

JDBC

Antonio Espín Herranz

Acceso a Bases de datos: JDBC

- Tecnología java para acceder a la bases de datos.
- Para acceder a una base de datos usaremos el driver el puente jdbc:odbc de java.
- Esto implica crear una DSN de Sistema, con las herramientas de Windows.
- Podemos conectar con Access, SQL Server, Oracle, etc.

Tipos de Driver

- **Tipo 1: Puente JDBC-ODBC.**
 - Sencillo, se distribuye con la JDK.
 - Bajo rendimiento por la traducciones a ODBC.
- Tipo 2: Driver API nativo / parte Java.
 - Las llamadas son directas a JDBC (sin traducir).
 - Se necesita iniciar en el cliente (no por Internet).
- Tipo 3: Driver protocolo de red / todo java.
 - Lleva un componente intermedio que evita la biblioteca del cliente.
 - Funciona bien en Internet o Intranet.
 - Los datos se pueden obtener mas lento por el servidor de datos.
- **Tipo 4: Driver protocolo nativo / todo java.**
 - No traduce a ODBC, mejor rendimiento que el 1 y 2.
 - No se necesita un SW especial en el cliente o servidor.
 - Se necesita un driver para cada tipo de BD.

Clases en java

- Paquete: java.sql.*;
- Conectar con la BD:
 - 1) Cargar los drivers disponibles:
 - Lo hacemos a través de la clase **Class** y el método estático `forName`. Usamos el puente `jdbc:odbc`.

```
try {  
    Class.forName(sun.jdbc.odbc.JdbcOdbcDriver");  
} catch (ClassNotFoundException e){  
    // .... Mostrar el error.  
}
```

- 2) Capturar la conexión con la base de datos: Clase **DriverManager**
Usaremos el método: `static Connection getConnection(String, String, String);`
Connection es un interface y no se puede instanciar.
Como parámetro indicaremos: `jdbc:odbc:xxx` → donde xxx será nuestra DSN
Los otros dos parámetros serán login y pwd, en caso de que la BD tenga.

