**Academia Java – JAVA**

***Accediendo a la Base de Datos***

Tabla de contenido

[ACCESO AL REPOSITORIO 3](#_Toc314579295)

[1. JDBC (Java Database Connectivity) 4](#_Toc314579296)

[1.1. Arquitectura 4](#_Toc314579297)

[1.2. Beneficios 5](#_Toc314579298)

[1.3. ¿Qué son los drivers y qué hacen? 6](#_Toc314579299)

[1.4. Manejo de conexiones 10](#_Toc314579300)

[1.5. API de JDBC 11](#_Toc314579301)

[1.5.1. Tipos de datos SQL y tipo Java 11](#_Toc314579302)

[1.5.2. Diferentes tipos de Statements 12](#_Toc314579303)

[a. El objeto Statement 12](#_Toc314579304)

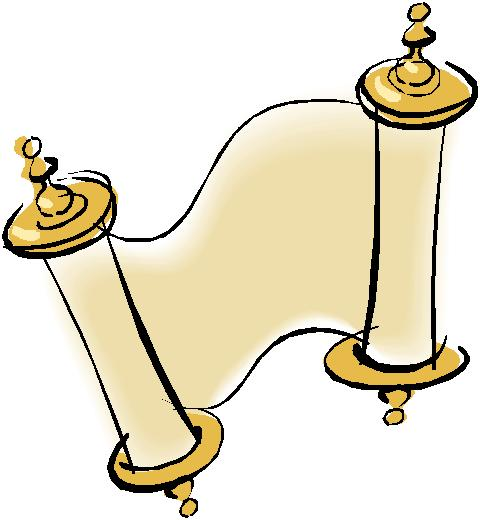
[b. El objeto PreparedStatement 12](#_Toc314579305)

[c. El objeto CallableStatement 13](#_Toc314579306)

[d. El objeto ResultSet 14](#_Toc314579307)

[1.6. Código “spaguetti” 14](#_Toc314579308)

# ACCESO AL REPOSITORIO



Un lugar perfecto para empezar con un requerimiento lo más cercano posible a un aplicación enterprise es la persistencia de los datos.

Si alguno ya ha tenido acercamiento a esta capa en el pasado sabe que existen muchas trampas en ella. Se debe inicializar el framework de acceso a datos, abrir conexiones, manejar excepciones y cerrarlas.

Y también sabemos que si hacemos algo mal corremos el riesgo de corromper o eliminar información invaluable de la empresa.

# JDBC (Java Database Connectivity)

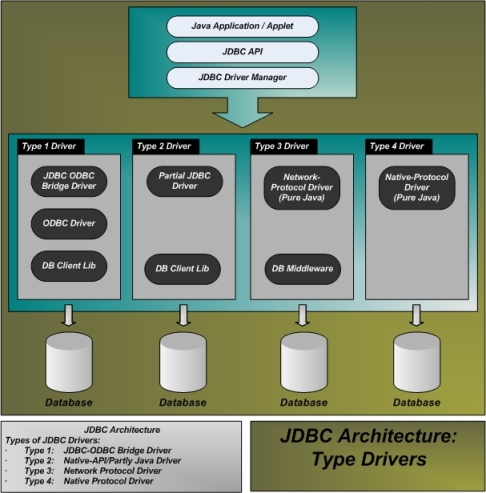
|  |  |
| --- | --- |
| Una de las desventajas principales de la versión 1.0.2 de Java era que no tenía soporte alguno para el acceso a bases de datos.  Esto limitó la utilidad de Java en el campo de los negocios. Sin embargo, a partir de la versión 1.1 del JDK, Java proporciona un soporte completo para bases de datos por medio de JDBC (Java Database Connectivity). |  |

JavaSoft, la subsidiaria de Sun Microsystems (hoy Oracle) referente a Java, introdujo JDBC el cual permite que los programas Java se conecten a cualquier base de datos utilizando diversos controladores (conocidos también como drivers) y un conjunto de objetos y métodos de la API (Interfaz de Programación de Aplicaciones) de Java.

# Arquitectura

|  |  |
| --- | --- |
| Una aplicación desarrollada en Java, cuando necesita conectarse a un repositorio de datos utiliza la API de Java para interactuar con el Driver Manager JDBC. El Driver Manager JDBC utiliza la API de Driver JDBC para cargar el driver apropiado de JDBC. |  |

Los drivers JDBC están puestos a disposición, los proveedores de las diferentes bases de datos del mercado los proveen. Dichos drives se comunican con la capa subyacente del DBMS (Database Manager System, Sistema Manejador de Bases de Datos).



