

# Manejo de Conectores en Java

I

## Índice

---

- ▶ Introducción
  - ▶ El desfase objeto-relacional
  - ▶ Bases de datos embebidas
  - ▶ Protocolos de acceso a bases de datos
  - ▶ Acceso a datos mediante ODBC
  - ▶ Acceso a datos mediante JDBC
  - ▶ Establecimiento de conexiones
  - ▶ Ejecución de sentencias de descripción de datos
  - ▶ Ejecución de sentencias de modificación de datos
  - ▶ Ejecución de procedimientos
  - ▶ Generación de informes
  - ▶ Gestión de errores
- 

▶ 2

## Introducción

- ▶ Por acceso a datos entendemos todo proceso de manipulación de información obtenida de una fuente de datos local o remota, cuya naturaleza puede ser de lo más variada.
- ▶ Las fuentes de datos pueden ser ficheros con multitud de formatos, bases de datos relacionales tanto locales como remotas, hojas de cálculo, bases de datos con otras arquitecturas: jerárquicas, orientadas a objetos, etc.
- ▶ Los conectores son el software necesario para realizar conexiones desde un programa Java con una base de datos relacional.

▶ 3

## El desfase objeto-relacional

- ▶ Las bases de datos orientadas a objetos van ganando terreno frente a las tradicionales bases de datos relacionales porque solucionan necesidades a las que no pueden dar respuesta con el modelo relacional: aplicaciones de ingeniería, sistemas de información geográfica GIS, experimentos científicos, multimedia, etc.
- ▶ Los objetos son elementos complejos sobre los que se fundamenta la POO. Las bases de datos relacionales no son capaces de almacenar este tipo de elementos.
- ▶ El almacenamiento de objetos es más complejo que el de las tablas relacionales; esto origina el desfase objeto-relacional.
- ▶ Cada vez que los objetos deben almacenarse/extraerse en una base de datos relacional se requiere un mapeo desde las estructuras del modelo relacional a las del entorno de programación.

▶ 4

## Bases de datos embebidas (I)

- ▶ Una base de datos embebida se utiliza cuando esta no contiene gran cantidad de datos. El motor de estas bases de datos queda incrustado en la aplicación y es de uso exclusivo en dicha aplicación.
- ▶ **SQLITE**
  - ▶ Es un SGBD escrito en C donde las bases de datos se guardan en forma de ficheros.
  - ▶ La biblioteca implementa la mayor parte del estándar SQL-92: transacciones, *triggers*, consistencia y la mayoría de consultas complejas.
  - ▶ Instalación. Desde la página <http://www.sqlite.org/download.html> descargar el fichero zip para la versión de sistema operativo deseado: Por ejemplo en **Precompiled Binaries for Windows** el zip correspondiente.
  - ▶ Para ejecutar desde la línea de comandos se necesita el fichero `sqlite3.exe`, escribiremos el nombre de la base de datos, que si no existe, se creará.

▶ 5

## 2.3 Bases de datos embebidas (II)

### ▶ Ejemplo SQLITE

```
D:\>sqlite3 d:\ejemplo.db
SQLite version 3.7.14 2012-09-03 15:42:36
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> BEGIN TRANSACTION;
sqlite> CREATE TABLE departamentos (
...> dept_no TINYINT(2) NOT NULL PRIMARY KEY,
...> dnombre VARCHAR(15),
...> loc VARCHAR(15)
...> );
sqlite>
sqlite> INSERT INTO departamentos VALUES (10, 'CONTABILIDAD', 'SEVILLA');
sqlite> INSERT INTO departamentos VALUES (20, 'INVESTIGACIÓN', 'MADRID');
sqlite> INSERT INTO departamentos VALUES (30, 'VENTAS', 'BARCELONA');
sqlite> INSERT INTO departamentos VALUES (40, 'PRODUCCIÓN', 'BILBAO');
sqlite> COMMIT;
```

▶ 6 `sqlite> .quit`

## Bases de datos embebidas (III)

### ► APACHE DERBY

- Es una base de datos relacional de código abierto implementada en Java.
- Soporta estándares SQL y ofrece un controlador integrado JDBC que permite incrustar Derby en cualquier solución basada en Java.
- Soporta el paradigma cliente servidor.
- Instalación. Desde la página.  
[http://db.apache.org/derby/derby\\_downloads.html](http://db.apache.org/derby/derby_downloads.html)
- se descarga la última versión: dbderby-10.15.1.3-bin.zip y se descomprime la carpeta que contiene.
- Para utilizar derby en los programas Java será necesario tener accesible la librería `derby.jar` en el `CLASSPATH` de nuestro programa.
- Apache Derby incluye un fichero `IJ.BAT` que permite ejecutar por consola órdenes para crear bases de datos y ejecutar sentencias DDL y DML.
  - `D:\db-derby-10.14.2.0-bin\bin>ij`
- Para crear una BD de nombre ejemplo en la carpeta C:\bddd\derby escribimos, desde el indicador `ij>` lo siguiente:
  - `connect 'jdbc:derby:C:\bddd\derby\ejemplo;create=true';`

► 7

## Bases de datos embebidas (IV)

### ► APACHE DERBY

- `Connect` es el comando para establecer la conexión.
- `jdbc:derby` es el protocolo JDBC especificado por DERBY.
- `ejemplo` es el nombre de la base de datos que se va a crear (en realidad se creará un directorio de nombre `ejemplo` con varios ficheros y directorios).
- `create=true` es el atributo usado para crear la base de datos.
- Para salir de la línea de comandos de `IJ` se escribe `exit`.
- Para ejecutar scripts en la línea de comandos se utiliza `run` seguido del nombre del script (puede ser con ruta), **entre comillas simples**.
- Para volver a utilizar una base de datos:  
`connect 'jdbc:derby:C:\bddd\derby\ejemplo';`

