## Driver manager

DriverManager es una clase estática de Java Plaform, Standard Edition (J2SE) y Java SE Development Kit (JDK). DriverManager gestiona el conjunto de controladores Java Database Connectivity (JDBC) que están disponibles para que los utilice una aplicación.

Las aplicaciones pueden utilizar varios controladores JDBC simultáneamente si es necesario. Cada aplicación específica un controlador JDBC mediante la utilización de un URL (Localizador universal de recursos). Pasando un URL de un controlador JDBC específico a DriverManager, la aplicación informa a DriverManager acerca del tipo de conexión JDBC que debe devolverse a la aplicación.

Para poder realizar esta operación, DriverManager debe estar al corriente de los controladores JDBC disponibles para que pueda distribuir las conexiones. Efectuando una llamada al método Class.forName, carga una clase en la máquina virtual Java (JVM) que se está ejecutando en función del nombre de serie que se pasa en el método. A continuación figura un ejemplo del método class.forName utilizado para cargar el controlador JDBC nativo:

Connection c = DriverManager.getConnection(jdbc:db2:*local);

## Connection

El objeto Connection representa una conexión con un origen de datos en Java Database Connectivity (JDBC). Los objetos Statement se crean a través de objetos Connection para procesar sentencias SQL en la base de datos. Un programa de aplicación puede tener varias conexiones simultáneas. Estos objetos Connection puede conectarse todos a la misma base de datos o a bases de datos diferentes.

La obtención de una conexión en JDBC puede realizarse de dos maneras:

- Mediante la clase DriverManager.
- Utilizando DataSources.

Es preferible utilizar DataSources para obtener una conexión, ya que mejora la portabilidad y la capacidad de mantenimiento de las aplicaciones. También permite que una aplicación utilice de forma transparente las agrupaciones de conexiones y sentencias y las transacciones distribuidas.

## Statements

PreparedStatement es heredera de Statement, y CallableStatement es heredera de PreparedStatement. Utilice los siguientes objetos Statement para ejecutar las distintas sentencias SQL:

Interfaz Statement - Ejecuta una sentencia SQL simple que no tiene ningún parámetro.

Interfaz PreparedStatement - Ejecuta una sentencia SQL precompilada que puede tener o no tener parámetros IN.

Interfaz CallableStatement - Ejecuta una llamada a un procedimiento almacenado de base de datos. El objeto CallableStatement puede tener o no tener parámetros IN, OUT e INOUT.

El objeto Statement permite someter a una base de datos varios mandatos SQL como si fuesen un solo grupo mediante el uso del soporte de proceso por lotes. Puede mejorar el rendimiento empleando el soporte de proceso por lotes ya que normalmente se tarda menos en procesar un grupo de operaciones que en procesarlas una a una.

Cuando utilice las actualizaciones por lotes, normalmente deberá desactivar el compromiso automático. La desactivación del compromiso automático permite al programa determinar si debe comprometer la transacción en el caso de que se produzca un error y no se hayan ejecutado todos los mandatos. En JDBC 2.0 y las especificaciones de JDBC posteriores, un objeto Statement puede hacer un seguimiento de una lista de mandatos que pueden someterse satisfactoriamente y ejecutarse conjuntamente en un grupo. Cuando el método executeBatch() ejecuta esta lista de mandatos por lotes, la ejecución de los mandatos se realiza en el orden en que se añadieron a la lista.

## Inserting and Updating Rows

### Insert

- La sentencia SQL utilizada debe ser una sentencia INSERT con una cláusula VALUES, indicando que no es una sentencia INSERT con SUBSELECT. El controlador JDBC reconoce esta restricción y realiza las acciones adecuadas.
- Debe utilizarse una PreparedStatement, indicando que no existe soporte optimizado para objetos Statement. El controlador JDBC reconoce esta restricción y realiza las acciones adecuadas.
- La sentencia SQL debe especificar marcadores de parámetro para todas las columnas de la tabla. Esto significa que no puede utilizar valores de constante para una columna ni permitir que la base de datos inserte valores predeterminados para ninguna de las columnas. El controlador JDBC no tiene ningún mecanismo para manejar las pruebas de marcadores de parámetro específicos en la sentencia SQL. Si establece la propiedad para realizar inserciones en bloque optimizadas y no evita los valores predeterminados o constantes en las sentencias SQL, los valores que terminen en la tabla de base de datos no serán correctos.

```
public void save(Usuario usuario) {
    String sql;
    if (usuario.getId() != null && usuario.getId() >0){
        sql = "UPDATE usuario set nombre =?, apellido=?, email=?,
password=?, id_categoria=? where id=?";
    }else {
        sql = "insert into
usuario(nombre,apellido,email,password,id_categoria) values(?,?,?,?,?)";
    }
    try (Connection c = getConnection();
    PreparedStatement stmt = c.prepareStatement(sql)){
        stmt.setString(1, usuario.getNombre());
        stmt.setString(2, usuario.getApellido());
        stmt.setString(3, usuario.getEmail());
        stmt.setString(4, usuario.getPassword());
        stmt.setLong(5,usuario.getCategoria().getId());

        if (usuario.getId() != null && usuario.getId()>0){
```

```
            stmt.setLong(6,usuario.getId());
        }
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }


}
```