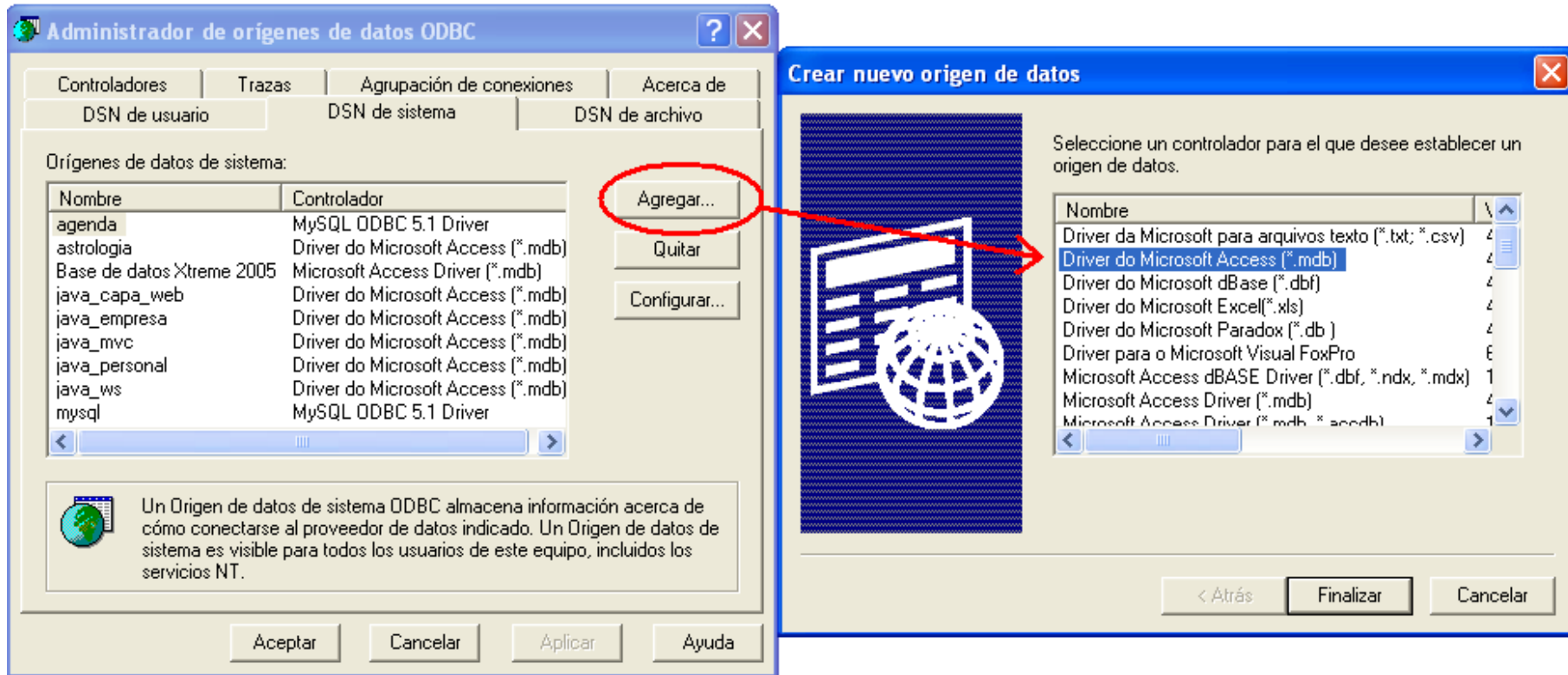
Conexión con el puente JDBC-ODBC

- Para utilizar este método no necesitamos ningún jar.
- La clase que utilizaremos se encuentra dentro de la JDK de java.
- Antes de nada tenemos que crear una DSN desde Windows (en el controlador ODBC) para poder conectar.
- Este método es el más genérico, nos sirve para múltiples BD, pero es el menos eficiente.

Creación de una DSN del Sistema

- Pasos:
 - Crear la base de datos en SQL Server, Access, etc.
 - Crear las tablas y rellenar los datos.
 - Inicio → Panel de control → Herramientas administrativas → **Orígenes de datos ODBC.**

DSN del Sistema ejemplo para Access

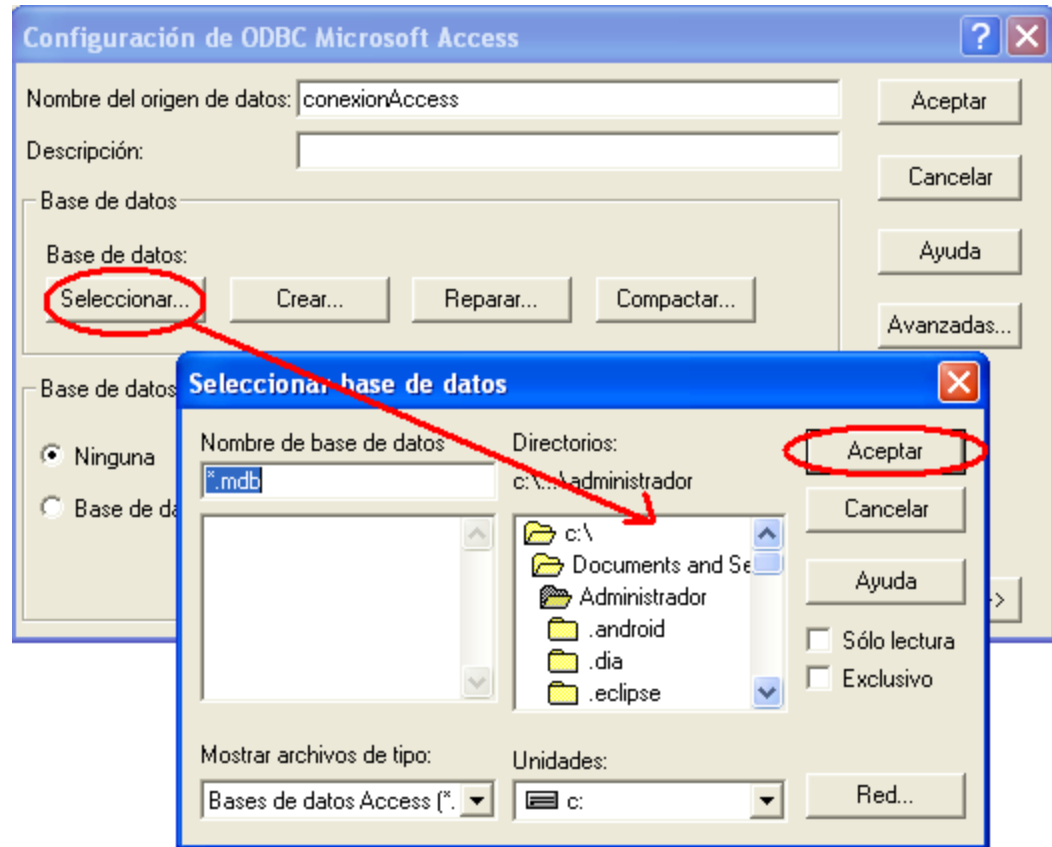


DSN del Sistema ejemplo para Access

- Dar un nombre al origen de datos:
conexionAccess.

