Universidad del Valle

Facultad de Ingeniería

Escuela de Ingeniería de Sistemas y Computación

Bases de Datos I - Laboratorio # 4: Conexión de Programas Java con DBMS

Entrega: Diciembre 06 de 2017

NORMAS PARA LA ENTREGA DE LOS LABORATORIOS

1. *Coloque su nombre y dirección de correo electrónico preferida en la carátula de su informe, así como el nombre del profesor a quien le entrega el informe. Los informes deben estar firmados por todos los integrantes del grupo, debajo del siguiente código de ética:*
   << Al firmar el presente informe, aseguramos que nuestro grupo NO ha copiado de nadie, ni dado copia a nadie, la solución que a continuación presentamos>>
2. *Presente una copia del código fuente de todos sus programas y la evidencia de la ejecución. Inclúyalas en la carpeta del laboratorio correspondiente. .*
3. *Organice su trabajo en carpetas, en lo posible una por cada punto del laboratorio.*
4. ***Lugar, Plazo y Medio de Entrega**: Todos los archivos que se soliciten en el informe (.java y .pdf) deben ser colocados en un archivo comprimido (.zip) del laboratorio que se está resolviendo. Presente su informe en la fecha indicada, descargándolo en el Campus Virtual.*
5. *Durante el curso **no se recibirán informes de laboratorio enviados por correo electrónico**.*
6. ***Recuerde que un requerimiento del curso es utilizar el DBMS Postgres**.*

Objetivos:

1. Identificar los aspectos generales para realizar una aplicación de Bases de Datos.
2. Conectarse a una base de datos utilizando una conexión JDBC
3. Implementar una pequeña aplicación en Java con un enlace al manejador de bases de datos.

Metodología:

Se debe elaborar un informe del trabajo realizado, deberá describir cada una de las actividades y responder a las preguntas. El laboratorio debe ser desarrollado en grupos de máximo 3 personas.

Usted podrá necesitar leer los documentos de ayuda sobre Postgres, para realizar esta parte del proyecto. Realice las búsquedas por internet. La aplicación deberá ser desarrollada en Java.

Aspectos Básicos I

JDBC - *Generalidades*

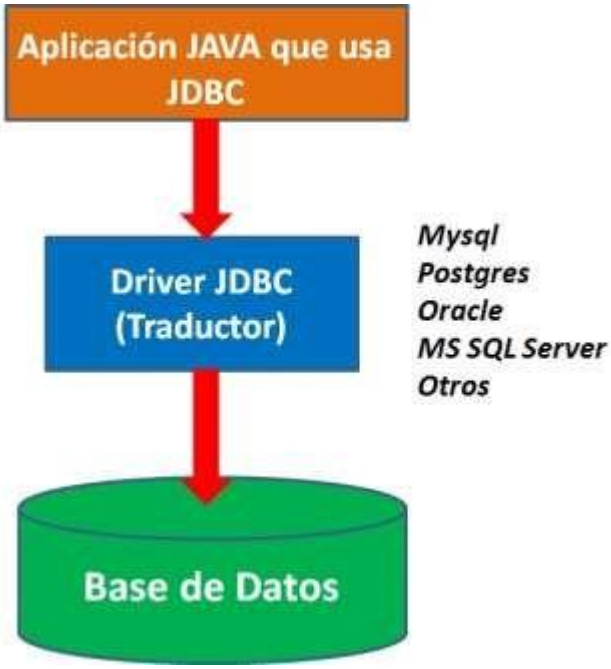
JDBC es el acrónimo de *Java Database Connectivity* basado en la interfaz de nivel de comandos de SQL, que ofrece a los programadores de aplicaciones una interfaz de programación de aplicaciones, independientes de la base de datos (API). Es decir, una aplicación JAVA que usa JDBC para acceder a un DBMS accederá a otro DBMS sin modificaciones, siempre que el nuevo sistema de bases de datos soporte JDBC.

Cada fabricante de bases de datos es responsable de implementar un controlador JDBC que permita interactuar con su sistema específico. Así, además de la independencia de la base de datos, otro fuerte de JDBC es la facilidad para conseguir los controladores de gran variedad de DBMS como lo son: MySQL, PostgreSQL, Oracle, DB2, Informix, entre otros.

¿Que es JDBC?

