Acceso a Bases de Datos Relacionales

Acceso a una Base de Datos. ¿Qué es JDBC?

JDBC es el interfaz común que Java proporciona para poder conectarnos a cualquier SGBD Relacional con dicho lenguaje de programación. Proporciona una API completa para trabajar con Bases de Datos Relacionales de forma que sea cual sea el motor con el que conectemos, la API siempre será la misma. Simplemente tendremos que *hacernos* con el Driver correspondiente al motor que queramos usar, que si que dependerá totalmente de éste. A pesar de eso tampoco es mucho problema ya que actualmente podemos encontrar un driver JDBC para prácticamente cualquier SGBDR existente [1)].

Ya que, como hemos dicho, el driver es lo único que depende exclusivamente del SGBD utilizado, es muy sencillo escribir aplicaciones cuyo código se pueda reutilizar si más adelante tenemos que cambiar de motor de Base de Datos o bien si queremos permitir que dicha aplicación pueda conectarse a más de un SGBD de forma que el usuario no tenga porque comprometerse a un SGBD concreto si la adquiere o quiere ponerla en marcha.

Operaciones básicas con JDBC

Conectando con la Base de Datos

La conexión con la Base de Datos es la única parte de la programación de aplicaciones con Bases de Datos a través de JDBC que depende directamente del SGBD que se vaya a utilizar. No es un cambio muy grande puesto que simplemente hay que seleccionar el Driver adecuado (que depende de cada SGBD) y la cadena de conexión (que también dependerá). El resto del código para la conexión es el mismo por lo que será muy fácil que esta parte la podemos incluir dentro de una sentencia condicional si queremos que nuestra aplicación soporte varios SGBDs diferentes. Simplemente dependiendo del driver que se haya seleccionado (por ejemplo en la instalación) la aplicación deberá cargar el driver (en nuestro caso utilizamos el driver para la versión 8 de MySQL) y cadena de conexión adecuado.

A continuación, se pueden ver dichas similitudes para dos casos: el primero el de conecta con una Base de Datos en MySQL y el segundo en PostgreSQL.

Conectando con una Base de Datos de MySQL

```
. . .
Connection conexion = null;

try {
  Class.forName("com.mysql.cj.jdbc.Driver");
  conexion = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/basededatos",
    "usuario", "contraseña");
} catch (ClassNotFoundException cnfe} {
  cnfe.prinStackTrace();
} catch (SQLException sqle) {
  sqle.prinStackTrace();
}
. . .
```

Conectando con una Base de Datos de PostgreSQL

```
. . .
Connection conexion = null;
```

```
try {
  Class.forName("org.postgresql.Driver");
  conexion = DriverManager.getConnection(
    "jdbc:postgresql://localhost:5432/basededatos",
    "usuario", "contraseña");
} catch (ClassNotFoundException cnfe} {
  cnfe.prinStackTrace();
} catch (SQLException sqle) {
  sqle.prinStackTrace();
} catch (InstantiationException ie) {
  ie.prinStackTrace();
} catch (IllegalAccessException iae) {
  iae.prinStackTrace();
}
. . .
```

Conectando con una Base de Datos de Oracle

```
. . .
Connection conexion = null;

try {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  conexion = DriverManager.getConnection(
    "jdbc:oracle:thin:@//localhost:1521/xe",
    "usuario", "contraseña");
} catch (ClassNotFoundException cnfe} {
  cnfe.prinStackTrace();
} catch (SQLException sqle) {
  sqle.prinStackTrace();
} catch (InstantiationException ie) {
  ie.prinStackTrace();
} catch (IllegalAccessException iae) {
  iae.prinStackTrace();
}
. . .
```

> *Ejercicio.* *¿Cómo podríamos hacer una aplicación donde el usuario/administrador pueda seleccionar qué tipo de SGBD quiere utilizar?*

Desconectar de la Base de Datos

A la hora de desconectar, basta con cerrar la conexión, que será la misma operación independientemente del driver utilizado.

```
. . .
try {
  conexion.close();
  conexion = null;
} catch (SQLException sqle) {
  sqle.printStackTrace();
}
. . .
```

