Acceso a Bases de Datos

Sistemas Gestores Bases de Datos

- Access
- SQL Server
- Oracle
- DB2
- MySQL
- MariaDB
- PostgreSQL

Documentales: MongoDB y CouchDB

Grafo: Neo4J y HypergraphDB

Clave-valor: Riak y Redis

Columna: Apache HBase y Cassandra

Orientadas a objetos

Más info en https://hostingdata.co.uk/nosql-database/

Repaso sintaxis SQL

Comandos

Cláusulas

Operadores

Funciones

Tipos de comandos

- Los DDL (Data Definition Languaje), que permiten crear, modificar y borrar nuevas bases de datos, tablas, campos y vistas.

- Los DML (Data Manipulation Languaje), que permiten introducir información en la BD, borrarla y modificarla.

- Los DQL (Data Query Languaje), que permiten generar consultas para ordenar, filtrar y extraer información de la base de datos

Comandos DDL

- CREATE: Crear nuevas tablas, campos e índices.
- ALTER: Modificación de tablas añadiendo campos o modificando la definición de los campos.
- DROP: Instrucción para eliminar tablas, campos e índices.

Comandos DML

- INSERT: Insertar registros a la base de datos.
- UPDATE: Instrucción que modifica los valores de los campos y registros especificados en los criterios.
- DELETE: Eliminar registros de una tabla de la base de datos.

Comandos DQL

- SELECT: Consulta de registros de la base de datos que cumplen un criterio determinado.

Repaso sintaxis SQL

Comandos

Cláusulas

Operadores

Funciones

Claúsulas

Las cláusulas son condiciones de modificación, utilizadas para definir los datos que se desean seleccionar o manipular:

- FROM: Utilizada para especificar la tabla de la que se seleccionarán los registros.
- WHERE: Cláusula para detallar las condiciones que deben reunir los registros resultantes.
- GROUP BY: Utilizado para separar registros seleccionados en grupos específicos.
- HAVING: Utilizada para expresar la condición que ha de cumplir cada grupo.
- ORDER BY: Utilizada para ordenar los registros seleccionados de acuerdo a un criterio dado

Repaso sintaxis SQL

Comandos

Cláusulas

Operadores

Funciones

Operadores lógicos y de comparación

- AND: Evalúa dos condiciones y devuelve el valor cierto, si ambas condiciones son ciertas.
- OR: Evalúa dos condiciones y devuelve el valor cierto, si alguna de las dos condiciones es cierta.
- NOT: Negación lógica. Devuelve el valor contrario a la expresión.

- < [...] menor que [...]
- > [...] mayor que [...]
- <> [...] diferente a [...]
- <= [...] menor o igual que [...]
- >= [...] mayor o igual que [...]
- = [...] igual que [...]
- BETWEEN: Especifica un intervalo de valores
- LIKE: Compara un modelo
- IN: Operadores para especificar registros de una tabla

JDBC

Java Data Base Connection

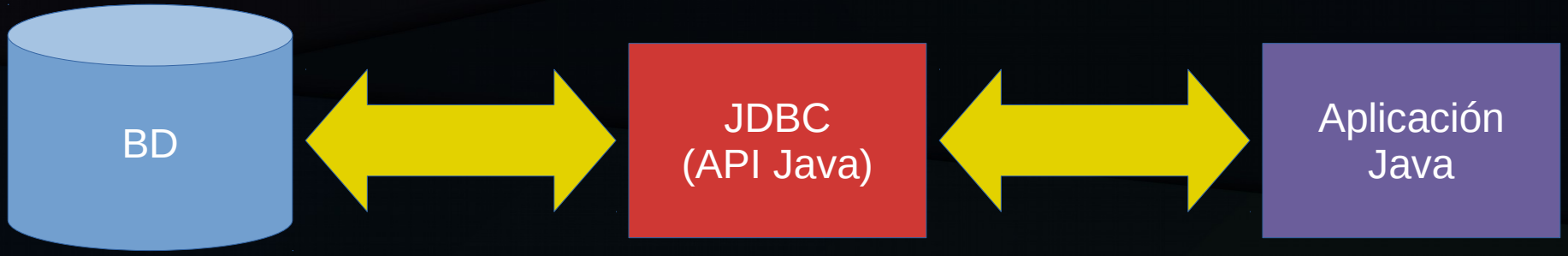
JDBC es una API estándar de Java para conectarse a SGBD **relacionales**.