Son las siglas de **Java DataBase Connectivity**, un **API** que a partir de un conjunto de clases, permite utilizar una serie de métodos para operar sobre una base de datos. Los métodos utilizados dirigen todas las peticiones hacia un software intermediario conocido como **Driver JDBC**, el cual se encarga de traducir los llamados de los métodos a órdenes nativas del gestor de Base de datos utilizado. Para el ejemplo una imagen, de cómo funciona JDBC.

Clase / Interfaz	**Función**
DriverManager	Establece conexión con la base de datos a través del Driver
Connection	Representa una conexión con la base de datos
Statement	Ejecución de consultas SQL
PreparedStatement	Ejecución de consultas SQL preparadas y procedimientos almacenados
ResultSet	Manipulación de registros en consultas de tipo **Select**
ResultSetMetadata	Proporciona información sobre la estructura de los datos



¿Que es un driver?

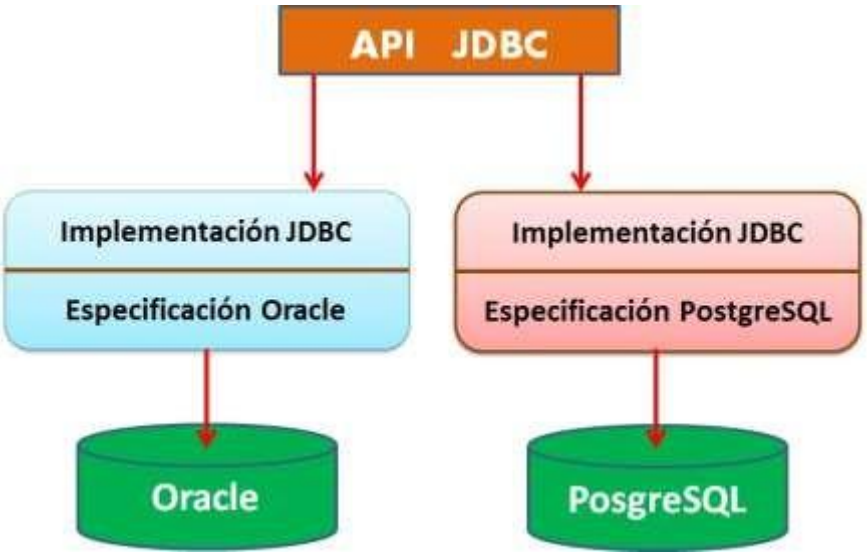
(También conocido como controlador), un Driver es un software que permite al sistema operativo o aplicación, interactuar con un periférico (base de datos). Para ajustar esta definición más a nuestro caso, un **Driver JDBC** es una clase Java que implementa toda la funcionalidad del **API JDBC**, proporcionando la comunicación entre la aplicación y la base de datos.

Normalmente son los fabricantes de bases de datos quienes distribuyen los diferentes Driver JDBC.

Un Driver JDBC se compone de dos capas o interfaces:

Capa de aplicación: Es la parte del driver que interactúa con la aplicación. Todos los driver JDBC, independientemente del gestor de bases de datos para el que se haya diseñado, nos proporcionan la misma interfaz de aplicación.

Capa de base de datos: Es la que interactúa con la base de datos, por lo que es específica para cada base de datos.



¿Qué se necesita para establecer una conexión con JDBC?
Solo es cuestión de aplicar los siguientes cuatro pasos:

1. **Conexión con la base de datos**
2. **Ejecución de consultas**
3. **Manipulación de registros**
4. **Desconexión de la base de datos**

Conexión con la base de datos

Conexión La conexión con Postgres por medio de JDBC requiere de cuatro parámetros en el momento de realizar la conexión, dichos parámetros son:

- Driver (driver)
- Url (name_dir)
- Login (user_db)
- Password (pass_db)

Driver: El parámetro driver es utilizado a la hora de utilizar la instrucción `class.forName(driver)`. Con este método lo que hacemos es cargar dinámicamente el driver para que sea reconocido por java a la hora de enviar las instrucciones al SMBD.
Url: El parámetro `Url` especifica la ubicación física del archivo `.jar` que contiene el código del driver, este archivo `.jar` contiene varias clases que implementan las diferentes estructuras de información que posea el SMBD.
Login: El parámetro `login` especifica el login del usuario que se desea conectar con el SMBD, por ejemplo el `login` puede ser `"perezJ"`.
Password: El parámetro `password`, como su nombre lo especifica es el password que utiliza el usuario para conectarse con el SMBD, el `password` puede ser `"1234"` (password para usuario `perezJ`).