# Beneficios

Antes de la API JDBC, la única forma de poder acceder a las bases de datos desde la Web fue a través de los programas CGI (Common Gateway Interface). JDBC es mucho más rápido que los scripts CGI. El uso de CGI generalmente requiere invocar otro programa (cliente del manejador de la base de datos) que debe ser ejecutado por la computadora en donde se encuentra la aplicación CGI. Este programa intermedio realiza el acceso a la base de datos, procesa datos y los envía en un flujo al programa que realiza la llamada. Esto requiere de varios niveles de procesamiento lo que a su vez incrementa el tiempo de espera así como también propicia que aparezcan más errores.

Invocar un script CGI implica en realidad ejecutar un nuevo script, mientras que la ejecución de una instrucción SQL a través de JDBC en la base de datos requiere sólo de cierto tipo de servidor que pase los comandos SQL a través de la base de datos. Esto abrevia en forma muy notable el tiempo que se necesita para ejecutar las instrucciones de SQL. Mientras que el script CGI debe invocar a la librería cliente para conectarse a la base de datos y procesar los resultados, la solución JDBC permite que el programa Java tenga la conexión a la base de datos para, de esa manera, manejar todo el procesamiento.

Por otro lado, podemos comparar JDBC contra ODBC. ODBC (Open Database Connectivity) es el estándar de Microsoft para la conexión a bases de datos. En principio, ODBC está limitado a que el cliente se encuentre en un sistema Microsoft mientras que el cliente JDBC puede residir en prácticamente cualquier sistema operativo ya que JDBC está escrito en Java y es portable a cualquier plataforma. Además, la configuración del driver o controlador para JDBC es más fácil en comparación con el de ODBC.

La API JDBC tiene clases Java para representar conexiones a bases de datos, sentencias SQL, conjuntos de resultados, etc. Además permite al programador escribir sentencias SQL en Java y procesar los resultados. JDBC es la API primaria para acceso a bases de datos en el lenguaje de programación Java.

# ¿Qué son los drivers y qué hacen?

1. ¿Qué hace el driver?

|  |  |
| --- | --- |
| Todos los programas Java que se conectan a una base de datos vía JDBC, utilizan un driver o controlador de bases de datos. El cual es el intermediario entre la capa de negocio y la capa de base de datos y así mismo funge como el “traductor” de las sentencias Java a sentencias SQL propias del manejador de bases de datos. |  |

1. Tipos de drivers y arquitecturas

Muchos motores de bases de datos tienen diferentes tipos de drivers asociados. Los drivers JDBC caen en una de las siguientes cuatro categorías:

* **Controlador que traduce JDBC a ODBC**. SUN incluye uno en su JDK. Exige la instalación y configuración de ODBC en la computadora cliente.
* **Controlador escrito parcialmente en Java y en código nativo**. Se instala en la computadora cliente software Java y código binario.
* **Controlador de Java puro que utiliza un protocolo de red** (http, por ejemplo) para comunicarse con un servidor de base de datos (por ello se denominan controladores de protocolo no nativo). Este servidor traduce al lenguaje específico del producto. No exige instalación en cliente.
* **Controlador de Java puro con protocolo nativo**. El controlador se comunica con un servidor de base de datos por medio de un protocolo específico a su marca. No exige instalación en cliente.

1. ¿Cómo registrar el driver?

Para poder continuar con la portabilidad de Java y JDBC, los drivers son clases Java que vienen empaquetadas en un archivo ZIP o JAR, eso depende del fabricante del driver.

En JDBC, para cargar o registrar el driver, simplemente se siguen los siguientes pasos:

|  |  |
| --- | --- |
| * Se agrega el jar al proyecto que usamos. En este caso creamos una carpeta especial de nombre “lib” donde agregaremos todas las librerías a usar: |  |
| * Se agrega el jar al build path del proyecto: |  |
| * Ahora tenemos las clases de la librería a disposición del proyecto: |  |
| * Se usa el método estático forName() de la clase Class del paquete java.lang, para este ejemplo se usará una base de datos SQL Server: |  |

En el siguiente punto se observara como se realiza la conexión a la base de datos.

