Fundamento Bases de Datos
Facultad de Ciencias UNAM
M.I. Gerardo Avilés Rosas
gar@ciencias.unam.mx
Laboratorio:
Efraín Hipólito Chamú <*chamugauss@gmail.com* >

PRACTICA 12
ORACLE Y JDBC

1. Introducción

La base de datos no es un elemento aislado dentro de una organización, sino que representa un medio para compartir la información necesaria entre todos los componentes de la misma. Por ello es importante conocer medios que permitan esta actividad.

Se han desarrollado distintas tecnologías para obtener estos resultados, algunas sobresalen por su facilidad de uso, otras por sus características y unas más por ser simplemente soluciones a la medida. Destacan entre ellas: ODBC y JDBC.

- ODBC
  "ODBC" (Open DataBase Connectivity), es una interfaz basada en el lenguaje de programación C para las aplicaciones que manejan SQL, provee un medio consistente de comunicación con las bases de datos y de acceso con los metadatos de la base. Cada compañía provee controladores específicos o 'puentes' para su SMBD. Consecuentemente, gracias a ODBC y SQL, es posible establecer una conexión a una base de datos y manipularla en una forma estándar.
  Aun cuando SQL permite la manipulación de bases de datos de manera sencilla, éste no fue diseñado para ser un lenguaje de aplicación general; sino que fue especificado en un principio con la idea de interactuar con bases de datos. Independientemente de los últimos cambios al estándar actual, SQL2003, es común utilizar otro lenguaje de programación donde se encuentren incrustadas sentencias SQL que se envían al SMBD y posteriormente se procesen los resultados.
  El principal inconveniente, se refiere a la imposibilidad de escribir programas que se ejecuten en distintas plataformas, aun cuando el tópico de estandarización en la conectividad a bases de datos se ha tratado de resolver.
  Por ejemplo, si se escribe un programa cliente en C++, posiblemente sea necesario volver a codificarlo si se transporta a otra plataforma.

- JDBC
  "JDBC" (Java DataBase Connectivity), es un API de Java que permite al programador ejecutar instrucciones en SQL, enviarlas al servidor de bases de

datos y procesar los resultados obtenidos. Para que una aplicación pueda realizar operaciones en una base de datos, ha de tener una conexión con ella, que se establece a través de un controlador (driver), que convierte el lenguaje de alto nivel a sentencias de SQL y son recibidas por la base de datos. Es decir, las tres acciones principales que realizará JDBC son: establecer la conexión a una base de datos, ya sea remota o no; enviar sentencias SQL a esa base de datos y, por último, procesar los resultados obtenidos.

JDBC es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa escrito en Java acceder a sistemas de bases de datos de forma homogénea. La aplicación de Java debe tener acceso a un controlador JDBC adecuado y este controlador es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

La necesidad de JDBC, a pesar de la existencia de ODBC, viene dada porque ODBC es una interfaz escrita en lenguaje C, que al no ser un lenguaje portable, haría que las aplicaciones también pierdan portabilidad. Además, ODBC tiene el inconveniente de que se ha de instalar manualmente en cada máquina; al contrario que los drivers JDBC, que se encuentran incluidos en las plataformas Java (J2SE y J2EE).

Toda la conectividad de bases de datos de Java se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación de bases de datos. La especificación JDBC requiere que cualquier driver JDBC sea compatible con al menos, el estándar SQL92.

- Acceso con JDBC a bases de datos

  El API JDBC soporta dos modelos distintos de acceso a bases de datos, los modelos de dos y tres capas.

- Modelo de Dos Capas

  Este modelo se basa en que la conexión entre la aplicación Java (o el 'applet' que se ejecuta en un navegador) se conecta directamente a la base de datos. Esto significa que el controlador JDBC específico para conectarse con la base de datos, debe residir en el sistema local. La base de datos puede estar en cualquier otra máquina y se accede a ella por medio de una conexión a la red. Esta es la configuración típica Cliente/Servidor: el programa cliente envía instrucciones SQL a la base de datos, ésta las procesa y envía los resultados de vuelta a la aplicación.

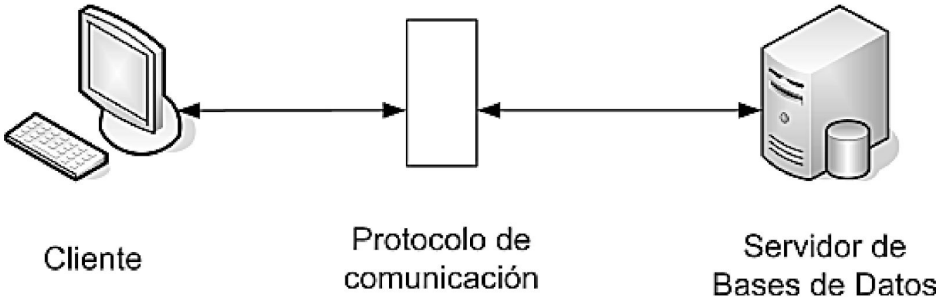

Figura 1 Modelo de Dos Capas

- Modelo de Tres Capas
  En este modelo de acceso a las bases de datos, las instrucciones son enviadas a una capa intermedia entre
  Cliente y el Servidor, la cual se encarga de enviar las sentencias SQL a la base de datos y obtener el resultado desde la base de datos. En este caso el usuario no tiene contacto directo, ni a través de la red, con la máquina donde reside la base de datos.
  Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los controladores JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de controlador.