- Esta cadena será la que utilizaremos dentro del programa java.

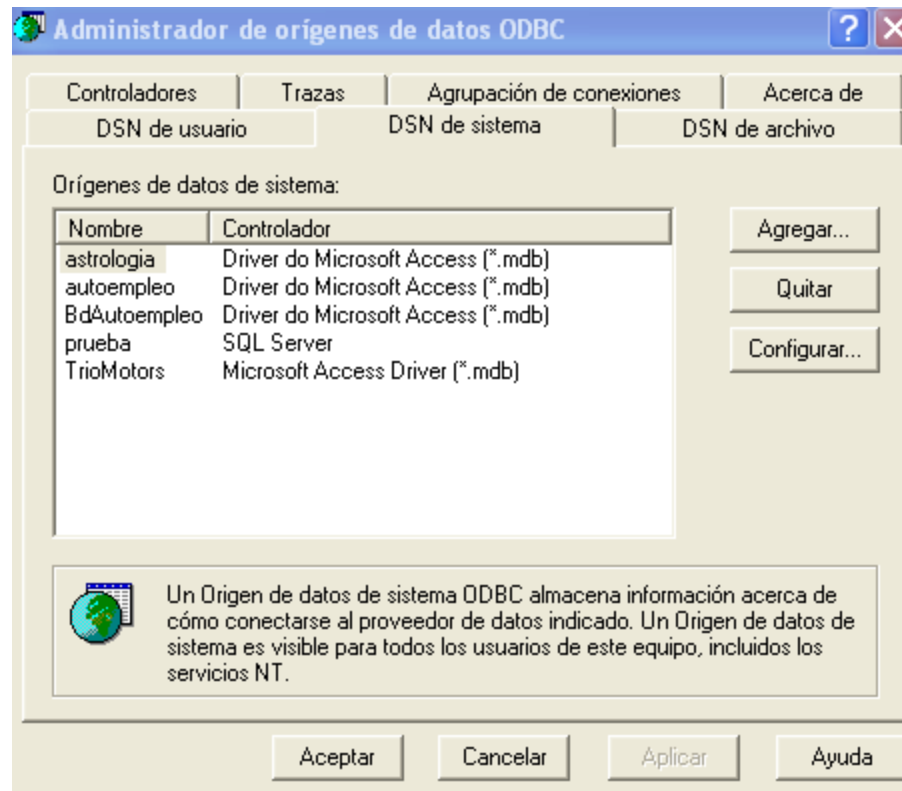
- Seleccionar el fichero de Access.



DSN del Sistema ejemplo para SQLServer

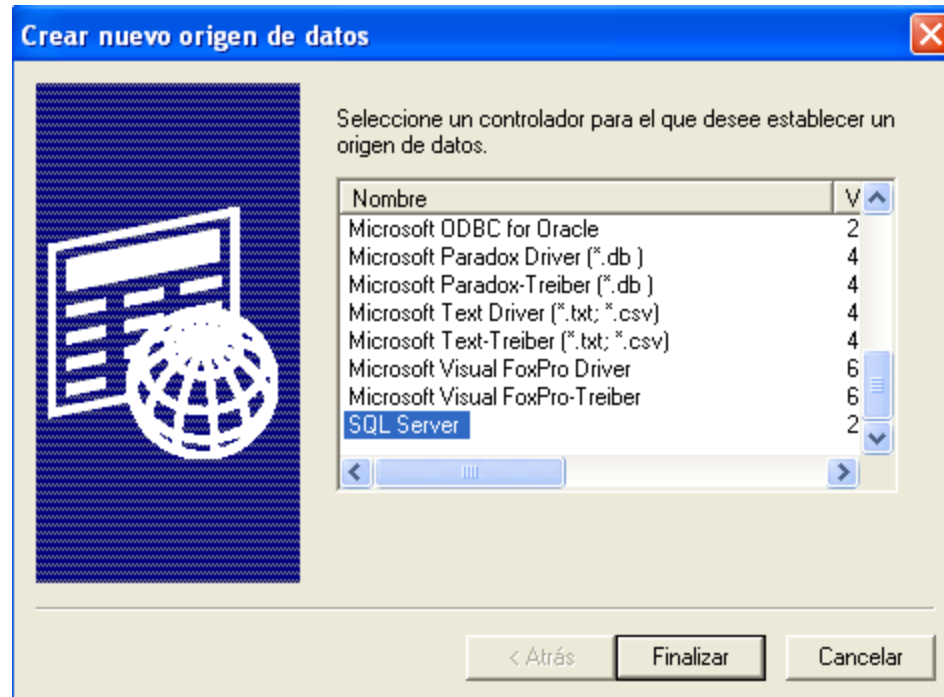
Panel de control → Herramientas Admin → ODBC

