **null**)

conn.close();

} **catch** (SQLException e1) {

***logger***.error(e1);

}

}

}

El método **dameConexion** nos crea la conexión a base de datos:

/\*\*

\* Crea una conexxión jdbc a partir de los parametros pasados por parametro

\*

\* **@param** tipo

\* valores: 0--> MySql 1--> SQLite

\* **@param** conexion:

\* cadena de conexión, en caso de SQLlite será la ruta del fichero db

\* en caso de que sea Mysql debemos indicar el host:puerto/esquema

\* **@param** usu:

\* usuario de la base de datos

\* **@param** pass:

\* contraseña de acceso

\* **@return**

\*/

**private** Connection dameConexion(ConexionBean con) {

**try** {

// carga del driver o conector

Class.*forName*(con.getDriver());

} **catch** (ClassNotFoundException e1) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha podido crear la conexión: " + e1.getMessage());

***logger***.error(e1);

}

Connection conexion = **null**;

**try** {

// creamos la conexion a base de datos

conexion = DriverManager.*getConnection*(con.getSufijo() + con.getCadena(),

con.getUsu(), con.getPass());

} **catch** (SQLException e1) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha podido crear la conexión: " + e1.getMessage());

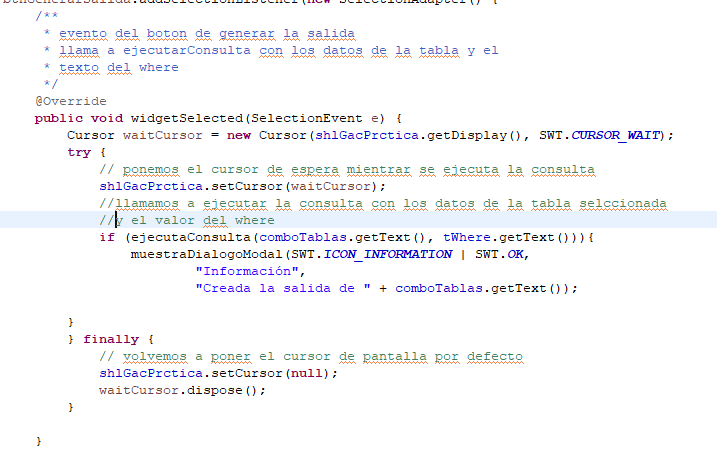
***logger***.error(e1);

}

**return** conexion;

}

1. Una vez conectados se carga en un combo las tablas encontradas en el sistema para que seleccionemos la deseada.
2. Opcionalmente se presenta un cuadro de texto para incorporar la sentencia where que consideremos oportuna.
3. Seleccionaremos el formato de salida mediante un combo y pulsaremos el botón de “**Generar**”. A partir del objeto ConexionBean solicitamos una conexión al método **dameConexion** y se genera la select con la tabla seleccionada y se añade el valor del campo **where**, el formato es sql estándar, no debe introducirse la palabra where.



El método **ejecutarConsulta** realiza todo el proceso de conexión generación de y ejecución de consulta y muestra de resultados:

/\*\*

\* A partir de la tabla seleccionada y la base de datos conectada se ejecuta

\*/

**private** **boolean** ejecutaConsulta(String tabla, String where) {

**if** (conexionBean == **null**) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"Para realizar este paso hace falta conectarse previamente");

***logger***.error("No hay conexión creada");

**return** **false**;

}

// Obtenemos la conexion

Connection conn = dameConexion(conexionBean);

ResultSet rs = **null**;

Statement st = **null**;

**try** {

// Creamos el Statement para poder hacer consultas

st = conn.createStatement();

String sql = "Select \* from " + tabla;

// En caso que hayamos incorporado una clausa where

**if** (!where.equals(""))

sql += " where " + where;

rs = st.executeQuery(sql);

**switch** (comboFormatSalida.getSelectionIndex()) {

// segun el combo de salida se genera el parseado

**case** 0:

// CSV;

tSalida.setText(rellenaCSV(rs));

lblSalida.setText(tabla + ".csv");

**break**;

**case** 1:

// HTML;

tSalida.setText(rellenaHTML(tabla, rs));

lblSalida.setText(tabla + ".html");

**break**;

**case** 2:

// XML;

tSalida.setText(rellenaXML(tabla, rs));

lblSalida.setText(tabla + ".xml");

**break**;

**case** 3:

// JSON

tSalida.setText(rellenaJSON(tabla, rs));

lblSalida.setText(tabla + ".json");