► 8

## Bases de datos embebidas (V)

### ► HSQLDB

#### ► (Hyperthreaded Structured Query Language Database)

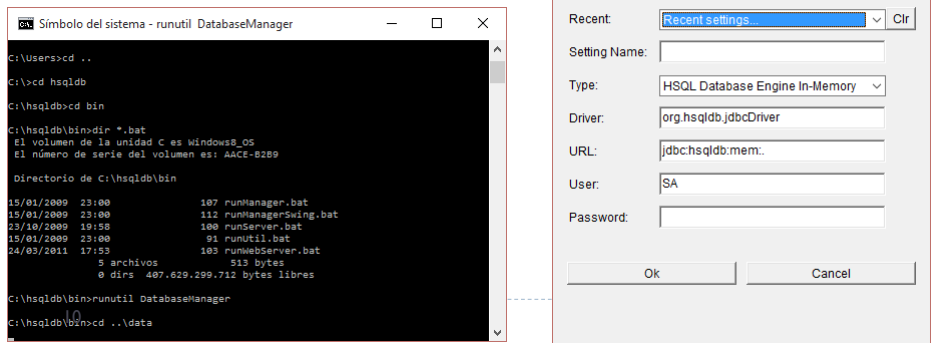
- Es un SGBD relacional escrito en Java. La suite ofimática OpenOffice desde la ver 2.0 lo incluye para dar soporte a la aplicación Base.
- Puede mantener la base de datos en memoria o en ficheros en disco.
- Permite integridad referencial, procedimientos almacenados en Java, disparadores y tablas de hasta 8Gb.
- Descargamos desde <http://sourceforge.net/projects/hsqldb/files> descargamos la última versión para Windows hsqldb-2.6.1.zip. Al descomprimirlo obtendremos el fichero hsqldb que copiaremos en la carpeta C:\bbdd\hsqldb.

► 9

## Bases de datos embebidas (VI)

### ► HSQLDB

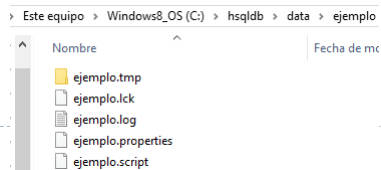
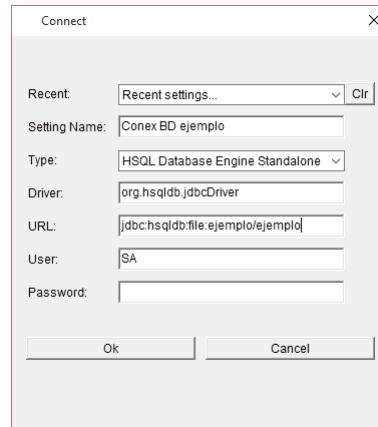
- Creamos la carpeta C:\bbdd\hsqldb\data para guardar ahí los ficheros de base de datos que creamos (puede que exista vacía).
- Desde c:\bbdd\hsqldb\bin ejecutamos el archivo runManagerSwing.BAT con el parámetro DatabaseManager para conectarnos a la BD y que se ejecute la interfaz gráfica del SGBD.



## Bases de datos embebidas (VII)

### ► HSQLDB

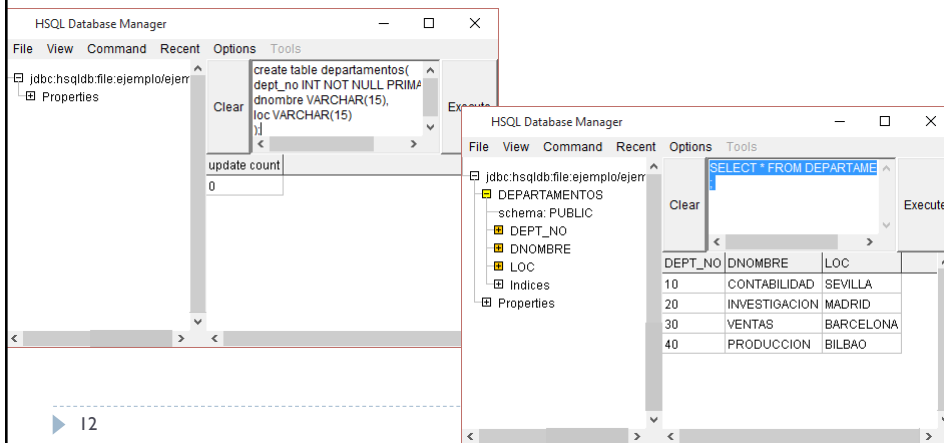
- Los parámetros a configurar son:
  - **Setting Name**: nombre de la conexión.
  - **Type**: seleccionamos *HSQL Database Engine Standalone*. (De esta forma la base de datos la tomará de un fichero, y si no existe, lo crea)
  - **URL**: el nombre de la carpeta donde se almacenará la base de datos y el nombre del fichero de base de datos. Si la carpeta no existe la crea y dentro de ella genera un conjunto de ficheros.



## Bases de datos embebidas (VIII)

### ► HSQLDB

- En la ventana que se abre se pueden introducir comandos tando DDL como DML.



## Bases de datos embebidas (IX)

### ► H2

- Es un SGBD relacional escrito en Java.
- Es rápido, de código abierto, y basado en el API JDBC.
- Puede utilizarse como base embebida o como base de datos en servidor.
- Se puede descargar de <http://www.h2database.com/html/main.html>
- Se trata de un fichero ejecutable con un instalador.
- Al ejecutarlo se descomprime en el directorio que tendremos que especificar como destino, por ejemplo C:\h2.
- En la carpeta bin instalada, C:\h2\bin se ejecuta el fichero h2.bat.

► 13

## Bases de datos embebidas (X)

### ► H2

- Se le puede dar un nombre a la configuración: **conex H2**
- Y un valor a la url de la conexión con el nombre de nuestra base de datos:  
**jdbc:h2:C:\H2\ejemplo\ejemplo**
- Se guarda la configuración, se prueba la conexión y se conecta.

The screenshot shows a web browser window with the URL `192.168.1.128:8082/test.do?jsessionid=6f978f594edce22`. The page has a menu bar with 'English', 'Preferencias', 'Tools', and 'Ayuda'. The main form is titled 'Registrar' and contains the following fields and buttons:

- Configuraciones guardadas:** A dropdown menu showing 'Generic H2 (Embedded)'.
- Nombre de la configuración:** A text input field containing 'Generic H2 (Embedded)', with 'Guardar' and 'Eliminar' buttons to its right.
- Controlador:** A text input field containing 'org.h2.Driver'.
- URL JDBC:** A text input field containing 'jdbc:h2:~\test'.
- Nombre de usuario:** A text input field containing 'sa'.
- Contraseña:** A text input field.
- Buttons:** 'Conectar' and 'Probar la conexión' at the bottom of the form.

Prueba correcta

► 14

## Bases de datos embebidas (XI)

### ► H2

The screenshot shows the H2 database web interface at the URL `192.168.1.128:8082/login.do?jsessionid=9666ab4e2df3fd6c19d5fe50ea8ffba6`. The interface includes a toolbar with buttons for `Ejecutar`, `Run Selected`, `Auto completado`, and `Eliminar`. Below the toolbar, there is a section titled **Comandos importantes** (Important Commands) with a table of shortcuts:

Icono	Descripción
?	Visualizar esta página de ayuda.
🔍	Ver histórico de comandos
▶	Ctrl+Enter Ejecuta la actual sentencia SQL
▶	Shift+Enter Ejecutes the SQL statement defined by the text selection
▶	Ctrl+Space Auto completado
🔌	Desconectar de la base de datos.