Click en el botón **Agregar (DSN del Sistema)**:



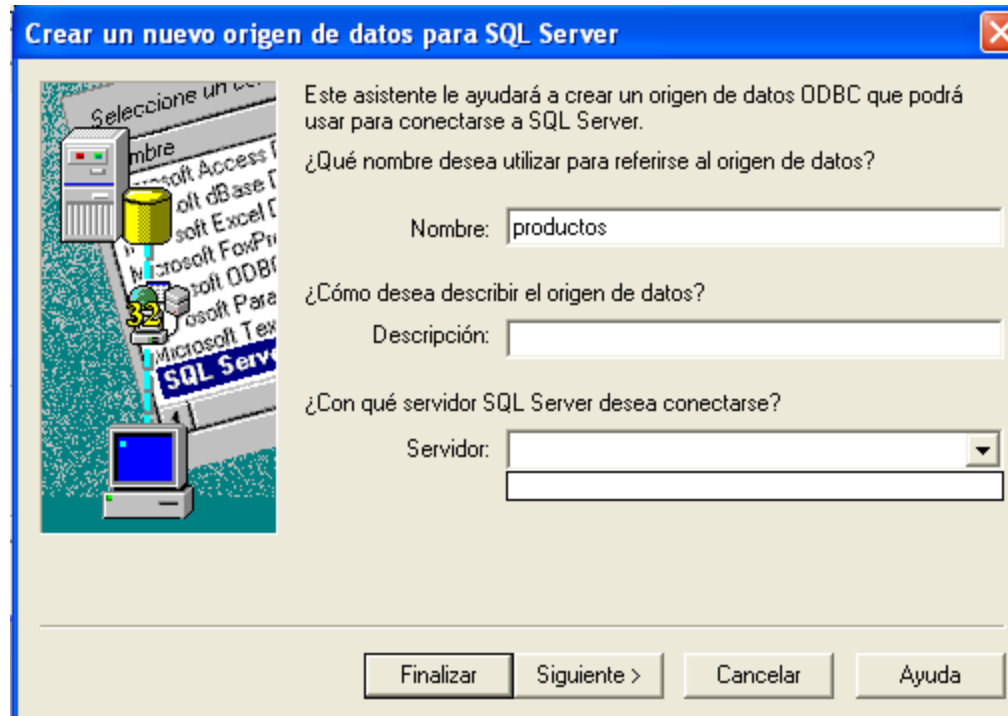
DSN del Sistema ejemplo para SQLServer

- Seleccionar el driver de SQL Server:



DSN del Sistema ejemplo para SQLServer

- Dar un nombre a la conexión y seleccionar el servidor.



Crear un nuevo origen de datos para SQL Server

Este asistente le ayudará a crear un origen de datos ODBC que podrá usar para conectarse a SQL Server.

¿Qué nombre desea utilizar para referirse al origen de datos?

Nombre: productos

¿Cómo desea describir el origen de datos?

Descripción:

¿Con qué servidor SQL Server desea conectarse?

Servidor:

Finalizar Siguiente > Cancelar Ayuda

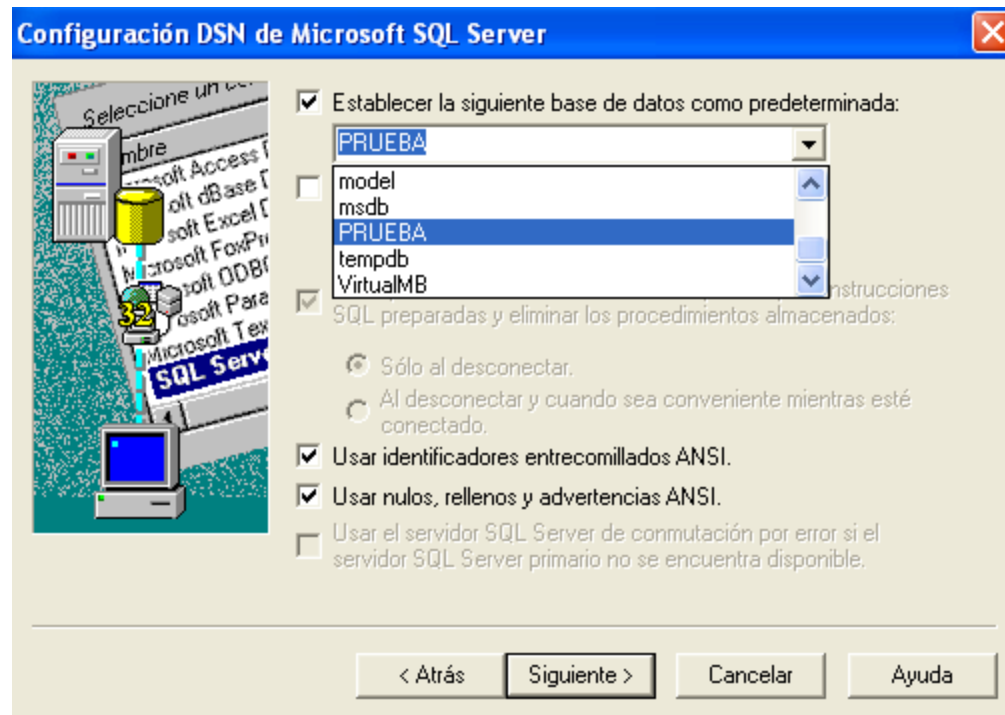
DSN del Sistema ejemplo para SQLServer

- Elegir el tipo de acceso:



DSN del Sistema ejemplo para SQLServer

- Seleccionar la BD a la que queremos acceder:



DSN del Sistema ejemplo para SQLServer

- Pulsar el botón de finalizar:



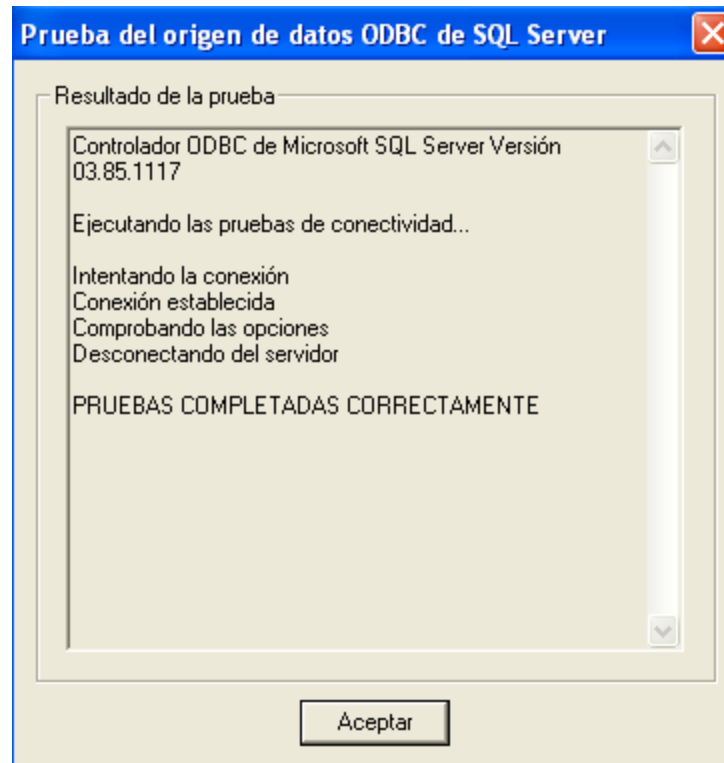
DSN del Sistema ejemplo para SQLServer

- Pulsar el botón Probar conexión:



DSN del Sistema ejemplo para SQLServer

- Pulsar los botones de aceptar:



Conexión con el puente JDBC-ODBC

- Una vez hemos creado la DSN ...
- Tenemos que cargar la clase y capturar una conexión:

// Cargamos la clase:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

// Capturamos la conexión, indicando el nombre de la DSN. En este caso, no necesitamos usuario ni password.

```
conexion = DriverManager.getConnection  
("jdbc:odbc:conexionAccess", "", "");
```

Conexión a MySQL

- El primer paso: adjuntar el driver (el jar) a nuestro proyecto o incluirlo dentro de nuestro servidor, dependiendo del tipo de aplicación.
 - Cargamos la clase del Driver:
`Class.forName("org.gjt.mm.mysql.Driver");`
 - Abrimos la conexión:
`conexion = DriverManager.getConnection
("jdbc:mysql://127.0.0.1/empresa2","root", "");`
- En este caso estamos conectando a MySQL, a una base de datos que se llama empresa2, con el usuario root, y no clave. El Servidor se encuentra en nuestra propia máquina: localhost (127.0.0.1).