**default**:

**break**;

}

} **catch** (SQLException e) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"Error al ejecutar la consulta: " + e.getMessage());

***logger***.error(e);

**return** **false**;

} **finally** {

//se liberan los objetos de conexión

**try** {

**if** (rs != **null**)

rs.close();

**if** (st != **null**)

st.close();

**if** (conn != **null**)

conn.close();

} **catch** (SQLException e1) {

***logger***.error(e1);

**return** **false**;

}

}

**return** **true**;

}

Cada tipo de salida se ha separado en un método individual diferente. El proceso de generación de salida se basa en maquetado de fichero según el formato

/\*\*

\* Recorre el resultset pasado como parametro y rellena un string con formato

\* JSON, ponemos una clave raiz con valos la lista de resultados

\*

\* **@param** rs

\* **@return**

\* **@throws** SQLException

\*/

**private** String rellenaJSON(String tabla, ResultSet rs) **throws** SQLException {

String resultado = "{\""+tabla + "\":[\n";

ResultSetMetaData metadata = rs.getMetaData();

String dato;

**while** (rs.next()) {

resultado += "{";

**for** (**int** i = 1; i <= metadata.getColumnCount(); i++) {

**if** (rs.getString(i) != **null**)

dato = rs.getString(i);

**else**

dato = "";

resultado += "\"" + metadata.getColumnName(i) + "\":" + "\"" + dato + "\"";

// no es el ultimo, se añade coma

**if** (i < metadata.getColumnCount()) {

resultado += ",\n";

}

}

resultado += "},\n";

}

// hay que quitar la ultima coma

resultado = resultado.substring(0, resultado.length() - 2);

resultado += "\n]}";

**return** resultado;

}

/\*\*

\* Crea la salida en xml, anidamos la marca principal con el nombre de

\* la tabla. Cada registro ira entre la marca <fila>..</fila>

\* **@param** tabla

\* **@param** rs

\* **@return**

\* **@throws** SQLException

\*/

**private** String rellenaXML(String tabla, ResultSet rs) **throws** SQLException {

ResultSetMetaData metadata = rs.getMetaData();

String resultado = "<" + tabla + ">\n";

**while** (rs.next()) {

resultado += "<fila>\n";

**for** (**int** i = 1; i <= metadata.getColumnCount(); i++) {

resultado += "<" + metadata.getColumnName(i) + ">";

**if** (rs.getString(i) != **null**)

resultado += rs.getString(i);

resultado += "</" + metadata.getColumnName(i) + ">\n";

}

resultado += "</fila>\n";

}

resultado += "</" + tabla + ">";

**return** resultado;

}

/\*\*

\* Genera el html a partir del resultset de la consulta realizada

\* **@param** tabla

\* **@param** rs

\* **@return**

\* **@throws** SQLException

\*/

**private** String rellenaHTML(String tabla, ResultSet rs) **throws** SQLException {

ResultSetMetaData metadata = rs.getMetaData();

//caberera del fichero

String resultado ="<html><body>";

resultado += "<h1>" + tabla + "</h1>\n";

//creación del objeto table

resultado += "<table>\n";

resultado += "<tr>\n";

// ponemos una celda de cabecera por cada campo

**for** (**int** i = 1; i <= metadata.getColumnCount(); i++) {

resultado += "<th>" + metadata.getColumnName(i) + "</th>\n";

}

resultado += "</tr>\n";

// por cada registro ponemos un tr y cada campo va enmarcado en celda (td)

**while** (rs.next()) {

resultado += "<tr>\n";

**for** (**int** i = 1; i <= metadata.getColumnCount(); i++) {

resultado += "<td>";

**if** (rs.getString(i) != **null**)

resultado += rs.getString(i);

resultado += "</td>\n";

}

resultado += "</tr>\n";

}

// cerramos la tabla y el tag de fichero

resultado += "</table>\n";

resultado +="</body></html>";

**return** resultado;

}

/\*\*

\* Metodo que genera la salida csv, el separador de campo es el ;

\* **@param** rs

\* **@return**

\* **@throws** SQLException

\*/

**private** String rellenaCSV(ResultSet rs) **throws** SQLException {

ResultSetMetaData metadata = rs.getMetaData();

String resultado = "";

**for** (**int** i = 1; i <= metadata.getColumnCount(); i++) {

resultado += metadata.getColumnName(i);

**if** (i < metadata.getColumnCount())

resultado += ";";

}

resultado += "\n";

**while** (rs.next()) {

**for** (**int** i = 1; i <= metadata.getColumnCount(); i++) {

**if** (rs.getString(i) != **null**)

resultado += rs.getString(i);

**if** (i < metadata.getColumnCount())

resultado += ";";

}

resultado += "\n";

}

**return** resultado;

}

1. Se da la opción de salvar la salida a un fichero. Mostrará un cuadro de dialogo para seleccionar el destino del fichero.