Insertar datos

```
. . .
String sentenciaSql = "INSERT INTO productos (nombre, precio) VALUES (?, ?)";
PreparedStatement sentencia = null;

try {
  sentencia = conexion.prepareStatement(sentenciaSql);
  sentencia.setString(1, nombreProducto);
  sentencia.setFloat(2, precioProducto);
  sentencia.executeUpdate();
} catch (SQLException sqle) {
  sqle.printStackTrace();
} finally {
  if (sentencia != null)
    try {
      sentencia.close();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
    }
}
. . .
```

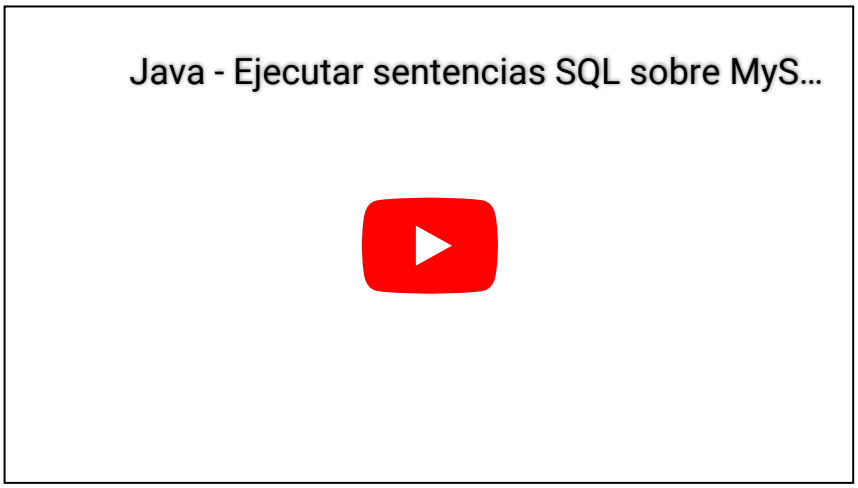
Modificar datos

```
. . .
String sentenciaSql = "UPDATE productos SET nombre = ?, precio = ? " +
                      "WHERE nombre = ?";
PreparedStatement sentencia = null;

try {
  sentencia = conexion.prepareStatement(sentenciaSql);
  sentencia.setString(1, nuevoNombreProducto);
  sentencia.setFloat(2, precioProducto);
  sentencia.setString(3, nombreProducto);
  sentencia.executeUpdate();
} catch (SQLException sqle) {
  sqle.printStackTrace();
} finally {
  if (sentencia != null)
    try {
      sentencia.close();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
    }
}
. . .
```

Eliminar datos

```
. . .
String sentenciaSql = "DELETE productos WHERE nombre = ?";
PreparedStatement sentencia = null;

try {
  sentencia = conexion.prepareStatement(sentenciaSql);
  sentencia.setString(1, nombreProducto);
  sentencia.executeUpdate();
} catch (SQLException sqle) {
  sqle.printStackTrace();
} finally {
  if (sentencia != null)
    try {
      sentencia.close();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
```

```
    }
}
. . .
```



Consultas

```
. . .
String sentenciaSql = "SELECT nombre, precio FROM productos";
PreparedStatement sentencia = null;
ResultSet resultado = null;

try {
  sentencia = conexion.prepareStatement(sentenciaSql);
  resultado = sentencia.executeQuery();
  while (resultado.next()) {
    System.out.println("nombre: " + resultado.getString(1));
    System.out.println("precio: " + resultado.getFloat(2));
  }
} catch (SQLException sqle) {
  sqle.printStackTrace();
} finally {
  if (sentencia != null)
    try {
      sentencia.close();
      resultado.close();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
    }
}
. . .
```

Hay que tener en cuenta que el objeto *ResultSet* es el cursor que contiene el resultado de la consulta y, al recorrerlo, podemos acceder a cualquier columna de dicho resultado indicando el número de ésta, siempre teniendo en cuenta que la posición 0 se reserva al número de la fila en el resultado de la consulta.