# Manejo de conexiones

Para crear y establecer una conexión a una base de datos desde JDBC, se requiere utilizar el método estático getConnection() de la clase DriverManager del paquete java.sql:



Donde:

url: indica en donde reside la base de datos

user: usuario de acceso a la base de datos

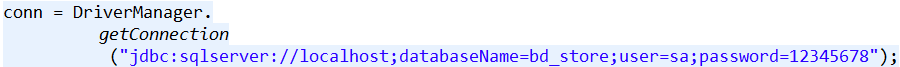
password: contraseña de usuario para acceso a la base de datos

La sintaxis estándar del URL es la siguiente:



Las llaves indican que para algunos drivers, después del número de puerto sigue un diagonal (/) o los dos puntos (:) para después indicar el nombre de la base de datos.

Por ejemplo, si deseáramos conectarnos a una base de datos SQL Server, la cadena de conexión seria:



Las cadenas podrían variar de acuerdo a las bases de datos con las cuales se trabaje.

Posteriormente a la creación de la conexión se debe tomar en cuenta si la conexión va a involucrar transacciones. En el caso de JDBC, al final de cada sentencia SQL se realiza el commit de manera automática y predeterminada.

Por ello, si vamos a manejar transacciones, podemos habilitar o deshabilitar la opción de autocommit. Para ello, sobre el objeto Connection recientemente creado, invocamos al método setAutoCommit() pasando un argumento tipo boolean. Si el boolean es true, entonces el autocommit está activo (esta es la opción por defecto de JDBC) mientras que si es false, entonces se ha desactivado el autocommit y siendo así, se debe realizar el commit de manera manual al final de la transacción.



Posteriormente, se tienen 3 opciones: crear un objeto de tipo Statement, un objeto de tipo PreparedStatement o un objeto de tipo CallableStatement.

Si se está utilizando una transacción, se debe realizar ya sea un commit (para confirmar las acciones hechas en las tablas) o un rollback (para deshacer los cambios). Ambas acciones son sobre la referencia de tipo Connection:



O bien,



Es recomendable que siempre se cierre la conexión una vez finalizada la ejecución de la(s) sentencia(s) SQL, con el método close():



# API de JDBC

# Tipos de datos SQL y tipo Java

Tengamos en cuenta que los tipos de SQL no se corresponden de forma exacta (nombre por nombre) con los de Java:

|  |  |
| --- | --- |
| **SQL** | **Java** |
| INT o INTEGER | int |
| FLOAT | float |
| DOUBLE | double |
| CHAR(x) o VARCHAR(x) | String |
| BOOLEAN | boolean |
| DATE | Date |
| TIME | Time |
| BLOB | java.sql.Blob |
| CLOB | java.sql.Clob |

# Diferentes tipos de Statements

## El objeto Statement

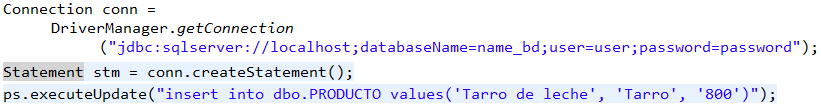
El objeto de tipo Statement se usa para crear sentencias SQL estáticas y regresar el resultado que éstas producen.

Por lo general, el objeto Statement se utiliza para la ejecución de sentencias de consulta SELECT y en ese caso, se invoca al método executeQuery() el cual regresa un objeto ResultSet que contiene los registros resultantes de la consulta:



También se utiliza para realizar sentencias INSERT, UPDATE y DELETE y en ese caso, se utiliza el método executeUpdate() llevando como argumento el String con la sentencia SQL.

Por ejemplo, la forma más sencilla seria armar la sentencia directamente con:



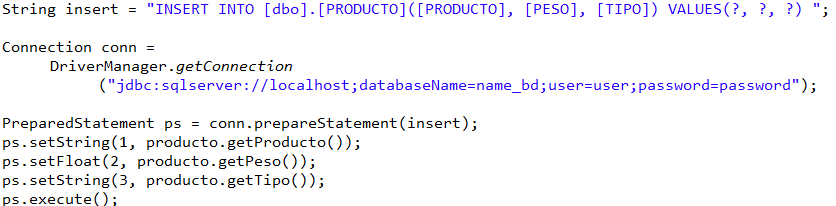
## El objeto PreparedStatement

El objeto de tipo PreparedStatement se utiliza para ejecutar sentencias SQL precompiladas (o dinámicas en cuanto a los valores de los parámetros en la condición de la sentencia SQL).