Es multiplataforma y consiste en un conjunto de clases e interfaces escritas en Java.

Una aplicación escrita utilizando JDBC podrá manejar bases de datos Oracle, SQL Server, ...

JDBC como intermediario

JDBC permite leer/escribir desde una BD relacional, independientemente del SGBD de la misma.



Otras APIs para conectarse a BBDD

- DAO (Data Access Objects)
- RDO (Remote Data Objects)
- ADO (ActiveX Data Objects)

El problema que ofrecen estas soluciones es que sólo son para plataformas Windows.

Tipos de Drivers para JDBC

Son los que nos permiten conectarnos a un tipo de BD en concreto.

(1) JDBC-ODBC bridge

(2) Native-API partly-Java

(3) JDBC-Net pure Java driver (tanto en C/S)

(4) JDBC de Java cliente

Funciones JDBC

- Establecer una conexión con una base de datos.
- Enviar sentencias SQL.
- Manipular datos.
- Procesar los resultados de la ejecución de las sentencias.

Instalación en Netbeans

Seguiremos las capturas del documento de teoría (apartado 4)

- XAMPP
- Instalación de JDBC Driver
- Conexión a la BD desde Netbeans

Uso de JDBC en nuestro proyecto

- Añadir la librería JDBC al proyecto
- Cargar el driver:

```
Try {
        Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
} catch (Exception e) {
        // manejamos el error
}
```

Observación sobre clases y métodos

Funcionan con todos los drivers disponibles para Java (JDBC es solo uno, hay muchos más).

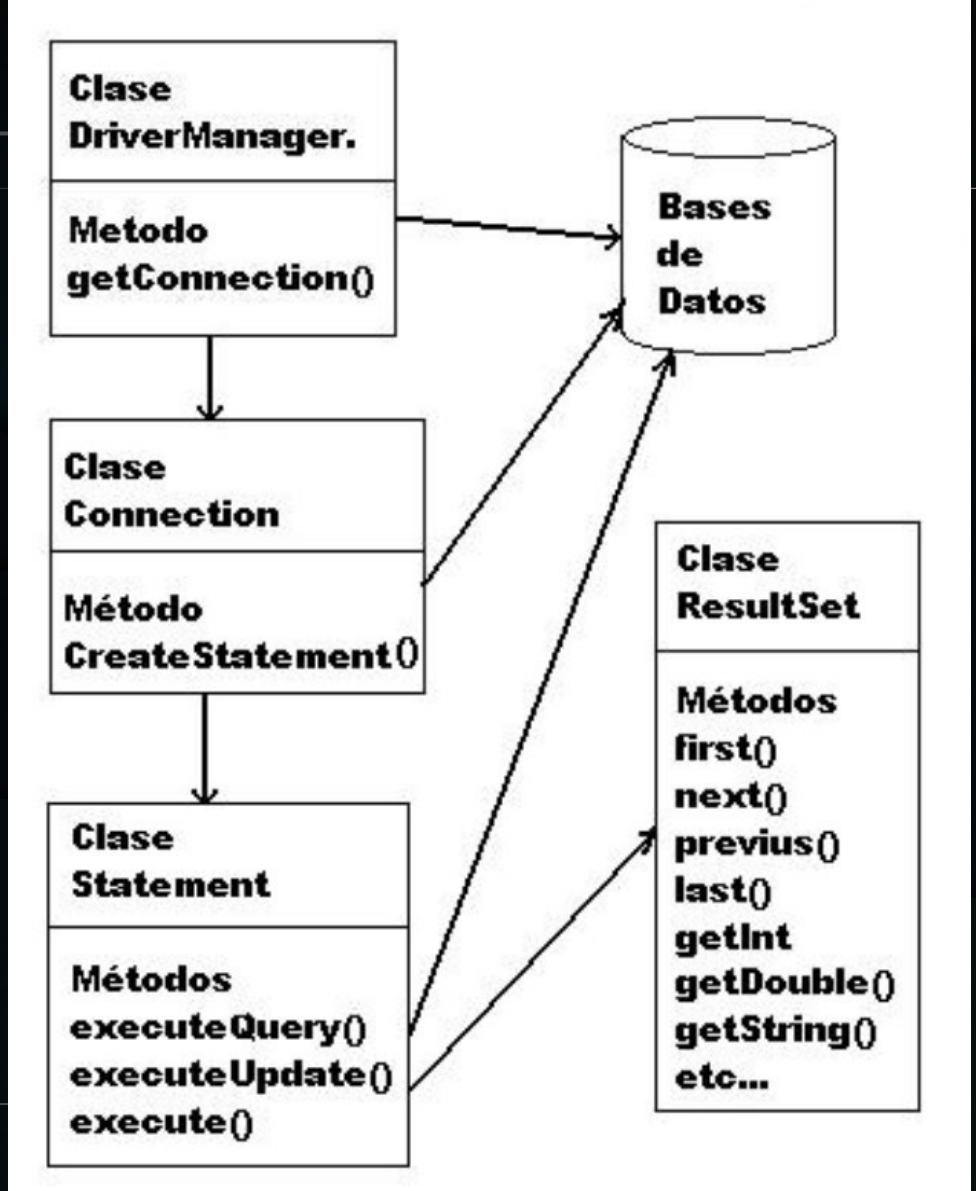
Es posible porque el estándar de Java sólo los define como interfaces y cada librería driver los implementa (define las clases y su código).