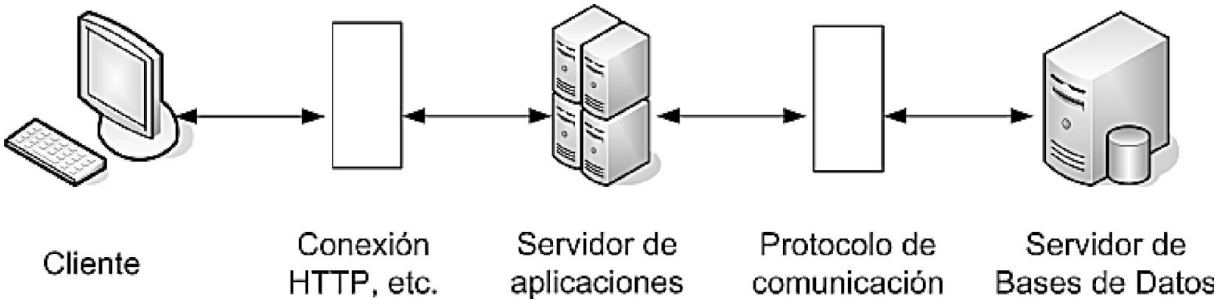

Figura 2 Modelo de Tres Capas

- Tipos de controladores
  Un controlador JDBC es una clase que implementa la interfaz 'JDBC Driver', acepta peticiones SQL por parte del programa y manipula esto para crear peticiones a una base de datos dada. Existen 4 tipos de controladores, su clasificación se basa en la forma en que opera cada uno de ellos.

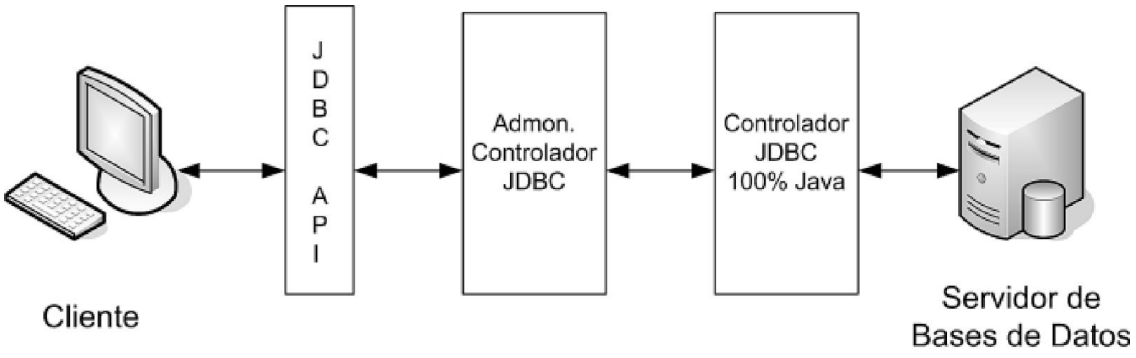

Figura 3 Acceso a una Base de Datos por medio de un controlador JDBC Tipo 4

- Protocolo nativo completamente de java
  Este tipo de controlador convierte las llamadas JDBC en llamadas del protocolo de red utilizado directamente por el SMBD. Esto permite llamadas directas desde la máquina cliente al servidor del SMBD y conjunta una solución práctica para accesos en 'Intranets'. El controlador es independiente de la plataforma y una vez compilado se puede utilizar en cualquier sistema. SQL Server proporciona un controlador de esta clase.

- Proceso general de conexión
  El proceso de conexión a una base de datos desde JDBC se compone, de manera general, de las siguientes etapas:
  - Establecer la conexión a la base de datos
  - Crear y enviar sentencias SQL al SMBD.
  - Recibir la información y procesar los resultados.