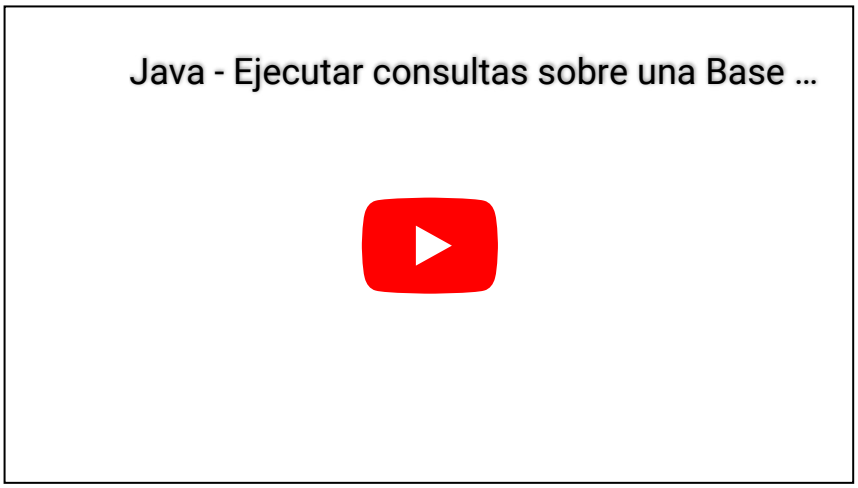
También se pueden parametrizar la consulta, tal y como se muestra en el siguiente ejemplo donde se mostrará la información de los productos de un determinado precio.

```
. . .
String sentenciaSql = "SELECT nombre, precio FROM productos " +
                      "WHERE precio = ?";
PreparedStatement sentencia = null;
ResultSet resultado = null;

try {
  sentencia = conexion.prepareStatement(sentenciaSql);
  sentencia.setFloat(1, filtroPrecio);
  resultado = sentencia.executeQuery();
  while (resultado.next()) {
    System.out.println("nombre: " + resultado.getString(1));
```

```
    System.out.println("precio: " + resultado.getFloat(2));
  }
} catch (SQLException sqle) {
  sqle.printStackTrace();
} finally {
  if (sentencia != null)
    try {
      sentencia.close();
      resultado.close();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
    }
}
. . .
```

En el caso de las funciones agregadas, podremos tener en cuenta que sólo van a devolver un valor, por lo que no será necesario preparar el código para recorrer el cursor. Podremos acceder directamente a la primera fila del *ResultSet* y mostrar el resultado, tal y como se muestra en el siguiente ejemplo:

```
. . .
String sentenciaSql = "SELECT COUNT(*) FROM productos";
PreparedStatement sentencia = null;
ResultSet resultado = null;

try {
  sentencia = conexion.prepareStatement(sentenciaSql);
  resultado = sentencia.executeQuery();
  resultado.next();
  System.out.println("Cantidad de productos: " + resultado.getInt(1);
} catch (SQLException sqle) {
  sqle.printStackTrace();
} finally {
  if (sentencia != null)
    try {
      sentencia.close();
      resultado.close();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
    }
}
. . .
```



Transacciones

En el ámbito de las Bases de Datos, una transacción es cualquier conjunto de sentencias SQL que se ejecutan como si de una sola se tratara. La idea principal es poder ejecutar varias sentencias, que están relacionadas de alguna manera, de forma que si cualquiera de ellas fallara o produjera un error, no se ejecutara ninguna más e incluso se deshicieran todos los cambios que hayan podido efectuar las que ya se habían ejecutado dentro de la misma transacción.