Below the table, there is a section titled **Ejemplo SQL Script**.

► 15

## Bases de datos embebidas (XII)

### ► H2

The screenshot shows the H2 database web interface with the same URL as the previous slide. The `Información de esquema` (Schema Information) pane on the left shows the database structure, including `DEPARTAMENTOS`, `EMPLEADOS`, and `INFORMATION_SCHEMA`. The `Instrucción SQL:` (SQL Statement) pane contains the following query:

```
select apellido from empleados
where fecha_alta between '1980-01-01' and '1990-12-31'
```

The `Resultado` (Result) pane shows the results of the query:

Apellido
López
Santana
Gil
Iglesias
Soroya

At the bottom of the results pane, it indicates `(5 filas, 3 ms)`.

► 16



## Bases de datos embebidas (XIII)

### ► DB4O

#### ► (DataBase 4 for Objects)

- Es un motor de bases de datos orientado a objetos, disponible para entornos Java y .NET.
- Características:
  - Se evita el problema del desfase objeto-relacional.
  - No existe un lenguaje SQL para la manipulación de datos, sino métodos delegados.
  - Se instala añadiendo un único fichero de biblioteca (JAR para Java o DLL para .NET).
  - Se crea un único fichero de base de datos con la extensión .YAP.
- Se descarga desde <http://supportservices.actian.com/versant/default.html> el archivo db4o-8.0.276.16149-java.zip. Al descomprimirlo hay una carpeta lib donde se encuentran los JAR (db4o-8.0.276.16149-all-java5.jar) que hay que agregar a nuestro proyecto para utilizar el motor de base de datos.

► 17

## Bases de datos embebidas (XIV)

### ► DB4O

- Si queremos utilizar en un proyecto una base de datos que almacene personas, lo primero será crear la clase Persona, con los atributos deseados (nombre y ciudad), junto con los métodos get, set, y los constructores.
- Para abrir/crear la base de datos:

```
ObjectContainer bd =  
Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), BDPer);
```
- Para cerrar la base de datos:

```
bd.close();
```
- Para almacenar un objeto en la base de datos:

```
bd.store(p1);
```

► 18

## Bases de datos embebidas (XV)

```
import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;

public class EjemDB4o {
    static String BDPPer = "BDPersonas.yap";

    public static void main(String[] args) {
        ObjectContainer bd = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), BDPPer);

        // Creamos Personas
        Persona p1 = new Persona("Juan", "Guadalajara");
        Persona p2 = new Persona("Ana", "Madrid");
        Persona p3 = new Persona("Luis", "Granada");
        Persona p4 = new Persona("Pedro", "Asturias");

        bd.store(p1); // Almacenar objetos Persona en la base de datos
        bd.store(p2);
        bd.store(p3);
        bd.store(p4);

        bd.close(); // cerrar base de datos
    }
}
```

## Bases de datos embebidas (XVI)

### ► DB4O

- Para recuperar objetos de la base de datos lo podemos hacer mediante consultas QBE (*Query-By-Example*). Se utiliza el método **queryByExample()**, pasándole como argumento un objeto (en este caso de tipo persona), en el que hemos establecido el filtro, por ejemplo poniendo un valor para el campo nombre.
- Por ejemplo si quisiéramos obtener todas las personas, crearíamos un objeto de tipo persona, con sus atributos a null, de esta forma se obtendrían todos los objetos independientemente del valor de sus atributos. Si quisiéramos obtener todas las personas de “Madrid”, crearíamos un objeto con el nombre a *null* y la localidad a “Madrid”.

## Bases de datos embebidas (XVII)

DB4O

```
Persona per = new Persona(null, null);
ObjectSet<Persona> result = bd.queryByExample(per);
if (result.size() == 0)
    System.out.println("No existen Registros de
Personas..");
else {
    System.out.printf("Número de registros: %d %n",
result.size());

    while (result.hasNext()) {
        Persona p = result.next();
        System.out.printf("Nombre: %s, Ciudad: %s %n",
p.getNombre(), p.getCiudad());
    }
}
```

► 21

## Bases de datos embebidas (XVIII)

### ► DB4O

- Para modificar o borrar un objeto primero hay que localizarlo, con una consulta.
- En el caso de las modificaciones se recupera el objeto, se modifican los atributos que se deseen y se vuelve a guardar con el método **store()** ya visto. Hay que verificar si se trata sólo de uno o de varios en cuyo caso habría que hacer un bucle.
- En el caso de los borrados, una vez localizado el elemento, se elimina con el método **delete()**, de esta forma `bd.delete(p)`;

► 22

## Bases de datos embebidas (XIX)

DB4O

```
ObjectSet<Persona> result = db.queryByExample(new
Persona("Juan", null));

if (result.size() == 0)
    System.out.println("No existe Juan...");
else {
    System.out.printf("Registros a borrar: %d %n",
result.size());

    while (result.hasNext()) {
        Persona p = result.next();
        db.delete(p);
        System.out.println("Borrado....");
    }
}
```

► 23

## Protocolos de acceso a bases de datos

- Existen dos normas de conexión a una base de datos SQL:
  - **ODBC**. (*Open Data Base Connectivity*). Define un API que pueden usar las aplicaciones para abrir una conexión con una base de datos, enviar consultas o actualizaciones.
  - **JDBC** (*Java Data Base Connectivity*). Define un API que pueden usar los programas Java para conectarse a los servidores de bases de datos relacionales.
- **OLE\_DB** de Microsoft, es una API de C++ para datos que no son bases de datos. Permite conexión con orígenes de datos, ejecución de comandos y devolución de resultados en forma de conjuntos de filas.
- **ADO** es un API, también de Microsoft, que ofrece una interfaz sencilla de utilizar con la funcionalidad OLE-DB, que puede llamarse desde los lenguajes de scripts.

► 24

## Acceso a datos mediante ODBC (I)

- ▶ Estándar de acceso a datos de Microsoft para posibilitar el acceso a datos desde cualquier aplicación con independencia del gestor de base de datos.
- ▶ Cada SGBD compatible con ODBC debe proporcionar una biblioteca que se debe enlazar con el programa cliente.
- ▶ Cuando el programa cliente hace una llamada a la API ODBC el código de la biblioteca se comunica con el servidor para realizar la acción solicitada y obtener los resultados.