Conexión con Oracle

- Al igual que con MySQL necesitamos el Driver (jar).
- Tenemos dos opciones:
 - **Cargar la clase** del driver con:
`Class.forName(...)`
 - **O registrar el driver** de Oracle:
`DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());`
- Después **abrimos la conexión**:
`con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:empresa","SYSTEM","Curso2011");`

Conexión con Oracle

- En el ejemplo anterior estamos conectando a una BD que se llama empresa. El usuario es **SYSTEM** y la password **Curso2011**.
- Con la cadena indicamos:
 - jdbc:oracle:thin:@localhost:1521:empresa
 - jdbc:oracle:thin:
 - El Driver nativo de java.
 - localhost:1521
 - Servidor localhost y puerto 1521.

Después de abrir la conexión

- Una vez hemos abierto la conexión (ya sea con MySQL, con Access a través de la DNS o con Oracle), los siguientes pasos son comunes:
 - Crear sentencias, para poder ejecutar comandos SQL, ejecutar procedimientos almacenados, etc.
 - Al final, cerraremos la conexión.

Interfaces en jdbc

- Una vez que tenemos establecida la conexión con la base de datos, las sentencias select las ejecutaremos a través de la interfaz **Statement**.
- Cuando la select esté parametrizada lo haremos a través de la interfaz **PreparedStatement**.
- Y por último cuando la select la tengamos en un procedimiento almacenado en la base de datos usaremos un interfaz **CallableStatement**.

Interfaz ResultSet

- Los registros de la base de datos se devuelven en un **ResultSet** y permanecen cargados en memoria.
- Cuando se genera un ResultSet, el cursor apuntará por encima del primer registro.
- Un cursor es como una flecha que apunta al registro actual.

Tipos de cursores dentro de ResultSet

- **TYPE_FORWARD_ONLY**: el cursor es solo hacia delante. Se avanza de uno en uno y solo podemos ir del registro actual al siguiente.
- **TYPE_SCROLL_INSENSITIVE**: Trabaja con snapshot, permite desplazamientos en cualquier dirección y podemos ir de un registro a cualquier otro. Es como una imagen de los datos.
- **TYPE_SCROLL_SENSITIVE**: Trabaja con dynaset → son actualizables, nos podemos mover como queramos y cualquier modificación que haga otro usuario lo veremos al instante. Este es el que se suele usar.

Más información sobre la conexión

- ¿Cómo afectan nuestras modificaciones a los datos de la Base de datos?
- **CONCUR_READ_ONLY:** Las modificaciones no se actualizan en la BD.
- **CONCUR_UPDATABLE:** Las modificaciones se actualizan en la BD.

Connection

- Representa la conexión con la BD.
 - Métodos:
 - close() → Cierra la conexión con la BD.
 - Statement createStatement(int ResultSetType, int ResultSetConcurrency)
Crea un Statement a través del cual podremos lanzar SQL a la base de datos.

Gestión de Transacciones

- Las bases de datos soportan transacciones.
- Una transacción es un conjunto de instrucciones que se ejecutan como una sola dejando la BD en un estado consistente.
- Principales sentencias:
 - Hacer efectivo (Commit): Convierte en permanentes los cambios realizados.
 - Deshacer (Rollback): Devuelve la BD al estado que tenia después de la ultima operación hecha efectiva con éxito.

Propiedades de las Transacciones

- Atomicidad: Si la transacción consta de una o mas sentencias se trata como una sola.
- Consistencia: Cuando se completa la transacción deja la BD en un estado valido.
- Aislamiento: Puede interactuar con otras transacciones.
- Durabilidad: Cuando se completa las transacción, los cambios realizados por ella se mantienen.