    - Estableciendo la conexión a la base de datos
      La primera tarea que debe ejecutarse, es importar el controlador JDBC, esto se logra con la siguiente instrucción en java:

      ```
      import java.sql.*;
      ```

      posteriormente hay que cargar el controlador, esto se lleva a cabo con la instrucción siguiente:

      ```
      Class.forName("oracle.jdbc.OracleDriver");
      ```

      existe otro método, pero ciertamente este es el más sencillo.
      Dentro de JDBC, una base de datos es representada por un URL (Uniform Resource Locator), éste debe tener la siguiente estructura:

jdbc:oracleserver://servidorBaseDeDatos:puerto;nombreBaseDeDatos

Para obtener la conexión a la base de datos, es necesario crear una instancia de la clase 'Connection' de JDBC. y posteriormente asignar la conexión al ejecutar el método 'DriverManager.getConnection()'.
Ejemplo general de uso:

```
Connection db = DriverManager.getConnection(url, usuario, contraseña);
```

hay que recordar, que usuario y contraseña, hacen referencia al usuario de la base de datos y su contraseña que tenga asignada, además son cadenas de texto que pueden provenir de múltiples objetos de Java ya sea mediante la lectura a partir de un archivo, de la entrada estándar por parte del usuario u otras.
En caso de tener éxito la ejecución del método, se obtiene una conexión que puede utilizarse para interactuar con el SMBD enviando las sentencias SQL apropiadas.

- Crear y enviar sentencias SQL al SMBD
  Una vez establecida la conexión a la base de datos por parte de la aplicación, es posible enviar sentencias SQL al SMBD. Para este propósito, se cuenta con tres clases para trabajar con las sentencias SQL.
  - Statement: Trabaja con una sentencia SQL sin parámetros.
  - PreparedStatement: Utilizado con sentencias SQL que se ejecutan con frecuencia, por tanto son pre compiladas y pueden tomar parámetros.
  - CallableStatement: Trabaja con procedimientos almacenados.

  Es preciso aclarar que en la presente práctica solamente se estudia a la clase 'Statement'. Un objeto 'Statement' es el que envía nuestras sentencias SQL al controlador de la base de datos. Para ello, sólo se crea una instancia del objeto y se ejecuta, pasando como parámetro la sentencia SQL que desea enviar. Para una sentencia SELECT, el método a ejecutar es 'executeQuery'. Para sentencias que crean o modifican tablas, el método a utilizar es 'executeUpdate'. Es necesario que exista un objeto Connection que represente una conexión activa para el objeto Statement. En el

siguiente ejemplo, se utiliza un objeto Connection llamado 'db'.

```
Statement sentencia = db.createStatement();
```

en ese momento se crea una instancia del objeto Statement '(sentencia)', pero no tiene ninguna sentencia SQL para enviar al controlador de la base de datos. Para lograr esto, se necesita ejecutar un método 'executeQuery' ó 'executeUpdate', dependiendo de la finalidad. Por ejemplo, en el siguiente fragmento de código, se suministra executeUpdate con la sentencia SQL del ejemplo anterior.

```
sentencia.executeUpdate("CREATE TABLE prueba "+
"(id_prueba INTEGER, nombre VARCHAR(30)"+
"cantidad INTEGER, total INTEGER);");
```

es importante mencionar que el método 'executeUpdate' recibe como parámetro una cadena, por lo tanto la asignación de la sentencia SQL puede llevarse a cabo en otro momento del programa y al ejecutar 'executeUpdate', sólo basta con enviar como parámetro la cadena en cuestión. En este caso interesa únicamente el método 'executeUpdate' puesto que permite enviar sentencias SQL las cuales no regresan resultados por parte de la base de datos, son sentencias DDL.

Por otro lado, el método más utilizado para ejecutar sentencias DML de SQL es 'executeQuery'. Este método se utiliza para ejecutar sentencias SELECT, que comprenden la mayoría de las sentencias SQL.

- Recibir la información y procesar los resultados
  El método 'executeQuery' se utiliza para enviar las sentencias DML de SQL, dado que en una ejecución correcta entrega los resultados de la consulta en un objeto llamado 'ResultSet', por eso es necesario declarar un objeto de la clase y crear una instancia del mismo, la cual contendrá los resultados. Un ejemplo es el siguiente:

```
ResultSet resultados = sentencia.executeQuery (
"SELECT nombre, edad FROM cliente");
```

el objeto 'resultados' contiene las tuplas de la consulta. Para acceder a ellas se utiliza el método 'next()', que devuelve 'true' si hay datos y 'false' en otro caso. Este método utiliza un 'cursor' para desplazarse sobre el conjunto de tuplas del resultado,

inicialmente se posiciona justo antes de la primera tupla de un objeto 'ResultSet'. Esto implica que primero se debe ejecutar el método next para mover el cursor a la primera tupla y convertirla en la tupla actual. Sucesivas invocaciones del método 'next' moverá el cursor a la tupla siguiente.

Entregables

Crea un proyecto en java realizando la conexión a tu base de datos de "Hércules" como se menciona en esta práctica y realiza lo siguiente:

1. Con base en la práctica 10 y 11 elige 2 consultas para mostrar.
2. Se deberá registrar un cliente, 1 socio y sus respectivas clases que toman, así como productos que pudieran comprar.
3. Se deberá eliminar información relacionada con algún socio.
4. Se deberá actualizar información de algún tipo de membresía.

La interfaz no importa, puedes hacer uso de jsp, html o algún framework de tu conveniencia.

Fecha de entrega: 17 de Noviembre del 2017, hora máxima: 23:59:59.