▶ 25

## 2.5. Acceso a datos mediante ODBC (II)

- ▶ **Pasos para usar ODBC:**
  - ▶ Configurar la interfaz ODBC, asignando un entorno SQL con la función `SQLAllocHandle()` y un manejador (*handle*) para la conexión a la base de datos en el entorno anterior. Así, el manejador traducirá las consultas de datos de la aplicación en comandos que el SGBD entienda. ODBC define los siguientes manejadores:
    - ▶ `SQLHENV`. Define el entorno de acceso a datos.
    - ▶ `SQLHDBC`. Identifica el estado y configuración de la conexión.
    - ▶ `SQLHSTMT`. Declaración SQL y conjunto de resultados asociados.
    - ▶ `SQLHDESC`. Recolección de metadatos usados para describir sentencias SQL.

▶ 26

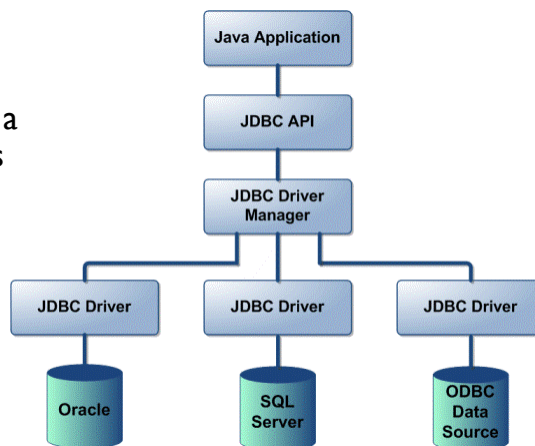
## Acceso a datos mediante ODBC (III)

- ▶ **Pasos para usar ODBC (continuación)**
  - ▶ Una vez reservados los manejadores, el programa abre la conexión a la BD usando `SQLDriverConnect()` o `SQLConnect()`.
  - ▶ Una vez realizada la conexión el programa puede enviar órdenes SQL a la BD utilizando `SQLExecDirect()`.
  - ▶ Al finalizar la sesión el programa se desconecta de la base de datos y libera la conexión y los manejadores del entorno SQL.
- ▶ La API ODBC tiene una interfaz escrita en C y no es apropiada para su uso directo desde Java; existen inconvenientes de seguridad, implementación, robustez y portabilidad de las aplicaciones.

▶ 27

## Acceso a datos mediante JDBC (I)

- ▶ JDBC proporciona una biblioteca estándar para acceder a fuentes de datos principalmente orientados a bases de datos relacionales que usan SQL.
- ▶ JDBC dispone de una interfaz distinta para cada base de datos; un driver o conector, lo que permite que las llamadas a los métodos Java de las clases JDBC se correspondan con el API de la base de datos.



▶ 28

## Acceso a datos mediante JDBC (II)

- ▶ JDBC consta de un conjunto de clases e interfaces que nos permiten escribir aplicaciones Java para gestionar las siguientes tareas con una base de datos relacional:
  - ▶ Conectarse a la base de datos.
  - ▶ Enviar consultas e instrucciones de actualización a la base de datos.
  - ▶ Recuperar y procesar los resultados recibidos de la base de datos en respuesta a las consultas.

▶ 29

## Acceso a datos mediante JDBC (III)

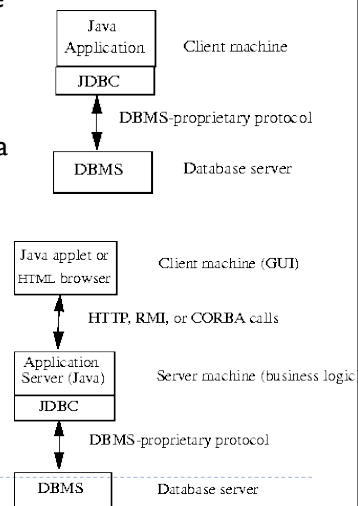
### ▶ Arquitecturas JDBC

- ▶ La API JDBC es compatible con los modelos de dos y tres capas para el acceso a BD.

- ▶ **Modelo de 2 capas.** Desde el programa Java se envían sentencias SQL al SGBD, que las procesa y devuelve al programa. El driver se encarga de manejar la comunicación a través de la red de forma transparente al programa.

- ▶ **Modelo de 3 capas.** Los comandos se envía a una capa intermedia que se encarga de enviar los comandos SQL a la base de datos y de recoger los resultados de la ejecución de las sentencias.

▶ 30



## Acceso a datos mediante JDBC (IV)

### ► Tipos de drivers

- **Tipo 1. JDBC-ODBC Bridge.** Permite el acceso a bases de datos JDBC mediante un driver ODBC. Exige la instalación de ODBC en la máquina cliente.
- **Tipo 2. Native.** Traduce las llamadas al API de JDBC Java en llamadas propias del motor de base de datos. Exige instalar en la máquina cliente, código binario propio del cliente de base de datos y del sistema operativo.
- **Tipo 3. Network.** Controlador de Java puro que utiliza un protocolo de red para comunicarse con un servidor de base de datos. Traduce las llamadas al API de JDBC Java en llamadas propias del protocolo de red usado por el motor de BD. No exige instalación en cliente.
- **Tipo 4. Thin.** Controlador de Java puro: Traduce las llamadas al API de JDBC Java en llamadas propias del protocolo de red usado por el motor de la base de datos. No exige instalación en cliente.

► 31

## Acceso a datos mediante JDBC (V)

### ► Cómo funciona JDBC

- JDBC define varias interfaces que permiten realizar operaciones con bases de datos. A partir de ellas se derivan las clases correspondientes, definidas en el paquete java.sql.
- <https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>

► 32



## Acceso a datos mediante JDBC (VI)

► **Los pasos a seguir para utilizar JDBC en una aplicación Java son:**

- Importar las clases necesarias.
- Cargar el driver JDBC.
- Identificar el origen de datos.
- Crear un objeto *Connection*.
- Crear un objeto *Statement*.
- Ejecutar consultas con el objeto *Statement*.
- Obtener los resultados mediante *ResultSet*.
- Liberar los objetos *ResultSet*, *Statement* y *Connection*.