Por ello es necesario utilizar Class.forName(...) para indicarle a Java qué driver vamos a utilizar.

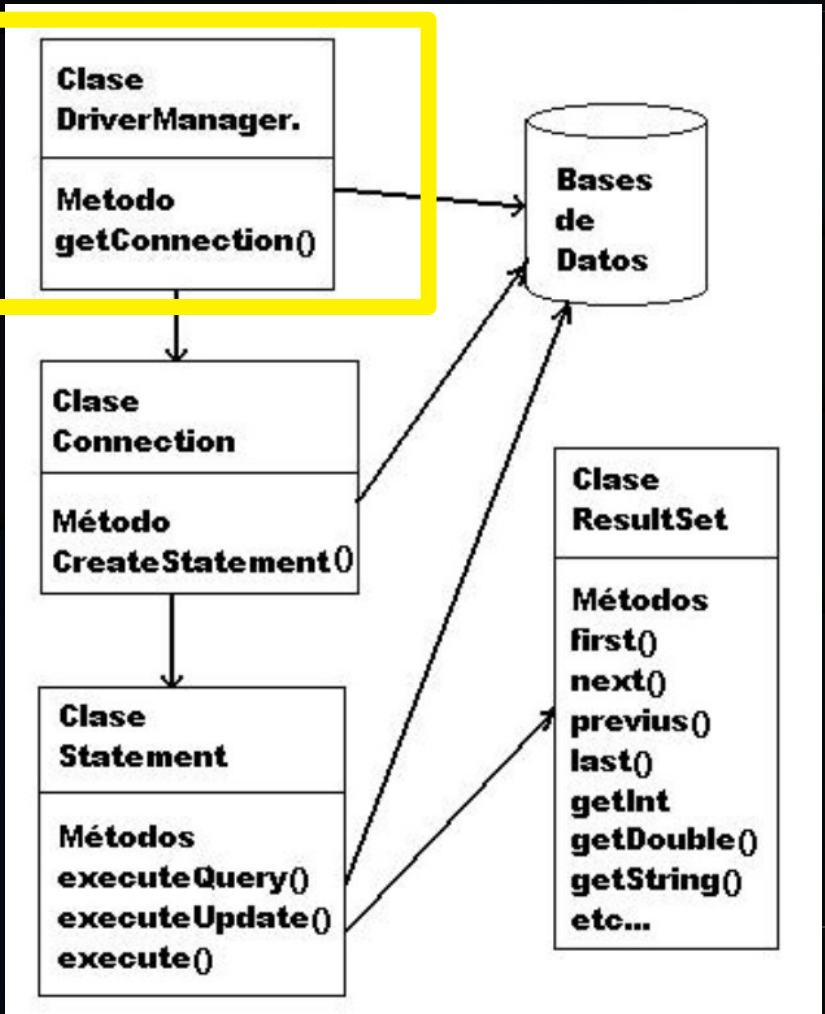
Excepciones

- **SQLException:** La conexión no ha podido producirse. Puede ser por multitud de motivos como una URL mal formada, un error en la red, host o puerto incorrecto, base de datos no existente, usuario y contraseña no validos, etc.