Para ello, tendremos que incorporar tres nuevas instrucciones a las que ya veníamos utilizando hasta ahora. Una instrucción para indicar que comienza una transacción (`conexion.setAutoCommit(false)`), otra para indicar cuando termina (`conexion.commit()`) y otra para indicar que la transacción actual debe abortarse y todos los cambios hasta el momento deben ser restaurados al estado anterior (`conexion.rollback()`).

```
. . .
/* Estos valores vendrán dados por el usuario al introducirlos o seleccionarlos
 * en el interfaz de la aplicación
 */
String nombreProducto = . . .;
float precioProducto = . . .;
int idCategoria = . . .;
// El id del producto que vamos a registrar aún no se conoce

String sqlAltaProducto = "INSERT INTO productos (nombre, precio) VALUES (?, ?)";
String sqlRelacionProducto =
       "INSERT INTO producto_categoria(id_producto, id_categoria) " +
       "VALUES (?, ?)";

try {
  //Inicia transacción
  conextion.setAutoCommit(false);

  PreparedStatement sentenciaAltaProducto =
                    conexion.prepareStatement(sqlAltaProducto,
                    PreparedStatement.RETURNS_GENERATED_KEYS);
  sentenciaAltaProducto.setString(1, nombreProducto);
  sentenciaAltaProducto.setFloat(2, precioProducto);
  sentencia.executeUpdate();

  // Obtiene el id del producto que se acaba de registrar
  ResultSet idGenerados = sentenciaAltaProducto.getGeneratedKeys();
  idsGenerados.next();
  int idProducto = idsGenerados.getInt(1);
  sentenciaAltaProducto.close();

  PreparedStatement sentenciaRelacionProducto =
                    conexion.prepareStatement(sqlRelacionProducto);
  sentenciaRelacionProducto.setInt(1, idProducto);
  sentenciaRelacionProducto.setInt(2, idCategoria);
  sentenciaRelacionProducto.executeUpdate();

  // Valida la transacción
  conexion.commit();
} catch (SQLException sqle) {
  sqle.printStackTrace();
} finally {
  if (sentencia != null)
    try {
      sentencia.close();
      resultado.close();
    } catch (SQLException sqle) {
      sqle.printStackTrace();
    }
}
. . .
```

Procedimientos y funciones almacenadas

Procedimientos almacenados

La ejecución de procedimientos almacenados sigue la misma estructura que cualquiera de las sentencias *SQL* de los ejemplos anteriores, con la excepción de que usaremos la clase `CallableStatement` para representar al procedimiento y el método `execute()` de la misma para ejecutarlo.

```
. . .
String sentenciaSql = "call eliminar_todos_los_productos()";
CallableStatement procedimiento = null;

try {
  procedimiento = conexion.prepareCall(sentenciaSql);
  procedimiento.execute();
} catch (SQLException sqle) {
  . . .
}
. . .
```

Y en el caso de que el procedimiento almacenado tuviera algún parámetro:

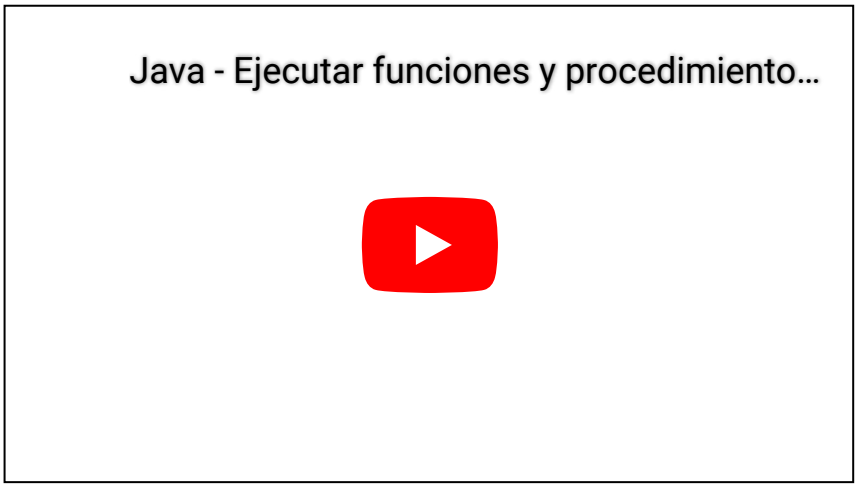
```
. . .
String sentenciaSql = "call eliminar_producto(?)";
CallableStatement procedimiento = null;

try {
  procedimiento = conexion.prepareCall(sentenciaSql);
  procedimiento.setString(1, nombreProducto);
  procedimiento.execute();
} catch (SQLException sqle) {
  . . .
}
. . .
```

Funciones almacenadas

En el caso de las funciones almacenadas, para su ejecución seguiremos la misma estructura que hemos visto en el caso de las consultas SQL para el caso concreto de las funciones agregadas, ya que nuestras funciones nos devolverán siempre un solo valor (o null en el caso de que no devuelvan nada).

```
. . .
String sentenciaSql = "SELECT get_precio_mas_alto()";
PreparedStatement sentencia = null;
ResultSet resultado = null;

try {
```

```
  sentencia = conexion.prepareStatement(sentenciaSql);
  resultado = sentencia.executeQuery();
  resultado.next();

  if (resultado.wasNull)
    System.out.println("No hay datos");
  else {
    float precioMasAlto = resultado.getFloat(1);
    System.out.println("El producto más caro vale " + precioMasAlto);
  }
} catch (SQLException sqle) {
  . . .
}
. . .
```