► **Ejemplo: Establecer una conexión con MySQL**

- Se crea una base de datos y un usuario de nombre EJEMPLO, con esta clave.
- Se crean las tablas departamentos y empleados
- Se crea la aplicación Java
  - Incluir el driver MySQL

► 33

## Acceso a datos mediante JDBC (VII)

```
import java.sql.*;
public class PruJDBC {
public static void main(String[] args) {
try{
    Connection conexion =
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo", "ejemplo", "ejemplo");
    Statement sentencia = conexion.createStatement();
    ResultSet resul = sentencia.executeQuery("SELECT * FROM departamentos");
    while (resul.next()) {
        System.out.println(resul.getInt(1) + " " + resul.getString(2) + " " +
resul.getString(3));
    }
} catch( SQLException e){
    System.out.println(e);
}
}
}
```

► 34

## Acceso a datos mediante el puente JDBC-ODBC (I)

- ▶ Para aquellos productos donde no hay controlador JDBC y sí uno ODBC existe un puente JDBC-ODBC. Se trata de un driver que implementa operaciones JDBC traduciéndolas en operaciones ODBC.
- ▶ El puente se instala automáticamente con el JDK como el paquete sun.jdbc.odbc (**no disponible a partir de la versión 8**).
- ▶ Ejemplo: Para acceder a una base de datos MySQL usando el puente JDBC-ODBC hay que crear un origen de datos on DNS (*Data Source Name*). En Windows sería:
  - ▶ Panel de Control-> Herramientas Administrativas -> Origen de datos (ODBC)
  - ▶ Una vez ahí, Agregar y seleccionar el driver ODBC para MySQL y Finalizar.
  - ▶ Damos nombre al origen de los datos; por ejemplo Mysql-odbc.
  - ▶ Escribimos el nombre de la máquina (dirección IP) donde se encuentra ese origen. Si es la misma máquina, ponemos localhost.
  - ▶ Como nombre de usuario ponemos uno que exista en la base de datos, como, ejemplo y su password.
  - ▶ De la lista DATABASE elegimos un esquema de base de datos, en este caso EJEMPLO y OK.

▶ 35

## Acceso a datos mediante el puente JDBC-ODBC (II)

- ▶ Por cada esquema de base de datos es necesario crear un origen de datos. Desde *Connect Options* podemos escribir el número de puerto por el que escucha MySQL; por defecto 3306.
- ▶ A continuación, se visualiza la pantalla inicial del Administrador de Orígenes de datos ODBC con el nuevo origen creado y pulsamos Aceptar.
- ▶ Si el driver ODBC para conectar MySQL no está instalado, lo descargamos de:  
<http://www.mysql.com/products/connector/>
  - ▶ ODBC Driver for MySQL (Connector/ODBC)
- ▶ Existen versiones de otros desarrolladores (*trial 30 días*):  
<https://www.devart.com/odbc/mysql/download.html>

▶ 36

### Acceso a datos mediante el puente JDBC-ODBC (III)

- En el programa Java que estamos siguiendo cambiamos dos líneas

```
// Cargar el driver
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
//Establecemos la conexión con la base de datos
Connection conexión =
DriverManager.getConnection("jdbc:odbc:Mysql-odbc");
```

(o los valores especificados por el driver de algún otro desarrollador)

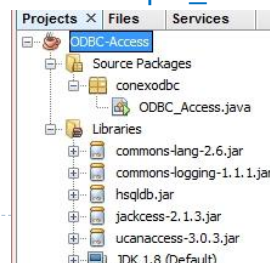
► 37

### Acceso a datos mediante el puente JDBC-ODBC (IV)

- Por ejemplo para conectar con una base de datos Access mediante el driver Ucanaccess ( <http://ucanaccess.sourceforge.net/site.html> ):

```
// Cargar el driver
Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
//Establecemos la conexión con la base de datos
Connection conexión =
DriverManager.getConnection("jdbc:ucanaccess://pru_odbc.mdb");
```

Habría que haber incluido las siguientes bibliotecas:



► 38

## Establecimiento de conexiones (I)

- ▶ Vamos a conectarnos a través de JDBC a las bases de datos embebidas vistas anteriormente.
- ▶ Para ello hay que tener creada la base de datos EJEMPLO con las tablas EMPLEADOS y DEPARTAMENTOS.
- ▶ Se puede utilizar la misma aplicación Java y sólo cambiar la carga del driver y la conexión a la base de datos.

▶ 39

## Establecimiento de conexiones (II)

### ▶ Conexión a SQLite

- ▶ Biblioteca sqlite-jdbc-3.7.2.jar

▶

[http://www.java2s.com/Code/Jar/s/Downloadssqlitejdbc372jar.htm#google\\_vignette](http://www.java2s.com/Code/Jar/s/Downloadssqlitejdbc372jar.htm#google_vignette)

...

Connection conexion =

DriverManager.getConnection("jdbc:sqlite:C:/BBDD/sqlite/ejemplo.db");

...

▶ 40

## Establecimiento de conexiones (III)

### ► Conexión a Apache Derby.

#### ► Bibliotecas

- ☐ derby.jar
- ☐ derbytools.jar
- ☐ derbyshared.jar

(facilitadas ambas en la carpeta lib con la base de datos).

...

Connection conexion =

DriverManager.getConnection("jdbc:derby:C:/BBDD/derby/ejemplo");

...

► 41

## Establecimiento de conexiones (IV)

### ► Conexión a HSQLDB.

- Biblioteca hsqldb.jar (facilitada en la carpeta lib con la base de datos).

...

Connection conexion =

DriverManager.getConnection("jdbc:hsqldb:file:C:/bbdd/hsqldb/dat  
a/ejemplo");

...

► 42

## Establecimiento de conexiones (V)

### ► Conexión a H2.

- Biblioteca h2-1.4.200.jar (facilitada en la carpeta bin con la base de datos).

...

```
Connection conexion =  
DriverManager.getConnection("jdbc:h2:C:/BBDD/h2/ejemplo");
```

...

► 43

## Establecimiento de conexiones (VI)

### ► Conexión a MySQL.

- Con versiones antiguas, biblioteca mysql-connector-java-5-1-38-bin.jar, y fichero com.mysql.jdbc.Driver.

► ...

```
Connection conexion =  
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo","ejemplo","ejemplo");
```

► ...

- A partir de la versión 8 de Java, mysql-connector-java-8.0.27.jar
- Al instalar MySQL se instala en la carpeta:
  - C:\Program Files (x86)\MySQL\Connector J 8.0

...

```
Connection conexion =  
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo","ejemplo","ejemplo");
```

...

► 44

## Establecimiento de conexiones (VII)

### ► Conexión a MySQL.

#### ► Si se produce la excepción

java.sql.SQLException: The server time zone value 'Hora estándar romance' is unrecognized or represents more than one time zone. You must configure either the server or JDBC driver (via the serverTimezone configuration property) to use a more specific time zone value if you want to utilize time zone support.

Se ejecutan estas instrucciones en el servidor MySQL:

```
SET @@global.time_zone = '+01:00';  
SET @@session.time_zone = '+01:00';
```

#### ► Las últimas versiones y documentación en:

► <https://dev.mysql.com/downloads/connector/j/>

► 45

## Establecimiento de conexiones (VIII)

### ► Conexión a Oracle.

#### ► driver JDBC Thin.

#### ► Hay que elegir el driver para la versión de la base de datos

► <http://www.Oracle.com/technetwork/database/features/jdbc/index-091264.html>

►  
...