- **SQLTimeOutException:** Se ha superado el LoginTimeout sin recibir respuesta del servidor.

Clases que utilizaremos



java.sql.DriverManager

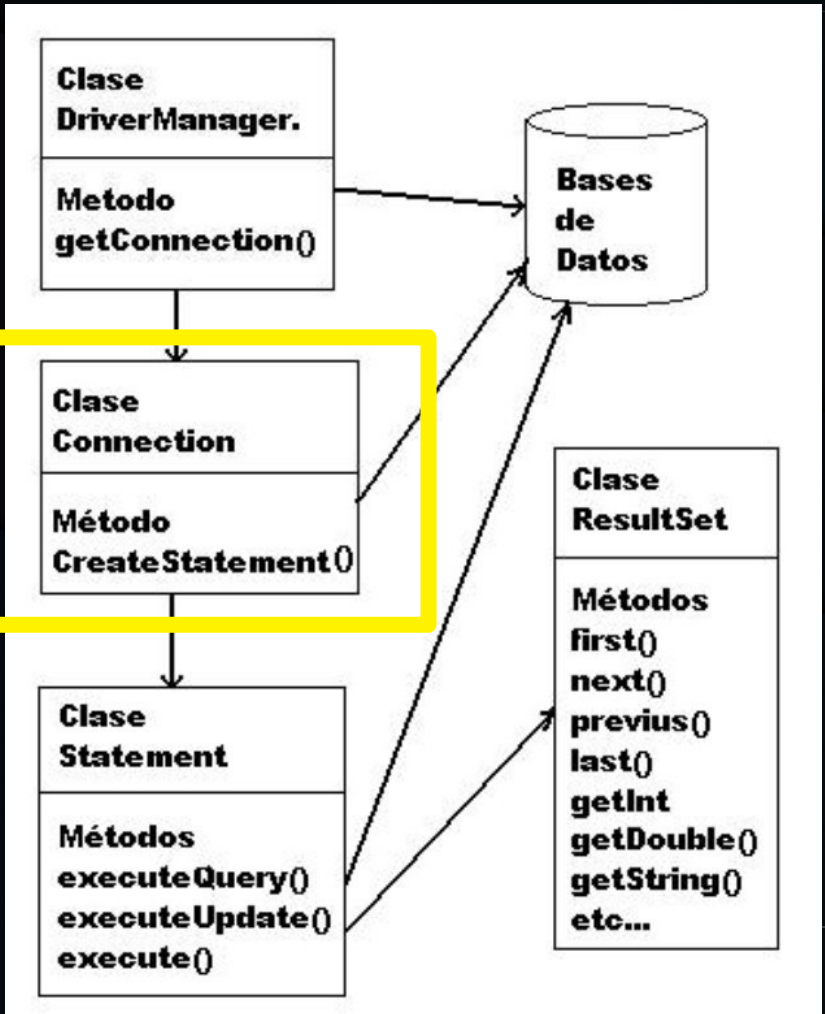


Es la capa gestora del driver JDBC. Se encarga de manejar el driver apropiado y permite crear conexiones con una base de datos mediante el método estático getConnection(...) que tiene dos variantes:

- DriveManager.getConnection(String url)
- DriveManager.getConnection(String url, String user, String password)

Puede lanzar SQLException y SQLTimeoutException

java.sql.Connection



Un objeto Connection representa una sesión de conexión con una base de datos. Una aplicación puede tener tantas conexiones como necesite, ya sea con una o varias bases de datos.

El método más relevante es createStatement() que devuelve un objeto Statement asociado a dicha conexión que permite ejecutar sentencias SQL.

El método createStatement() puede lanzar excepciones de tipo SQLException.

- Statement stmt = conn.createStatement();
- conn.close();

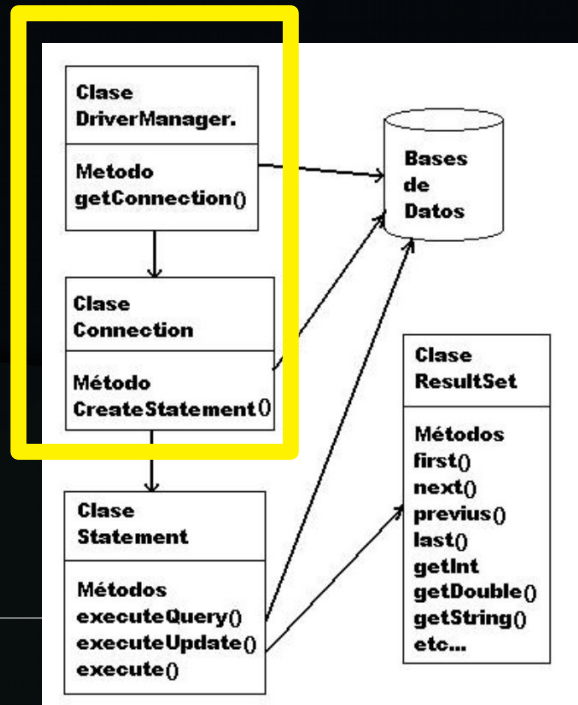
Puede lanzar excepciones de tipo SQLException

Creando una conexión y un stament

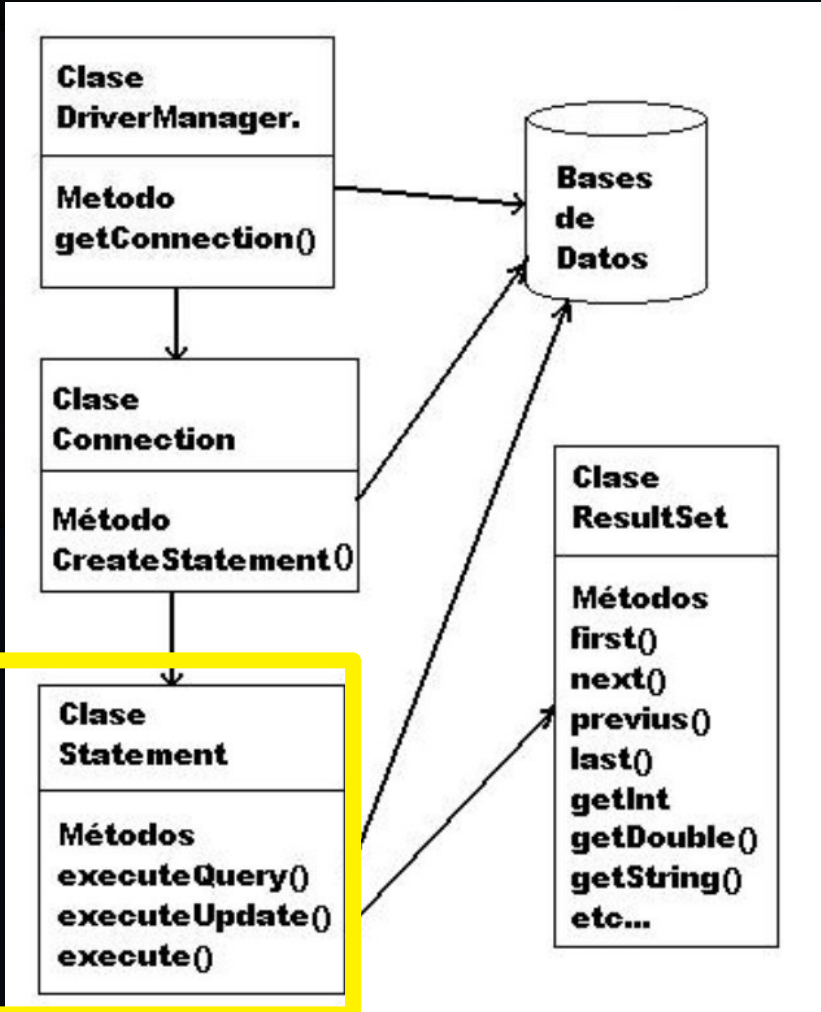
```
String url = "jdbc:mysql://localhost:3306/prueba?serverTimezone=UTC";


Connection conn = DriverManager.getConnection(url,"root","");


Statement st =conn.createStatement();
```