Los tres últimos parámetros `Url, Login` y `Password` son usados en el método
`DriverManager.getConnection(name_dir, user_db, pass_db);`

Ejecución de consultas

Después de realizar la conexión podemos realizar las tres operaciones típicas que se realizan con un SMBD: consultar información, insertar información y borrar información.

La clase `Statement`

Un objeto `Statement` se usa para enviar sentencias SQL a la base de datos. Hay tres tipos de objetos `Statement`, todos los cuales actúan como contenedores para la ejecución de sentencias en una conexión dada:

Statement, PreparedStatement que hereda de **Statement** y **CallableStatement** que hereda de **PreparedStatement**. Estas están especializadas para enviar tipos particulares de sentencias SQL. Un objeto **Statement** se usa para ejecutar una sentencia SQL simple sin parámetros. Un objeto **PreparedStatement** se usa para ejecutar sentencias SQL precompiladas con o sin parámetros IN; y un objeto **CallableStatement** se usa para ejecutar un procedimiento de base de datos almacenado.

Una vez establecida la conexión con una base de datos particular, esta conexión puede usarse para enviar sentencias SQL. Un objeto **Statement** se crea mediante el método de **Connection createStatement**, como podemos ver en el siguiente fragmento de código:

```
Statement stmt = con.createStatement();
```

La sentencia SQL que será enviada a la base de datos es alimentada como un argumento a uno de los métodos de ejecución del objeto **Statement**. Por ejemplo:

```
ResultSet rs = sta.executeQuery(sql)
```

La interfase **Statement** nos suministra tres métodos diferentes para ejecutar sentencias SQL, **executeQuery, executeUpdate y execute**. El método a usar está determinado por el producto de la sentencia SQL.

El método **executeQuery** se utiliza para sentencias que producen como resultado un único **ResultSet** tal como las sentencias **SELECT**.

El método **executeUpdate** se usa para ejecutar sentencias **INSERT, UPDATE ó DELETE** así como sentencias SQL DDL (Data Definition Language) como **CREATE TABLE o DROP TABLE**.

El método **execute** se usa para ejecutar sentencias que devuelven más de un **ResultSet**, más que un update count o una combinación de ambos. Se considera una característica avanzada.

La clase PreparedStatement()

Una PreparedStatement es un tipo especial de sentencias SQL de base de datos que se deriva de una clase más general, la clase **Statement**.

La característica principal de un objeto **PreparedStatement** es que, al contrario que un objeto **Statement**, se le entrega una sentencia SQL cuando se crea. La ventaja de esto es que en la mayoría de los casos, esta sentencia SQL se enviará al controlador de la base de datos inmediatamente, donde será compilado. Como resultado, el objeto **PreparedStatement** no sólo contiene una sentencia SQL, sino una sentencia SQL que ha sido precompilada. Esto significa que cuando se ejecuta la **PreparedStatement**, el controlador de base de datos puede ejecutarla sin tener que compilarla primero.

Para realizar la consulta de información contenida en diferentes tablas de la BD, es decir hacer un SELECT, se deben ejecutar una serie de métodos pertenecientes a las clases **Statement** o **ResultSet**

Manipulación de registros

La clase ResultSet

El objeto **ResultSet** proporciona varios métodos para obtener los datos de columna correspondientes a una fila. Todos ellos tienen el formato **get<*Tipo*>**, siendo <*Tipo*> un tipo de datos Java™. Algunos ejemplos de estos métodos son **getInt, getLong, getString, getTimestamp y getBlob**. Casi todos estos métodos toman un solo parámetro, que es el índice que la columna tiene dentro del **ResultSet** o bien el nombre de la columna.

Las columnas de **ResultSet** están numeradas, empezando por el 1. Si se emplea el nombre de la columna y hay más de una columna que tenga ese mismo nombre en el **ResultSet**, se devuelve la primera.

Interface ResultSetMetaData

Cuando se llama al método **getMetaData** en un objeto **ResultSet**, el método devuelve un objeto **ResultSetMetaData** que describe las columnas de ese objeto **ResultSet**. En los casos en que la sentencia

SQL que se va a procesar no se conoce hasta el momento de la ejecución, puede utilizarse **ResultSetMetaData** para determinar cuál de los métodos **get** hay que emplear para recuperar los datos.

Metadatos significa datos sobre datos, es decir, podemos obtener más información de los datos.