Button btnGuardar = **new** Button(grpSalida, SWT.***NONE***);

btnGuardar.addSelectionListener(**new** SelectionAdapter() {

/\*\*

\* Guarda la salida en un fichero

\*/

@Override

**public** **void** widgetSelected(SelectionEvent e) {

String tabla = lblSalida.getText().trim();

// control de que haya generado la salida

**if** (tabla == **null** || tabla.equals("")) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha creado todavia la salida");

**return**;

}

// dialogo para seleccionar la salida de fichero

FileDialog dlg = **new** FileDialog(shlGacPrctica, SWT.***SAVE***);

dlg.setFileName(lblSalida.getText());

dlg.setOverwrite(**true**);

String fileName = dlg.open();

// guardamos el fichero si se selecciono el destino

**if** (fileName != **null** && !fileName.equals("")) {

**try** {

Files.*write*(Paths.*get*(fileName), tSalida.getText().getBytes());

} **catch** (IOException e1) {

muestraDialogoModal(SWT.***ICON\_ERROR*** | SWT.***OK***, "Error",

"No se ha podido crear el fichero: " + e1.getMessage());

***logger***.error(e1);

}

}

}

});

## Alcance y limitaciones de la solución propuesta

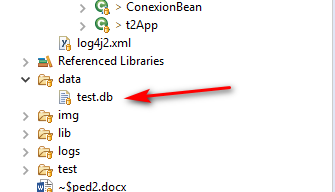
Es necesario tener una máquina virtual java 1.8 y un sistema operativo modo Windows.

Alcanza solo a los SGBD descritos en el punto anterior, por tanto no se conectara a otros tipos, si introducimos una cadena de conexión distinta a SQLite o MySQL obtendremos una ejecución errónea.

Por otra parte, la generación en HTML es fija, no existe la posibilidad de partir de una plantilla y modificarla posteriormente, a nos que se modifique el código del programa.

## Descripción de los casos de Prueba

Para el plan de pruebas hemos utilizado una base de datos SQLite de ejemplo que se ha incluido en la carpeta **data** de la entrega



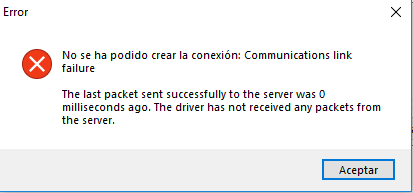
Las pruebas de MySQL se han realizado con las tablas del esquema de sistema “mysql” de sistema del gestor.



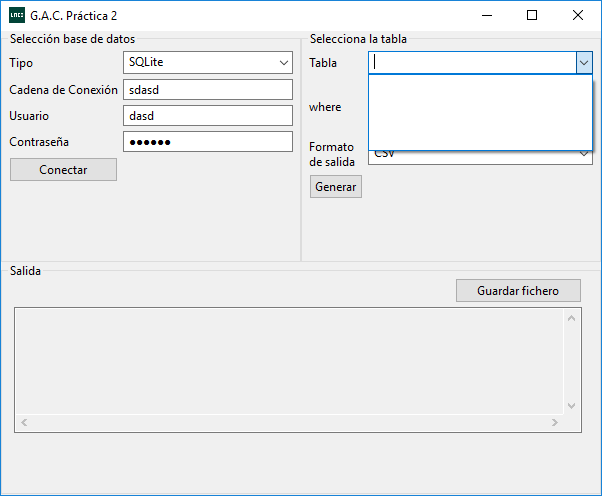
En la carpeta test se presentan las diferentes salidas

Las pruebas de control de errores se realizan con las siguientes situaciones:

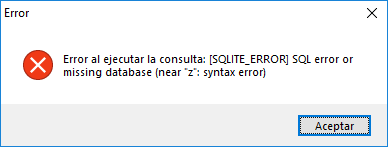
* Conexión de datos erróneas, en MYSQL se muestra el mensaje de error siguiente:



En SQLite no muestra error ya que intenta abrir un fichero, pero no carga el combo de tablas



* Otro posible caso de error es la introducción de un **where** erróneo, en este caso nos mostrara el error de SGDB:

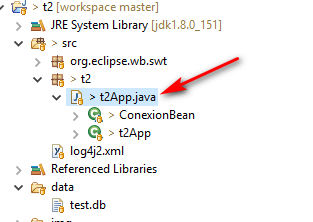


El resto de datos de pantalla se insertan mediante comboBox por lo que no son susceptibles a error.