## Deleting Rows

Nos sirve para eliminar elementos a través de una sentencia sql esta sentencia debe tener una conexión a la base de datos.

```
@Override
public void delete(Long id) {
    try(Connection c = getConnection();
    PreparedStatement stmt = c.prepareStatement("delete from usuario
where id = ?")) {
    stmt.setLong(1,id);
    stmt.executeUpdate();

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }

}
```

## Result Set

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. Algunos de los métodos get<Tipo> tienen parámetros adicionales, como el objeto opcional Calendar, que se puede pasar a

los métodos getTime, getDate y getTimestamp. Consulte el Javadoc del paquete java.sql para obtener todos los detalles.

En los métodos get que devuelven objetos, el valor de retorno es null cuando la columna del ResultSet es nula. En tipos primitivos, no puede devolverse null. En estos casos, el valor es 0 o false. Si una aplicación debe distinguir entre null, y 0 o false, puede utilizarse el método wasNull inmediatamente después de la llamada. A continuación, este método puede determinar si el valor era un valor 0 o false real o si ese valor se ha devuelto debido a que el valor de ResultSet era de hecho null.

## Mapping Between SQL & Java Data Types

Para escribir programas JDBC y SQLJ eficientes, necesita utilizar las mejores asignaciones entre tipos de datos Java y tipos de datos de columnas de tablas.

*Tabla 1. Asignaciones de tipos de datos Java a tipos de datos del servidor de bases de datos para actualizar tablas de bases de datos*

| tipo de datos java | Tipo de datos de base de datos |
|---|---|
| corto, java.lang.Short | PEQUEÑO |
| booleano 1 , byte 1 , java.lang.Boolean, java.lang.Byte | PEQUEÑO |
| int, java.lang.Integer | ENTERO |
| largo, java.lang.largo | GRANDE 12 |
| java.math.BigInteger | GRANDE 11 |
| java.math.BigInteger | CHAR( $n$ ) 11 , 5 |
| flotador, java.lang.Float | REAL |
| doble, java.lang.Double | DOBLE |
| java.math.BigDecimal | DECIMAL( $p$ , $s$ ) 2 |
| java.math.BigDecimal | DECFLOAT( $n$ ) 3 , 4 |
| java.lang.Cadena | CARACTERÍSTICA ( $n$ ) 5 |
| java.lang.Cadena | GRÁFICO( $m$ ) 6 |
| java.lang.Cadena | VARCHAR( $n$ ) 7 |
| java.lang.Cadena | VARGRAFICO( $m$ ) 8 |
| java.lang.Cadena | CLOB 9 |
| java.lang.Cadena | XML 10 |
| byte[] | CHAR( $n$ ) PARA DATOS DE BIT 5 |
| byte[] | VARCHAR( $n$ ) PARA DATOS DE BIT 7 |
| byte[] | BINARIO( $n$ ) 5 , 13 |
| byte[] | VARBINARIO( $n$ ) 7 , 13 |
| byte[] | BLOB 9 |

## PreparedStatement

Para crear objetos PreparedStatement nuevos, se utiliza el método prepareStatement. A diferencia de en el método createStatement, la sentencia SQL debe suministrarse al crear el objeto PreparedStatement. En ese momento, la sentencia SQL se precompila para su utilización.

```
@Override
public void save(Usuario usuario) {
    String sql;
    if (usuario.getId() != null && usuario.getId() >0){
        sql = "UPDATE usuario set nombre =?, apellido=?, email=?,
password=?, id_categoria=? where id=?";
    }else {
        sql = "insert into
usuario(nombre,apellido,email,password,id_categoria) values(?,?,?,?,?)";
    }
    try (Connection c = getConnection();
    PreparedStatement stmt = c.prepareStatement(sql)){
        stmt.setString(1, usuario.getNombre());
        stmt.setString(2, usuario.getApellido());
        stmt.setString(3, usuario.getEmail());
        stmt.setString(4, usuario.getPassword());
        stmt.setLong(5,usuario.getCategoria().getId());

        if (usuario.getId() != null && usuario.getId()>0){
            stmt.setLong(6,usuario.getId());
        }
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }


}
```

## SQLException

La clase SQLException y sus subtipos proporcionan información acerca de los errores y avisos que se producen mientras se está accediendo a un origen de datos.

A diferencia de la mayor parte de JDBC, que se define mediante interfaces, el soporte de excepciones se suministra en clases. La clase básica para las excepciones que se producen durante la ejecución de aplicaciones JDBC es

SQLException. Todos los métodos de la API JDBC se declaran capaces de lanzar SQLExceptions. SQLException es una ampliación de java.lang.Exception y proporciona información adicional relacionada con las anomalías que se producen en un contexto de base de datos. Específicamente, en una SQLException está disponible la siguiente información:

- Texto descriptivo
- SQLState
- Código de error
- Una referencia a las demás excepciones que también se han producido