Connection conexion =

```
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:  
XE", "ejemplo", "ejemplo");
```

...

► 46

## Ejecución de sentencias de descripción de datos (I)

- ▶ Cuando desarrollamos una aplicación JDBC conocemos la estructura de las tablas, los datos que estamos manejando y las relaciones que hay.
- ▶ Si desconocemos la estructura de la base de datos, puede obtenerse a través de los **metaobjetos**.
- ▶ El objeto **DatabaseMetaData** proporciona información sobre la base de datos a partir de los métodos que contiene.
- ▶ Podemos conectar con la BD MySQL de nombre ejemplo y mostrar información sobre el producto, el driver, la URL, el nombre de usuario y las tablas y vistas del esquema actual.

▶ 47

## Ejecución de sentencias de descripción de datos (II)

```
public class DatosBase {
    public static void main(String[] args) {
        try {
            Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/ejemplo", "ejemplo", "ejemplo");

            DatabaseMetaData dbmd=conexion.getMetaData();

            ResultSet result = null;
            String nombre= dbmd.getDatabaseProductName();
            String driver= dbmd.getDriverName();
            String url= dbmd.getURL();
            String usuario= dbmd.getUserName();

            System.out.println("(INFORMACIÓN BASE DATOS");
            System.out.println("Nombre: "+nombre);
            System.out.println("Driver: "+driver);
            System.out.println("URL: "+url);
            System.out.println("Usuario: "+usuario);
```

▶ 48



## Ejecución de sentencias de descripción de datos (III)

```
// información sobre tablas y vistas
result= dbmd.getTables(null, "ejemplo", null, null);
while (result.next()){
    String catalogo= result.getString(1);
    String esquema= result.getString(2);
    String tabla= result.getString(3);
    String tipo= result.getString(4);
    System.out.println(tipo + " - Catálogo: "+ catalogo+
        ", Esquema: "+esquema+", Nombre: "+tabla);
}
conexion.close();
}
catch ( ClassNotFoundException | SQLException cn ) {cn.printStackTrace();}
}
```

► 49

## Ejecución de sentencias de descripción de datos (IV)

- El método `getTables()` devuelve información sobre las tablas y vistas. Sus parámetros son:
  - Primero: catálogo de la base de datos. null indica todos los catálogos.
  - Segundo: esquema de la base de datos. Obtiene las tablas del esquema indicado. null indica el esquema actual (en algunos SGBD significa todos, p.e. en Oracle)
  - Tercero: patrón para seleccionar las tablas. Se puede utilizar `_` o `%`. Ej. “de%”, correspondería a todas las tablas que empiecen por de.
  - Cuarto: es un array de String en el que se indica qué tipos de tabla se desean:
    - `TABLE`, para tablas
    - `VIEW`, para vistas
    - `null`, todos los tipos

► 50

## Ejecución de sentencias de descripción de datos (V)

- ▶ Con `result.getString()`, podríamos utilizar las siguientes constantes:

- ▶ `TABLE_CAT`, nombre del catálogo al que pertenece.
- ▶ `TABLE_SCHEM`, nombre del esquema.
- ▶ `TABLE_NAME`, nombre de la tabla o vista.
- ▶ `TABLE_TYPE`, el tipo TABLE o VIEW.
- ▶ `REMARKS`, comentarios.

```
String catalogo= result.getString("TABLE_CAT");
String esquema= result.getString("TABLE_SCHEM");
String tabla= result.getString("TABLE_NAME");
String tipo= result.getString("TABLE_TYPE");
```

▶ 51

## Ejecución de sentencias de descripción de datos (VI)

- ▶ Otros métodos importantes del objeto `DatabaseMetaData` son:

- ▶ `getColumns(catálogo, esquema, nombre_tabla, nombre_columna)` devuelve información sobre las columnas filtradas. Se puede utilizar `_` y `%`. Si se pasa null en los cuatro parámetros se devuelve información sobre todas las columnas y tablas del esquema actual.

```
ResultSet columnas=null;
columnas= dbmd.getColumns(null, null, null, null);
while (columnas.next()){
    String name= columnas.getString("TABLE_NAME");//getString(3);
    String nombreCol= columnas.getString("COLUMN_NAME");//getString(4);
    String tipoCol= columnas.getString("TYPE_NAME");//getString(6);
    String tamCol= columnas.getString("COLUMN_SIZE");//getString(7);
    String nula= columnas.getString("IS_NULLABLE");//getString(18);
    System.out.println("Tabla: "+name+" Columna: "+nombreCol+", Tipo: "+
tipoCol +
    ", Tamaño: "+tamCol+", ¿Es nula?: "+ nula);
}
```

▶ 52

## Ejecución de sentencias de descripción de datos (VII)

### ► Otros métodos importantes del objeto DatabaseMetaData son:

- `getPrimaryKeys(catálogo, esquema, nombre_tabla)` devuelve la lista de columnas que forman la clave primaria.
- `getExportedKeys(catálogo, esquema, nombre_tabla)` devuelve la lista de las claves ajenas que utilizan la clave primaria de esta tabla.

```
ResultSet pk= dbmd.getPrimaryKeys(null, "ejemplo", "departamentos");
```

```
StringBuilder claves= new StringBuilder();
```

```
String sep="";
```

```
while (pk.next()){  
    claves.append(sep);  
    claves.append(pk.getString(4));  
    sep=" + ";  
}
```

```
System.out.println("Clave primaria: "+ claves.toString());
```

```
ResultSet fk= dbmd.getExportedKeys(null, "ejemplo", "departamentos");
```

```
while (fk.next()){  
    String fk_name=fk.getString("FKCOLUMN_NAME");  
    String fk_tablename= fk.getString("FKTABLE_NAME");
```

```
    System.out.println("Tabla FK: "+fk_tablename+", Clave ajena: "+
```

```
    fk_name);
```

► 53

## Ejecución de sentencias de descripción de datos (VIII)

### ► **ResultSetMetaData**

- Se trata de una interfaz que permite obtener metadatos a partir de un *ResultSet*. Significa que podemos obtener información sobre los tipos y propiedades de las columnas, como el número de columnas devuelto. Su principales métodos son:
  - `getColumnCount()`. Devuelve el número de columnas devueltas por una consulta.
  - `getColumnName(indice de la columna)`. Devuelve el nombre de la columna.
  - `getColumnTypeName(indice)`. Devuelve el nombre del tipo de dato que contiene la columna específico del sistema de bases de datos.
  - `isNullable(indice)`. Devuelve 0 si la columna puede contener valores nulos.
  - `getColumnDisplaySize(indice)`. Devuelve el máximo ancho en caracteres de la columna.