Por lo general, PreparedStatement se utiliza para la ejecución de sentencias INSERT, DELETE y UPDATE ya que éstas últimas requieren condicionantes por lo que se crea el String con la sentencia SQL desde la obtención del objeto PreparedStatement.

Para asignar el valor de una condicionante, se utilizan los diversos métodos setAlgo() donde “Algo” representa a cada uno de los diversos tipos de datos que existen en Java (int, byte, float, double, String, etc) y posteriormente se invoca al método executeUpdate() el cual regresa un valor int que indica el número de registros afectados por la sentencia (ya sea INSERT, UPDATE o DELETE).

Por ejemplo, utilizando el objeto PreparedStatement:



Se observa que JDBC sustituye los signos de interrogación (?) o comodines por los valores según el orden establecido por el primer argumento del método setAlgo(). Esto también se puede hacer con consultas SELECT.

## El objeto CallableStatement

El objeto de tipo CallableStatement se utiliza para ejecutar procedimientos almacenados (*stored procedures*) SQL.

La API de JDBC brinda una sintaxis para que todos los procedimientos almacenados sean llamados de la misma manera en los diferentes sistemas manejadores de bases de datos.

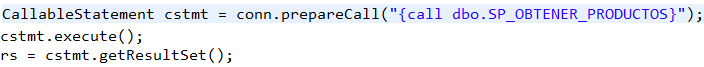
Dicha sintaxis provee una manera que incluye un parámetro de resultado y otra manera que no incluye dicho parámetro. Si el parámetro es utilizado, debe ser registrado como un parámetro OUT. Los parámetros restantes pueden ser utilizados para entrada, salida o ambos. Los parámetros son referenciados a través de un número secuencial, empezando por 1. Las sintaxis son:

{?=call <nombre\_del\_procedimiento\_almacenado>[<arg1>,<arg2>, ...]}

{call <nombre\_del\_procedimiento\_almacenado>[<arg1>,<arg2>, ...]}

Los valores de los parámetros IN (de entrada) están establecidos por los métodos setXXX() heredados de PreparedStatement. Los tipos de todos los parámetros OUT (de salida) deben ser registrados ejecutando el procedimiento almacenado, sus valores son recuperados después de la ejecución a través de los métodos getXXX() que provee CallableStatement.

Un objeto CallableStatement también puede regresar un objeto ResultSet o múltiples objetos ResultSet. En este último caso, los múltiples objetos ResultSet son manejados por métodos heredados de Statement.



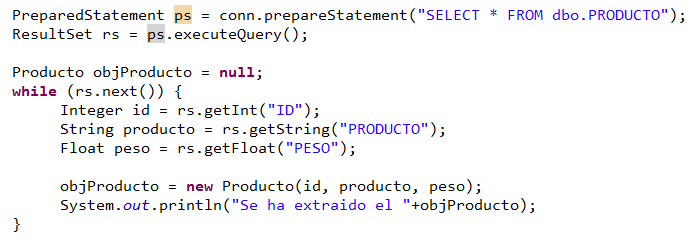
El ResultSet obtenido se trata como cualquier otro objeto de tipo ResultSet en el código.

## El objeto ResultSet

Este objeto representa los resultados de la ejecución de un query a la base de datos. Este objeto está muy relacionado a la sentencia SELECT ya que es la única que permite registros como respuesta, además de los procedimientos almacenados.

Este objeto tiene un cursor el cual, de entrada está antes del primer registro resultante (si hay registros resultantes) y conforme se navega por los registros, el ResultSet mantiene el cursor en el registro actual. De manera predeterminada un ResultSet no es actualizable y su cursor se mueve solamente hacia delante (TYPE\_FORWARD\_ONLY).

La interfaz ResultSet provee los metodos getXXX() donde XXX representan los diversos tipos de datos con los que cuenta Java (boolean, long, int, etc.) para regresar los datos de las columnas del resultado de la sentencia SQL. El argumento que los métodos getXXX() requieren puede ser el índice que la columna ocupa en la sentencia SELECT o el nombre de la columna (esta última opción es de caso no-sensitivo).



# Código “spaguetti”

El siguiente código es sintácticamente correcto, pero es un ejemplo de programación spaguetti.

Este tipo de cosas son un ejemplo de hábitos de programación de personas que no tienen en cuenta patrones de programación:



No siempre los programas más cortos son los mejores. Tampoco quiero decirles que deban explayarse enormemente, pero es bueno siempre revisar los buenos patrones de programación y también los antipatrones para no seguirlos.