java.sql.Statement



Ejecuta sentencias en la BD.

ResultSet executeQuery(String sql): Ejecuta la sentencia sql indicada (de tipo SELECT).

Devuelve un objeto ResultSet con los datos proporcionados por el servidor.

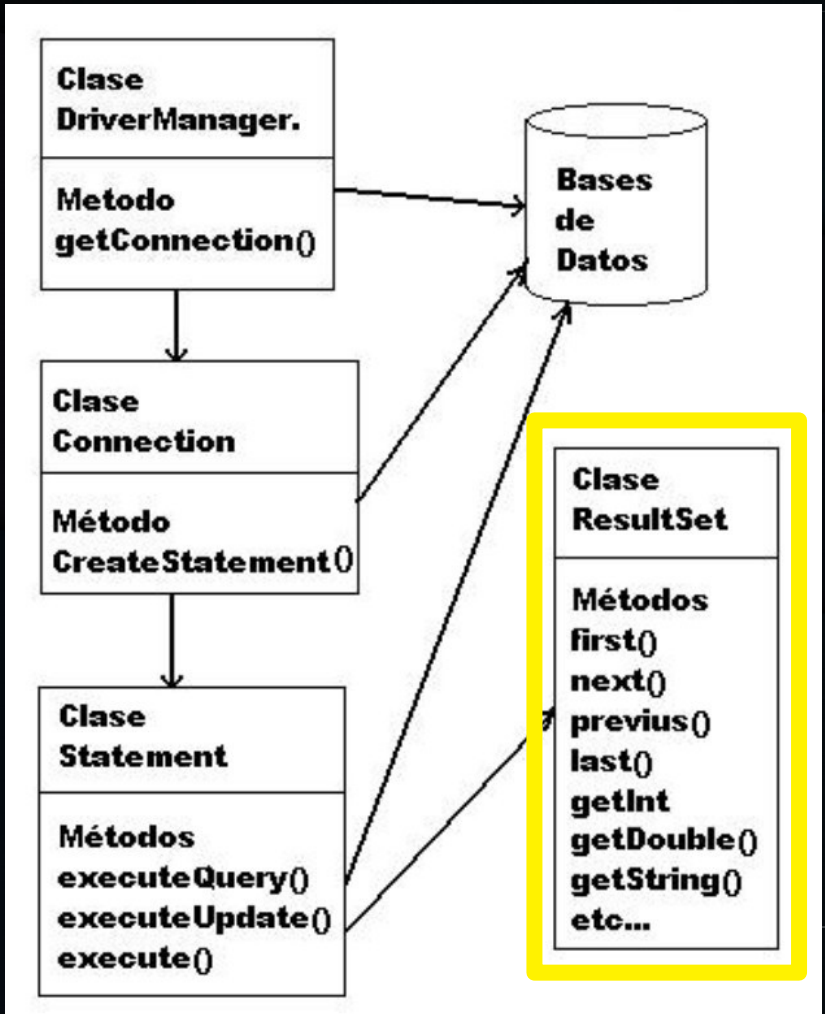
```
ResultSet rs = stmt.executeQuery("SELECT * FROM
vendedores");
```