Si tiene que obtener metadatos de una tabla como el número total de columnas, nombre de las columnas, tipo de cada columna, etc., la interfaz **ResultSetMetaData** es útil porque proporciona métodos para obtener metadatos del objeto **ResultSet**. Algunos de los métodos de la interfaz **ResultSetMetaData:**

Método	Descripción
public int getColumnCount() throws SQLException	Retorna el número total de columnas en el objeto ResultSet.
public String getColumnName(int index) throws SQLException	Retorna el nombre de la columna especificada en el índice
public String getColumnTypeName(int index) throws SQLException	Retorna el nombre del tipo de columna especificada en el índice
public String getTableName(int index) throws SQLException	Retorna el nombre de la tabla para la columna especificada en índice

Desconexión de la base de datos

Se debe ejecutar el método **close()** a los diferentes objetos: **ResultSet, Statement, Connection**:

```
rs.close();
st.close();
con.close();
```

El método **close()** se emplea para para cerrar o destruir el objeto. Esto significa que hasta que se llame en forma explícita a su método **close()** para cerrar la conexión, la conexión a la base de datos permanecerá activa.

SELECT: *El* orden en que se ejecuta una consulta SELECT SQL con el JDBC es el siguiente:

a) Se ejecuta el método ***executeQuery*** perteneciente a la clase **Statement**, el cual recibe un parámetro de tipo **String** el cual es la consulta SQL que buscamos ejecutar
b) La ejecución del método anterior devuelve un objeto ***ResulSet***
c) Dentro de un ciclo se ejecuta una llamada al método ***next()*** del objeto ***ResultSet*** el cual devuelve una a una las tuplas que concordaron con el **SELECT** que ejecutamos, teniendo cuidado de pasar el nombre correcto de los atributos a obtener.

Ejemplo de un **SELECT**:
Se quiere obtener un listado de todos los trabajadores de la empresa: Sentencia SQL:

```
SELECT nombre, apellido FROM persona;
```

Sentencia JDBC:

```
ResultSet result = stmt.executeQuery("SELECT * FROM usuario;");
```

de este modo se obtiene el resultado de la consulta dentro de la variable **result** y solo quedaría examinar lo que se encuentra en ella:

1. Ir a la primera fila de datos:

```
result.next();
```

2. Obtener los datos de la columna de cada fila:

```
String nombre = result.getString("nombre");
String apellido = result.getString("apellido");
// También pueden usarse getInt(), getDate(), entre otros
```

Luego se puede imprimir la información fácilmente de esta forma:

```
System.out.println("Empleado: "+nombre+" "+apellido); //También puede usar JOptionPane
```

Nota: No debe olvidar que de esta forma imprimirá solo la primera fila del resultado, en caso de que retorne más de un registro deberá iniciar el bloque declarando el **result.next** dentro de un ciclo que puede ser:

```
While (result.next()) {
      // Instrucciones . . .
}
```

INSERT: El orden en que se ejecuta una instrucción **INSERT** SQL, con el JDBC, es el siguiente:

1. Se realiza una llamada al método **executeUpdate** de la clase **Statement**, dicho método recibe un String el cual contiene el cuerpo de la instrucción **INSERT** que queremos realizar
2. La ejecución del método anterior devuelve un entero que nos indica si hubo o no un error en la instrucción **INSERT**
3. La ejecución del método **executeUpdate** puede generar excepciones las cuales deben ser atrapadas en un bloque **try - catch**

Ejemplo de un **INSERT**:

```
Statement insercion = con.createStatement();
Try {
     insercion.executeUpdate("insert into persona values  ('100500','020')");
}// fin del try
catch (SQLException e)  {
       System.err.println("fallo al tratar de insertar ");
}// fin del catch
```

DELETE: El orden en que se ejecuta una consulta DELETE SQL con el JDBC es el siguiente:

a) Se realiza una llamada al método **executeUpdate** de la clase **Statement**, dicho método recibe un **String** el cual contiene el cuerpo de la instrucción **DELETE** que queremos realizar

b) La ejecución del método anterior devuelve un entero que indica si hubo o no, error en la instrucción **DELETE**

c) La ejecución del método **executeUpdate** puede generar excepciones las cuales deben ser atrapadas en un bloque **try - catch**

Ejemplo de un **DELETE**

```
Statement borrado = con.createStatement();
Try {
    borrado.executeUpdate("DELETE FROM persona  WHERE id = '001' ");
}// Fin del try
catch (SQLException e)  {
    System.err.println("Error al tratar de Borrar una tupla de Persona");
}// Fin del catch
```

Trabajo

Para los puntos siguientes tome como referencia el esquema generado en el laboratorio No. 3 Problema del Curso: ***Alquiler de coches "Autos UV"***, realice una pequeña aplicación usando lo visto sobre JDBC de modo que le permita realizar una conexión a la base de datos y que en caso de no conectarse permita ver cuál es el problema en la conexión.

