111mil Programadores</>

Base de Datos

Apunte 3

Prof. Germán C. Basisty

basisty.german@gmail.com

LANÚS MUNICIPIO

FUNDACIÓN DIIEJ
Desarrollo para la Inclusión e Igualdad en el Empleo Juvenil

Índice de contenidos

JDBC

Java Database Connectivity, más conocida por sus siglas **JDBC**, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación **Java**, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto **SQL** del modelo de base de datos que se utilice.

java.sql

JDBC ofrece el paquete java.sql, en el que existen clases muy útiles para trabajar con bases de datos.

Clase	Descripción
DriverManager	Utilizada para cargar un driver
Connection	Utilizada para establecer conexiones con las bases de datos
Statement	Utilizada para ejecutar sentencias SQL y enviarlas a las BBDD
PreparedStatement	Utilizada para ejecutar consultas de forma repetitiva
ResultSet	Utilizada para almacenar el resultado de la consulta

Implementación

1) Importar las clases necesarias:

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
```

2) Crear un objeto de la clase **Connection** para gestionar la comunicación con la RDBM.

```
Connection dbConnection = null;
```

3) Para configurar la conexión a la base de datos, necesitamos la URL de conexión, que se declara de la siguiente manera:

*jdbc:postgresql://**direccion_ip**:**puerto**/**nombre_db***

Una práctica habitual es guardar la URL en un objeto tipo String, por ejemplo:

```
String url = "jdbc:postgresql://127.0.0.1:5432/my_database";
```

4) Cargar el driver de PostgreSQL:

```
Class.forName("org.postgresql.Driver");
```

5) Conectar con la base de datos:

```
dbConnection = DriverManager.getConnection(url, username, password);
```

donde ***dbConnection*** es nuestro objeto de la clase *Connection*, **url** es el *String* que contiene la URL de conexión, **username** y **password** son *Strings* que contienen la información de nombre de usuario y contraseña respectivamente.
Tanto la carga del driver como la conexión a la base de datos pueden fallar; por eso es necesario encerrar los respectivos comandos dentro de un bloque try - catch para que levante una excepción en el caso que sea necesario:

```
try {
        Class.forName("org.postgresql.Driver");
        dbConnection = DriverManager.getConnection(url, username, password);
} catch (Exception e) {
        System.out.println("ERROR: " + e);
}
```

try intenta ejecutar su bloque de código, y si todo sale bien el flujo del programa continúa. Sin embargo, si se produce algún error (excepción), este error es gestionado por el **catch** y almacenado en el objeto **e** de la clase *Exception*, y se lo trata en el bloque de código del propio **catch**; en este caso, se imprime la cadena de texto "ERROR" y se concatena el mensaje de error que está en el objeto **e**.

6) Una vez establecida la conexión, para ejecutar consultas necesitaremos de un objeto de la clase **Statement**, que debe ser inicializado utilizando el método **createStatement()** de la clase **Connection**:

```
Statement Query;
```

y luego:

```
Query = dbConnection.createStatement();
```

La clase **Statement** tiene un método **execute(String)** que recibe un String como parámetro, que debe contener la sentencia SQL a ejecutar. Una vez ejecutada la consulta, se debe llamar al método **close()** del objeto statement para cerrar la consulta correctamente:

```
Query.execute("INSERT INTO tabla VALUES(valor1, valor2)");
Query.close();
```

Como la inicialización de la consulta o la ejecución de la misma pueden fallar, deben encerrarse en un bloque **try - catch**, por ejemplo:

```
String queryString = "INSERT INTO persona VALUES(25969243, 'German Basisty')";
Statement Query;

 try {
        Query = dbConnection.createStatement();
        Query.execute(queryString);
        Query.close();
} catch (Exception e) {
        System.out.println("ERROR: " + e);
}
```

7) Para las consultas que devuelven algún resultado, es necesario implementar un objeto de la clase **ResultSet** que nos permita gestionar los datos retornados a través de una tabla virtual, para luego iterar sobre las tuplas virtuales según sea necesario. A su vez, se debe utilizar el método **executeQuery(String)** de la clase Statement.

```
ResultSet Result = null;

try {
        Query = dbConnection.createStatement();
        Result = Query.executeQuery("SELECT col1, col2 FROM tabla");

        while(Result.next()) {
                System.out.print(Result.getString(1) + " | " + Result.getString(2));
                System.out.println();
        }

        Query.close();
} catch (Exception e) {
        System.out.println("ERROR: " + e);
}
```

ResultSet cuenta con el método **next()** para iterar de forma secuencial por las tuplas obtenidas. Los atributos de las tuplas están numerados comenzando por el 1.
ResultSet tiene métodos específicos para obtener el valor de cada atributo en un formato específico:

- **getString(n)** para devolver el atributo n como String
- **getInt(n)** para devolver el atributo n como entero
- **getFloat(n)** para devolver el atributo n como float
- **getDate(n)** para devolver el atributo n como fecha
- **etc**

Ejemplo

- En una base de datos “prueba” se crea una tabla persona, con las columnas dni de tipo entero y nombre de tipo texto:

```
CREATE TABLE persona (
        dni             integer,
        nombre          text
);
```

- Desarrollar una aplicación java que permita conectarse a la base de datos, insertar registros, y luego recuperar todas las tuplas (archivo jdbcExample.java):

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

public class jdbcExample {
   public static void main(String[] args) {
        Connection dbConnection = null;
        String url = "jdbc:postgresql://127.0.0.1:5432/prueba";
        String username = "nombreDeUsuario";
        String password = "1234";

        try {
                Class.forName("org.postgresql.Driver");
                dbConnection = DriverManager.getConnection(url, username, password);
        } catch (Exception e) {
                System.out.println("ERROR: " + e);
        }

        String queryString = "INSERT INTO persona VALUES(13242424555, 'Juan Perez')";
        Statement Query;

        try {
                Query = dbConnection.createStatement();
                Query.execute(queryString);
                Query.close();
        } catch (Exception e) {
                System.out.println("ERROR: " + e);
        }
```

```
        queryString = "SELECT * FROM persona";
        ResultSet Result = null;

        try {
                Query = dbConnection.createStatement();
                Result = Query.executeQuery(queryString);

                while(Result.next()) {
                        System.out.print(Result.getString(1) + " | " + Result.getString(2));
                        System.out.println();
                }

                Query.close();
        } catch (Exception e) {
                System.out.println("ERROR: " + e);
        }
    }
}
```

Compilación

Para poder compilar el código anterior, es necesario descargar el **jdbc** de PostgreSQL del sitio oficial:

https://jdbc.postgresql.org/download/postgresql-42.1.4.jar

Y guardarlo en el mismo directorio en donde se encuentra el .java

El comando de compilación es:

```
javac -cp postgresql-42.1.4.jar jdbcExample.java
```

donde el modificador -cp incluye el archivo **postgresql-42.1.4.jar** al *classpath* en la compilación en curso. El *classpath* es una variable de entorno que indica en que directorios buscar clases java necesarias para la compilación o ejecución de un programa.

Para ejecutar el programa una vez compilado:

```
java -cp .:./postgresql-42.1.4.jar  jdbcExample
```