# Ubicación de los programas fuente y explicación del resto de directorios/ficheros anexados

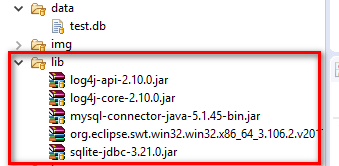
A continuación, se pasa a describir la entrega realizada, al descomprimir el fichero zip nos encontraremos con el siguiente árbol de directorios.

**Carpeta src:** Contiene los fuentes de la aplicación. En esta caperta se encuentra el archivo como la clase **t2App.java** dentro de la carpeta t2, en esta clase esta la funcionalidad principal de la aplicación



Dentro de la carpeta src también encontramos el paquete org que eclipse genera automáticamente cuando creamos un proyecto SWT.

**Carpeta lib,** se sitúan las librerías de terceros que hemos utilizado en el proyecto



**Carpeta test.** Contiene los ficheros de pruebas y sus salidas.

**Carpeta data.** Contiene el fichero de SQLite utilizado para las pruebas (**test.db**)

**Carpeta bin**. Contiene las clases compiladas puede estar vacío ya que solo tiene contenido en el momento de la compilación.

**Carpeta img**. Contiene las imágenes que se pueden utilizar en el proyecto

**Carpeta logs**. Contiene las trazas de ejecución del programa.

**Ficheros en el raíz**. Nos vamos a encontrar con os siguientes ficheros en el raíz.

**t2.jar**, librería ejecutable del programa

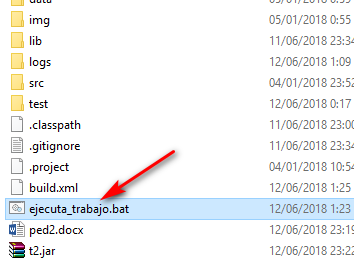
**ejecuta\_trabajo.bat**, en Windows, arranca el programa.

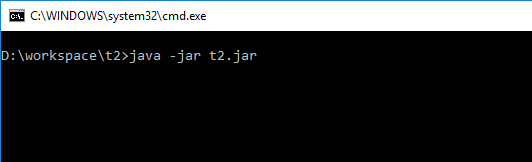
# Manuales de uso/instalación

Para usar el programa solo es necesario disponer de una máquina virtual java 1.8 instalada en el equipo y el sistema operativo debe ser visual.

Los pasos para ejecutar el programa son los siguientes:

* + 1. En Windows, ejecutar el archivo por lotes **ejecuta\_trabajo.bat**





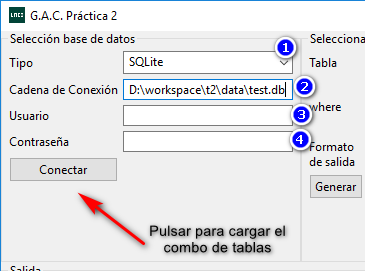
* + 1. Nos mostrara la siguiente ventana donde deberemos seleccionar primero los datos de conexón en la parte izquierda de la pantalla:
       1. Tipo de conexión.
       2. Cadena de conexión,
          - Mysql 🡪 formato maquina:puerto/base\_ de\_datos. Por ejemplo,