Control de las transacciones

- Sobre el objeto Connection:
 - void setAutoCommit(boolean): Con true (hace efectivas las transacciones (es el valor por defecto). Con false (se deben hacer efectivas).
 - boolean getAutoCommit(): Devuelve el modo actual.
 - void commit(): Hace efectiva la transacción actual.
 - void rollback(): Devuelve la BD al estado que tenia tras el ultimo commit() realizado con éxito.

Statement

- Representa la sentencia SQL → “select * ...”.
- Métodos:
 - **ResultSet executeQuery(String sql)**
Podemos ejecutar una consulta en la base de datos y recuperar el conjunto de registros. Se utiliza con SELECT.
Este método lanza la excepción SQLException.
 - **int executeUpdate(String sql)**
Para actualizar registros en la BD, se utiliza con INSERT, UPDATE, DELETE. Devuelve el numero de registros afectados.

PreparedStatement

- Hereda de Statement.
- Representa una sentencia SQL parametrizada.
- Los parámetros se representan con ?.
- Los parametros se asignan con los mismos que CallableStatement (para procedimientos almacenados).

Ejemplo

- Partimos de una conexión creada:
- `String sql = "insert into cliente (nombre, codigo) values (?, ?)";`
- `PreparedStatement ps =
conexion.prepareStatement(sql);`
- `ps.setString(1, "Juan");`
- `ps.setInt(2, 34555);`
- `ps.executeUpdate();`

CallableStatement

- Hereda de PreparedStatement.
- Para llamar a procedimientos almacenados.
- La forma de llamarlo: { call nombreProc }
- Ejemplo:
 - String sql = “{ call ajustarPrecios }”
 - CallableStatement cs = connection.prepareCall();
 - cs.executeUpdate();

CallableStatement con parámetros

- Los parámetros se numeran del 1 .. N
- Métodos:
 - `void setDouble(int paramIndex, double x);`
 - `void setFloat(int paramIndex, double x);`
 - `void setInt(int paramIndex, int x);`
 - `void setLong(int paramIndex, long x);`
 - `void setString(int paramIndex, String x);`

Ejemplo

- Partimos de una conexión creada y los parámetros viene por los argumentos de main:
- `conexion =`
- `String sql = "{ call set_price(?, ?) }";`
- `CallableStatement stmt = conexión.prepareCall(sql);`
- `stmt.setInt(1, Integer.parseInt(args[0]));`
- `stmt.setDouble(2, Double.parseDouble(args[1]));`
- `stmt.executeUpdate();`

Parámetros de salida

- Si un procedimiento tiene parámetros de salida, hay que registrarlos antes de llamar al procedimiento, teniendo en cuenta lo siguiente:
 - Utilizamos el método **registerOutParameter**.
 - Y también hay que indicar el tipo del parámetro de salida, ver la clase: **java.sql.Types**.
 - Una vez que hemos ejecutado el procedimiento, **recuperamos** los parámetros de salida **con** **getXXX(int posicion)**.

Ejemplo 1ª parte (Oracle)

- Partiendo de este procedimiento almacenado:
create or replace PROCEDURE OPERACIONES
(numero1 **IN** NUMBER
, numero2 **IN** NUMBER
, suma **OUT** NUMBER
, diff **OUT** NUMBER
) AS BEGIN
 suma:=numero1+numero2;
 diff:=numero1-numero2;
END OPERACIONES;

Ejemplo 2ª parte (Oracle)

```
Connection con=null;
CallableStatement proc;

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:empresa","SYSTEM","Curso2011");
// Crear el procedimiento con 4 parámetros:
proc = con.prepareCall("{call operaciones(?,?,?,?)");
// Los dos parámetros de entrada, indicando posición y un valor dado por las variables a y b:
proc.setInt(1, a);
proc.setInt(2, b);
// Los dos parámetros de salida: posición y tipo:
proc.registerOutParameter(3, java.sql.Types.NUMERIC);
proc.registerOutParameter(4, java.sql.Types.NUMERIC);
// Ejecutar el procedimiento Almacenado:
proc.executeUpdate();
// Recuperar los parámetros de salida:
suma = proc.getInt(3);
diff = proc.getInt(4);

System.out.println("Suma: " + suma + " Diff: " + diff);
```

Llamar a una Function

- Para llamar a una función de la BD utilizamos también CallableStatement.
- En este caso la llamada se haría así:
`con.prepareCall("{?=call IMPORTE_PEDIDO(?)})");`
- En este caso el primer ? Representa el valor devuelto por la función y se tratará como un parámetro de salida.