►

► 54

## Ejecución de sentencias de descripción de datos (IX)

```
Statement sentencia= conexion.createStatement();
ResultSet rs= sentencia.executeQuery("Select * from departamentos");
ResultSetMetaData rsmd= rs.getMetaData();
int nCol= rsmd.getColumnCount();
System.out.printf("%-20s%-15s%-8s%-5s\n", "NOMBRE", "TIPO", "NULA",
"ANCHO");

    for (int i=1; i<= nCol; i++){
        System.out.printf("%-20s%-15s%-8s%-5s\n",
            rsmd.getColumnName(i),
            rsmd.getColumnTypeName(i),
            rsmd.isNullable(i),
            rsmd.getColumnDisplaySize(i));
    }
```

► 55

## Ejecución de sentencias de modificación de datos (I)

- La interfaz **Statement** dispone de métodos que permiten ejecutar sentencias SQL.
- Al tratarse de una interfaz, no puede crear objetos directamente; estos se crean con el método **createStatement()** de un objeto **Connection** válido.  
**Statement sentencia = conexión.createStatement()**
- Al crear un objeto *Statement* se crea un espacio de trabajo para crear consultas SQL, ejecutarlas y obtener sus resultados.

► 56

## Ejecución de sentencias de modificación de datos (II)

- ▶ Los métodos disponibles mediante *Statement* son:
  - ▶ `executeQuery(String)`. Para sentencias SQL que recuperan datos (SELECT) de un único objeto `ResultSet`
  - ▶ `executeUpdate(String)`. Se utiliza con sentencias de manipulación de datos (DML): INSERT, UPDATE, DELETE y de definición de datos (DDL): CREATE, DROP, ALTER.
  - ▶ `execute(String)`. Se utiliza para sentencias que devuelven más de un `ResultSet`, como por ejemplo la ejecución de procedimientos almacenados.
- ▶ Mediante los objetos `ResultSet` se puede acceder a los valores de las columnas, por nombre o por posición y obtener información sobre ellas: el número o su tipo.

▶ 57

## Ejecución de sentencias de modificación de datos (III)

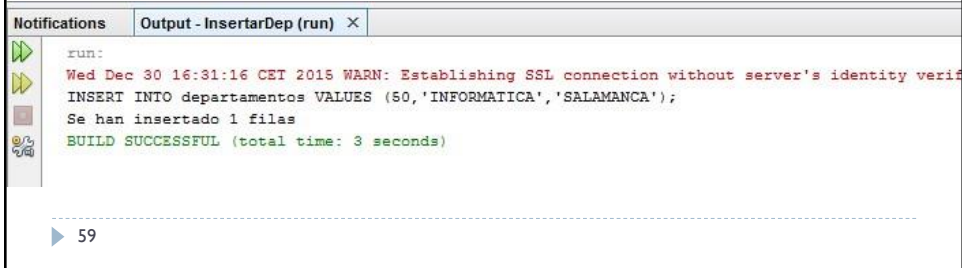
Ejemplo de aplicación que inserta un departamento en la base de datos ejemplo de MySQL, tomando los datos de la línea de órdenes de la aplicación.

```
public class InsertarDep {  
    public static void main(String[] args) {  
        try {  
            Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ejemplo",  
"ejemplo","ejemplo");  
            String dep= args[0];  
            String dnombre= args[1];  
            String loc= args[2];  
            String sql= "INSERT INTO departamentos VALUES (" +dep+", "+dnombre+", "+  
loc+",)";  
            System.out.println(sql);  
            Statement sentencia=conexion.createStatement();
```

▶ 58

## Ejecución de sentencias de modificación de datos (IV)

```
int filas= sentencia.executeUpdate(sql);
    System.out.printf("Se han insertado %d filas\n", filas);
    sentencia.close();
    conexion.close();
}
catch ( ClassNotFoundException | SQLException cn ) {cn.printStackTrace();}
}
```



## Ejecución de sentencias de modificación de datos (V)

### Ejecución de scripts

- ▶ Algunas bases de datos permiten la ejecución de varias sentencias de definición de datos DDL, o de manipulación de datos DML, en una misma línea. Este sería el procedimiento para poder ejecutar un script, convirtiendo sus líneas en una cadena de caracteres.
- ▶ El caso habitual podría ser un script que creara tablas e insertara elementos en las tablas.
- ▶ Para que sea posible la ejecución de varias sentencias seguidas es necesario añadir a la conexión la propiedad **allowMultiQueries= true**
- ▶ MySQL sería una de las bases de datos que lo permiten.

60

## Ejecución de sentencias de modificación de datos (VI)

### En MySQL sería:

```
Connection connmysql = (Connection) DriverManager
    .getConnection("jdbc:mysql://localhost/ejemplo?allowMul
tiQueries=true", "ejemplo", "ejemplo");
Statement sents = connmysql.createStatement();
int res = sents.executeUpdate(consulta);
```

- ▶ Siendo consulta un String al que se habría volcado un script.

▶ 61

## Ejecución de sentencias de modificación de datos (VII)

### Sentencias preparadas

- ▶ La interfaz **PreparedStatement** nos va a permitir construir una cadena de caracteres SQL con *placeholder* (marcadores de posición) que representa los datos que serán asignados más tarde.
- ▶ El *placeholder* se representa con el símbolo **?**.
- ▶ Ej.: `INSERT INTO departamentos VALUES (50, 'INFORMATICA', 'SALAMANCA')`

```
String sql = "INSERTO INTO departamentos VALUES (?, ?, ?)";
```

- ▶ Cada *placeholder* tiene un índice, el 1 sería el primero de la cadena, el 2 el segundo y así sucesivamente.

▶ 62

## Ejecución de sentencias de modificación de datos (VIII)

### Sentencias preparadas

- ▶ Antes de ejecutar un **PreparedStatement** es necesario asignar los datos, para que cuando se ejecute, la base de datos asigne variables de unión con estos datos y ejecute la orden SQL.
- ▶ Los objetos **PreparedStatement** se pueden precompilar una sola vez y ejecutarse las que queramos asignando diferentes valores a los marcadores de posición. Por el contrario, en los objetos **Statement** la sentencia SQL se suministra en el momento de ejecutar la sentencia.