Ejercicios

1. Implementa el sistema de login de una aplicación (usuario, contraseña y nivel de acceso) teniendo en cuenta que la información de los usuarios será una tabla de la base de datos

2. Realiza una aplicación que cargue el contenido de una tabla de una base de datos en un componente `JTable`

3. Realiza una aplicación que cargue el contenido de una tabla de Productos (#id, nombre, descripcion y precio) de una base de datos en un `JList`, mostrando el nombre y precio (formateado como €)

Ejercicios Examen

1. Escribe un programa que conecte con una Base de Datos MySQL de forma totalmente transparente para el usuario (Script de la Base de Datos)
    a. Añade la opción de listar el contenido de una tabla `granjeros` en un `JList`, mostrando solamente su nombre y el dinero, ordenado por este último campo
    b. Añade la opción de registrar nuevas filas en la tabla granjeros. Se deberá permitir registrar un granjero sólo indicando el nombre por parte del usuario, aunque los demás campos sean obligatorios en la Base de Datos
    c. Añade la opción de subir el nivel de los granjeros (incrementar su nivel en una unidad)
    d. Añade la opción de buscar granjeros introduciendo su nombre o descripción
    e. Añade la opción de eliminar filas de la tabla `granjeros` indicando el `id` de la fila que se quiere eliminar
    f. Añade la opción de listar el contenido de la tabla `granjeros` en un `JList`

Proyectos de ejemplo

Todos los proyectos de ejemplo de esta parte están en el repositorio java-jdbc [http://www.github.com/codeandcoke/java-jdbc] y en el repositorio de JavaFX [https://github.com/codeandcoke/java-javafx] de GitHub.

Los proyectos que se vayan haciendo en clase estarán disponibles en el repositorio datos-ejercicios [http://www.github.com/codeandcoke/datos-ejercicios], también en GitHub.

Para manejaros con Git recordad que tenéis una serie de videotutoriales en La Wiki de Entornos de Desarrollo [https://entornos-desarrollo.codeandcoke.com/apuntes:git]

Práctica 2.1

Objetivos

Desarrollar una aplicación que conecta con una Base de Datos Relacional

Enunciado

Siguiendo el mismo diseño de la aplicación de las práctica 1.1, se deberá implementar una aplicación que conecte con una Base de Datos en MySQL, según los requisitos que se enumeran a continuación

Requisitos (1 pto cada uno)

- La aplicación deberá conectar con una Base de Datos de forma transparente para el usuario, de forma que los datos de conexión puedan configurarse en un fichero a parte (fichero *properties*)
- El usuario tiene que poder dar de alta, modificar y eliminar datos
- Mostrar todos los registros en un `JList`
- Implementar un sistema de autenticación de usuarios para la aplicación
- Añadir alguna forma de búsqueda

Otras funcionalidades (1 pto cada una)

- Ejecución de dos funciones almacenadas (a través del interfaz) realizando alguna tarea útil para la aplicación
- Ejecución de dos procedimientos almacenados (a través del interfaz) realizando alguna tarea útil para la aplicación
- Uso de transacciones para realizar alguna operación compleja (de alta, baja o modificación)
- Que la aplicación sea también capaz de conectar con una Base de Datos en PostgreSQL
- Añadir soporte para multiusuario, implementando lo necesario para que varios usuarios simultáneos puedan trabajar con la aplicación sin que se produzcan problemas (por ejemplo, que dos usuarios estén modificando el mismo elemento)
- Utilizar la herramienta Git (y GitHub) durante todo el desarrollo de la aplicación. Utilizar el gestor de *Issues* para los problemas/fallos que vayan surgiendo
- Añadir una opción al usuario que permita recuperar el último elemento borrado
- Añadir una opción a la aplicación que permita eliminar todos los datos del programa

---

1)
http://www.oracle.com/technetwork/java/index-136695.html [http://www.oracle.com/technetwork/java/index-136695.html]