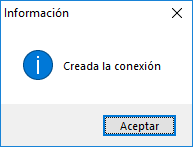
Localhost:3306/mysql

* + - * + SQLite🡪 path dentro del sistema operativo, por ejemplo

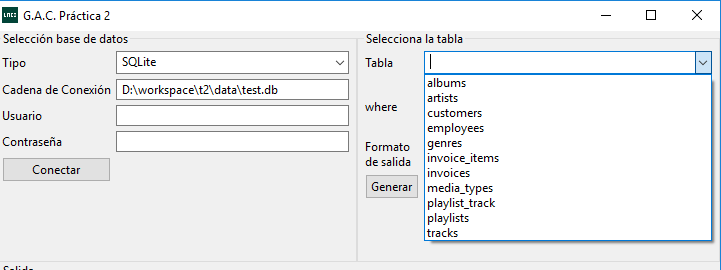
D:\workspace\t2\data\test.db

* + - 1. Usuario de conexión
      2. Contraseña de conexión



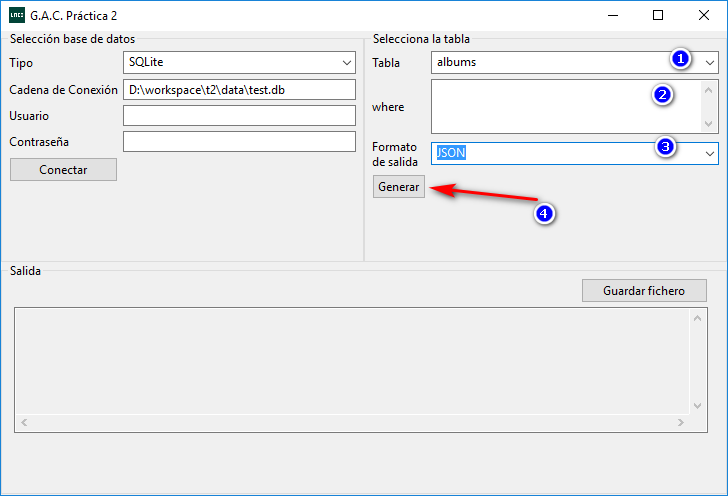


* + 1. Una vez que conectados, tendremos los nombres de las diferentes tablas cargados en el combo “tablas”.

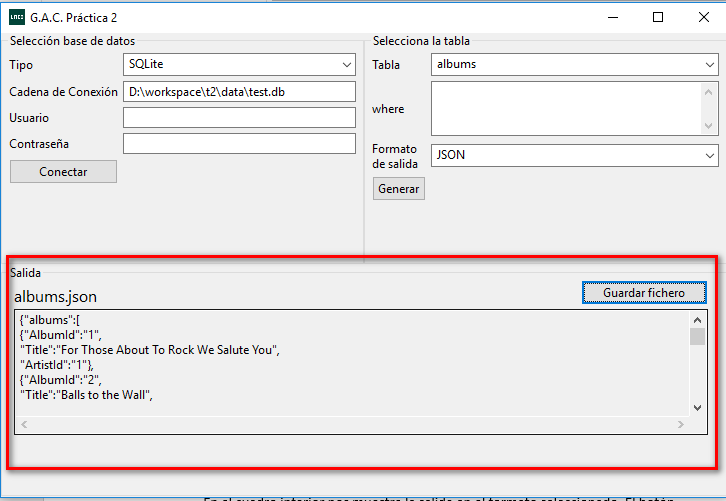


Podemos adicionalmente introducir un filtro where sobre los datos.

En este punto también debemos seleccionar el formato de salida y pulsaremos “Generar”.



En el cuadro inferior nos muestra la salida en el formato seleccionado. El botón Guardar fichero nos permite exportar a disco la salida.



# Conclusiones.

Teniendo en cuenta que nos encontramos ante un trabajo universitario que no tiene otro fin más que el académico, cabe destacar que el actual “estado del arte” de los entornos de desarrollo y lenguajes de progresión existe un gran avance en la búsqueda de mayor interrelación entre los sistemas, haciendo más fácil el traspaso de información entre sistemas y la interfaz de usuarios.

El presente trabajo va en la línea comentada en el párrafo anterior, por ejemplo, se gestiona la conexión con el SGDB de manera sencilla, hemos aportado la solución para dos sistemas, pero la inclusión de más sistemas sería baste fácil (no se ha contemplado partir de una configuración por fichero que podría evitar tocar código fuente por la necesidad también de añadir la librería de conexión en la distribución).

Del mismo modo, los formatos de salida json, xml y csv están ampliamente extendidos y son de facto estándares de comunicación entre sistemas de información, por tanto, la generación automática de exportación de datos a formatos entendibles por otros sistemas es un punto que está tomando mucha importancia, como ejemplo podemos indicar el auge de los servicios RESTFull basados en JSON como formato de mensaje y que actualmente se utilizan para la comunicación de dispositivos móviles o de IoT con sus respectivos backend.

Por otra parte, la intención de presentar el trabajo en un entorno gráfico es que sea lo más comprensible e intuitiva posible a la hora de probar la funcionabilidad dentro de un contexto académico. En otras situaciones, por ejemplo, como parte de otro sistema que permita publicar datos de una tabla en un servicio web, REST o SOAP (la generación xml sería algo más compleja pero asumible) la distribución sería como una API, sin entorno gráfico.