Ejemplo 1ª parte (Oracle)

- Partimos de esta función de la BD:

```
create or replace FUNCTION IMPORTE_PEDIDO(id_pedido number) RETURN float AS
```

```
vcargo float;  
vdetalles float;  
vtotal float;
```

```
BEGIN
```

```
-- Calcular el cargo:
```

```
vcargo := 0;  
vdetalles := 0;
```

```
select cargo into vcargo from tbpedidos where id = id_pedido;  
select sum(preciounidad * cantidad * (1- descuento)) into vdetalles  
  from tbdetallespedidos where idpedido = id_pedido;
```

```
vtotal:=vcargo + vdetalles;  
return vtotal;
```

```
END IMPORTE_PEDIDO;
```


Ejemplo 2ª parte (Oracle)

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
con = DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:empresa","SYSTEM","Curso2011");

proc = con.prepareCall("{?=call IMPORTE_PEDIDO(?)}")
proc.setInt(2, 10248);
proc.registerOutParameter(1, java.sql.Types.FLOAT);

proc.executeUpdate();
importe = proc.getFloat(1);

System.out.println("Importe: " + importe);
```

ResultSet

- El resultado de la consulta, podremos acceder campo a campo, a cada uno de los registros.
- Métodos:
 - close() → Cierra el ResultSet.
 - first() → Mueve al primero.
 - last() → Mueve al último.
 - next() → Mueve al siguiente.
 - previous() → Mueve al anterior.

Mas métodos de ResultSet

- **String getString("nombreCampo")** → Devuelve el contenido del campo indicado.
- **int getInt("nombreCampo")** → Devuelve el contenido del campo indicado.
- **long getLong("nombreCampo")** → Devuelve el contenido del campo indicado.
- **boolean isBeforeFirst()** → true si el cursor está colocado antes del primer registro, en esta posición no podemos leer.
- **boolean isAfterLast()** → true si el cursor está colocado después del último registro, en esta posición no podemos leer.
- **ResultSetMetaData getMetaData()** → Devuelve el objeto que lleva la información de los campos del resultado, es decir, los nombres de los campos, el tipo, el número de campos.

Interface ResultSetMetaData

- ResultSet nos da el resultado de la consulta → Los datos.
- Con ResultSetMetaData podemos acceder a la información de los campos del resultado, es decir, el nombre de las columnas, el número de columna, el tipo.

Métodos de ResultSetMetaData

- `int getColumnCount()` → Devuelve el número de columnas del ResultSet.
- `String getColumnLabel(int column)` → Devuelve el nombre de la columna. OJO empiezan en 1 y van hasta N.
- `String getColumnName(int column)` → Devuelve el nombre de la columna. OJO empiezan en 1 y van hasta N.

Ejemplo: Conectar con la BD Oracle I

```
public static void main(String[] args) {
    Connection con = null;
    Statement sentencia = null;
    String sql;
    ResultSet tabla;
    ResultSetMetaData cabeceras;
    try {
        // 1. CARGAR LA CLASE DEL DRIVER:
        Class.forName("oracle.jdbc.Driver");

        // 2. ABRIR LA CONEXION:
        con = DriverManager.getConnection("jdbc:oracle:thin:@nombre_maquina:1521:BD","user","password");

        // 3. CREAR UNA SENTENCIA PARA EJECUTAR SQL:
        sentencia = con.createStatement();

        // 4. SQL a ejecutar:
        sql = "select * from personas";

        // 5. EJECUTAR LA SENTENCIA:
        tabla = sentencia.executeQuery(sql);

        // 5.1 CAPTURAR INFORMACIÓN DE LOS CAMPOS:
        cabeceras = tabla.getMetaData();

        // 5.2 CAPTURAR EL NUMERO DE COLUMNAS:
        int numCols = cabeceras.getColumnCount();
    }
}
```

Ejemplo: Conectar con la BD Oracle II

// 6. LEER FILA A FILA EL RESULTADO DEL SQL:

```
while (tabla.next() == true){  
  
    // Imprimir las columnas de cada fila:  
    for (int i = 1 ; i <= numCols ; i++)  
        System.out.print(tabla.getString(i) + " ");  
  
        // Salto de linea:  
        System.out.println();  
}
```

// 7. Cerrar la sentencia:

```
tabla.close();
```

// 8. Cerrar la conexión con la BASE de datos:

```
con.close();
```

```
} catch (ClassNotFoundException e) {  
    e.printStackTrace();  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

```
}
```