1. ***Opción de consulta:*** defina dos diferentes consultas (que incluyan subconsultas y operaciones join) de su base de datos y muéstrelas utilizando el JDBC. [5]
2. ***Opción de inserción:*** Muestre como se realiza la inserción de 10 registros diferentes en **Vehiculo usando JDBC, ciclos** y una instrucción **INSERT**, muestre las tuplas insertadas en un **JTextArea**. Adjunte el método java que hace esta tarea.[5]
3. ***Opción de modificación:*** Modifique al menos cuatro de los registros de la tabla **Cliente usando JDBC, ciclos** y una instrucción **UPDATE**, muestre las tuplas modificadas en el **JTextArea**. Adjunte el método java que hace esta tarea. [5]
4. ***Opción de borrado:*** Borre algunos de los registros de la tabla `Marca` utilizando JDBC, **ciclos** y una instrucción **DELETE**, muestre las tuplas borradas en el **JTextArea**. Adjunte el método java que hace esta tarea.[5]
5. ***Opción de listado:*** Muestre el estado actual (en el **JTextArea**) de las tuplas en las tablas trabajadas, usando funciones JDBC. Adjunte el método java que hace esta tarea. [6]
6. ***Opción de carga de datos:*** Cree un programa, usando el lenguaje java, que genere datos aleatorios en un archivo plano con más de 5000 registros. Utilizando este archivo, cargue estos datos en la tabla `Cliente`

usando una función (tal como lo hace el ORACLE BULK LOADER), usando el JDBC en una aplicación Java. Indique los tiempos de carga. Tenga en cuenta que parte de los registros generados pueden no cumplir con las restricciones de la tabla, por ejemplo, dos tuplas pueden tener el mismo valor en un atributo que es llave primaria; por esto, puede que sea necesario generar más de 5000 registros diferentes para que se produzcan al menos 5000 inserciones en una tabla. [7]

7. Cree un pequeño menú (usando la clase **`JMenu`**) que permita trabajar las opciones arriba indicadas en los puntos 1 a 6. [17]

Ejemplos de consultas:

- Obtener la información de los concesionarios que no venden los vehículos que venden los concesionarios de Cali. (Utilice los operadores **IN** o **EXISTS**, según el caso).
- Obtener el código del vehículo de aquellos Vehículos vendidos a clientes de 'Bogotá'. (Utilice los operadores **IN** o **EXISTS**, según el caso).
- Obtener los datos más relevantes del (los) cliente(s) que han adquirido la mayor cantidad de vehículos.
- Obtener los nombres de los concesionarios que distribuyen vehículos en una cantidad superior a 5.
- Obtener el código del vehículo de aquellos Vehículos que han sido adquiridos por un cliente de 'Bogotá' en un concesionario de 'Cali'. (Utilice los operadores **IN** o **EXISTS**, según el caso).
- Obtener la referencia, modelo y marca de aquellos Vehículos comprados en un concesionario de la misma ciudad que la del cliente que lo compra. (Utilice los operadores **IN** o **EXISTS**, según el caso).
- Obtener la información de los concesionarios que distribuyen todas las marcas de vehículos

¿Qué entregar?

Presente una copia impresa de:
(1) los pantallazos de muestra de su IGU
(2) el código completo para su aplicación, con los comentarios para algo no obvio, y
(3) el informe sobre su comprobación.

Usted puede escoger usar aplicación de Java o un Applet, pero la aplicación es más rápida y más flexible. Como usted conoce acerca del poder de Java, tiene la ventaja de haber visto los disparadores, usted debe completar aspectos que hagan falta en la aplicación. En particular, si hay una restricción de integridad que antes no se manejó, entonces manéjela ahora. Los datos que usted genere y cargue en su base de datos deben satisfacer todas las restricciones de integridad.

Usted debe incluir consultas y modificaciones entre sus opciones.

No se tendrá en cuenta las consultas simples ni listados triviales. Pregunte, si lo desea, la consulta o el listado, con el profesor antes de entregar el informe.

Recuerde que el estilo de la programación (como: comentarios, buena alineación o sangría de las instrucciones, los nombres de variables significativos y consistentes, etc.) es importante!!

Entrega: Diciembre 6 de 2017