int executeUpdate(String sql): Ejecuta la sentencia sql indicada (de tipo DML como por ejemplo INSERT, UPDATE o DELETE). Devuelve un el numero de registros que han sido insertados, modificados o eliminados.

```
int nr = stmt.executeUpdate("INSERT INTO vendedores
VALUES (1,'Pedro Gil', '2017-04-11', 15000);")

stmt.close();
```

java.sql.DriverManager



Contiene un conjunto de resultados (datos) obtenidos tras ejecutar una sentencia SQL, normalmente de tipo SELECT. Es una **estructura de datos en forma de tabla** con **registros (filas)** que podemos recorrer para acceder a la informacion de sus **campos (columnas)**.

ResultSet rs = stmt.executeQuery("SELECT * FROM vendedores");

ResultSet utiliza internamente un cursor que apunta al 'registro actual' sobre el que podemos operar. Inicialmente dicho cursor esta situado antes de la primera fila y disponemos de varios métodos para desplazar el cursor. El más común es next():

- **boolean next()**: Mueve el cursor al siguiente registro. Devuelve true si fue posible y false en caso contrario (si ya llegamos al final de la tabla).

Algunos de los métodos para obtener los datos del registro actual son:

- **String getString(String columnLabel)**: Devuelve un dato String de la columna indicada por su nombre. Por ejemplo: rs.getString("nombre")

- **String getString(int columnIndex)**: Devuelve un dato String de la columna indicada por su nombre. La primera columna es la 1, no la cero. Por ejemplo: rs.getString(2)

Ver ejemplo de conexión completa
y ejecución de consultas
en página 22 de la teoría.

Uso de navegabilidad y concurrencia

Podemos navegar entre los resultados y que se actualicen o no desde la BD, si llamamos al método de la clase Connection, con dos argumentos:

```
Statement createStatement(
    int resultSetType,
    int resultSetConcurrency)
```

Navegabilidad entre los resultados

Podemos conseguir:

- Movernos solo hacia el siguiente registro.
- Atrás y adelante, sin que se actualicen los registros desde la BD.
- Atrás y adelante, actualizándose los registros desde la BD

Navegabilidad entre los resultados

Hay tres constantes diferentes:

ResultSet.TYPE_FORWARD_ONLY

ResultSet.TYPE_SCROLL_INSENSITIVE

ResultSet.TYPE_SCROLL_SENSITIVE

Concurrencia en los resultados

Podemos abrir los resultados:

- Sólo de lectura
- Como lectura y escritura

Concurrencia en los resultados

Hay tres constantes diferentes:

ResultSet.CONCUR_READ_ONLY

ResultSet.CONCUR_UPDATABLE

Ejemplo navegabilidad y concurrencia

```
// Cargamos la clase que implementa el Driver
Class.forName("com.mysql.cj.jdbc.Driver").newInstance();
// Creamos una nueva conexión a la base de datos 'prueba'
String url = "jdbc:mysql://localhost:3306/prueba?serverTimezone=UTC";
Connection conn = DriverManager.getConnection(url,"root","");

// Obtenemos un Statement de la conexión
Statement st = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

// Ejecutamos una consulta SELECT para obtener la tabla vendedores
String sql = "SELECT * FROM vendedores";
```

- `ResultSet rs = st.executeQuery(sql);`

Bibliografía

Bases de datos NoSQL

ODBC Wikipedia

ODBC en Microssoft

Clase ResultSet