▶ 63

## Ejecución de sentencias de modificación de datos (IX)

### Sentencias preparadas

- ▶ Los métodos de **PreparedStatement** tienen los mismos nombres (`executeQuery()`, `executeUpdate()` y `execute()`) que en **Statement** pero no se necesita enviar la cadena de caracteres con la orden SQL en la llamada ya que lo hace el método `prepareStatement(String)`.

```
String sql="INSERT INTO departamentos
VALUES (?, ?, ?) ";
PreparedStatement sentencia =
    conexión.prepareStatement(sql);
sentencia.setInt(1, Integer.parseInt(dep));
sentencia.setString(2, dnombre);
sentencia.setString(3, loc);
int filas= sentencia.executeUpdate();
```

▶ 64



## Ejecución de sentencias de modificación de datos (X)

### Sentencias preparadas

- ▶ Se utilizan los métodos `setInt(índice, entero)`, `setString(índice, cadena)`, `setFloat(índice, float)`, `setBoolean(índice, boolean)`, `setByte(índice, byte)`, `setShort(índice, short)`, `setLong(índice, long)`, `setDouble(índice, double)`, `setDate(índice, Date)`, `setTime(índice, Time)` y `setNull(índice, int tipoSQL)` para asignar los valores a cada uno de los marcadores de posición.
  - ▶ También se puede utilizar esta interfaz con la orden SELECT. Ej.:
  - ▶ `String sql= "SELECT apellido, salario FROM empleados WHERE dept_no= ? AND oficio = ? ORDER BY 1";`
  - ▶ `PreparedStatement sentencia= conexion.prepareStatement(sql);`
  - ▶ `sentencia.setInt(1, Integer.parseInt(dep));`
  - ▶ `sentencia.setString(2, oficio);`
  - ▶ `ResultSet rs= sentencia.executeQuery();`
- ▶ .65

## Ejecución de procedimientos (I)

- ▶ Los procedimientos almacenados en la base de datos consisten en un conjunto de sentencias SQL y del lenguaje procedural que utiliza el SGBD agrupadas bajo un nombre de procedimiento para realizar tareas sobre la base de datos.
- ▶ Se definen una vez y pueden utilizarse cuando se necesiten.
- ▶ Pueden definirse con parámetros de entrada (**IN**), salida (**OUT**), entrada/salida (**IN/OUT**) o sin parámetros.
- ▶ Si devuelven un valor se denominan **funciones**.
- ▶ Dependiendo del SGBD, pueden presentar diferencias, por ejemplo en MySql sólo admiten parámetros IN.

## Ejecución de procedimientos (II)

- ▶ Ejemplo de procedimiento en MySql:

```
DELIMITER //
CREATE PROCEDURE subida_salario (d INT, subida INT)
BEGIN
    UPDATE empleados SET salario = salario + subida WHERE dept_no=d;
    COMMIT;
END;
//
DELIMITER ;
```

- ▶ Se invocaría:

```
CALL subida_salario(20, 100);
```

▶ 67

## Ejecución de procedimientos (III)

- ▶ Ejemplo de función en MySql:

```
DELIMITER //
CREATE FUNCTION factorial(x INT) RETURNS INT(11)
BEGIN
    DECLARE fact INT;
    SET fact = 1;
    WHILE x > 0 do
        SET fact = fact * x;
        SET x = x - 1;
    END WHILE;
    RETURN fact;
END; //
```

- ▶ Se ejecutaría de esta forma:

```
SELECT factorial(5)
```

▶ 68

## Ejecución de procedimientos (IV)

- ▶ La interfaz **CallableStatement** permite que se pueda llamar desde Java a los procedimientos almacenados. Para crear un objeto se llama al método `prepareCall(String)` del objeto *Connection*.
- ▶ Ejemplo. Declaración de la llamada al procedimiento `subida_salario` que tiene dos parámetros y se les da valor con el marcador de posición `?`.  

```
String sql = "{call subida_salario(?,?)}";  
CallableStatement llamada=conexion.prepareCall(sql);
```

▶ 69

## Ejecución de procedimientos (V)

- ▶ Hay cuatro formas de declarar llamadas a los procedimientos y funciones dependiendo de si hay, o no, parámetros y de la devolución de valores.
- ▶ `{call procedimiento}` para un procedimiento almacenado sin parámetros.
- ▶ `{?=call function}` para una función almacenada que devuelve un valor y no recibe parámetros. El valor se recibe a la izquierda del igual y es el primer parámetro.
- ▶ `{call procedimiento(?, ?, ...)}` para un procedimiento almacenado que recibe parámetros.
- ▶ `{?=call function(?, ?, ...)}` para una función almacenada que devuelve un valor (primer parámetro) y recibe varios parámetros.

## Ejecución de procedimientos (VI)

Ejemplo de aplicación que sube el sueldo a los empleados en la base de datos ejemplo de MySQL, haciendo uso de un procedimiento almacenado.

```
public class EjecProced {  
    public static void main(String[] args) {  
        try {  
            Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ejemplo", "ejemplo",  
"ejemplo");  
            int dep = 20;  
            int incremento = 111;  
            String sql = "{call subida_salario(?,?)}";  
            System.out.println(sql);  
            CallableStatement llamada=conexion.prepareCall(sql);  
            llamada.setInt(1, dep);  
            llamada.setInt(2, incremento);  
            llamada.execute();  
            llamada.close();  
            conexion.close();  
        } catch (ClassNotFoundException | SQLException cn) { cn.printStackTrace();  
    }  
}
```

71

## Informes con JasperReports (I)



- ▶ **JasperReports** es una herramienta escrita en Java, de código abierto y licencia GPL, para generar informes. Permite generar informes PDF, HTML, XLS, RTF, ODT, CSV, TXT y XML.
- ▶ Se descarga de <http://community.jaspersoft.com/download>.
- ▶ Hay una versión *Server* o simplemente una *Library*. Además existen herramientas gráficas para el diseño de plantillas *Jaspersoft Studio* y *iReport Designer* (ya sin soporte).
- ▶ Para obtener información:
- ▶ <http://community.jaspersoft.com/wiki/jasperreports-library-tutorial>



72



## Informes con JasperReports (II)

- ▶ Para generar un informe en un proyecto, lo primero es añadir las siguientes bibliotecas: *commons-beanutils-1.9.0.jar*, *commons-codec-1.5.jar*, *commons-collections-3.2.1.jar*, *commons-digester-2.1.jar*, *commons-logging-1.1.1.jar*, *itext-2.1.7.js4.jar*, *Jackson-annotations-2.1.4.jar*, *Jackson-core-2.1.4.jar*, *Jackson-databind-2.1.4.jar*, *jasperreports-6.2.0.jar*, *jasperreports-fonts-6.2.0.jar*, y *tolos.jar*.
- ▶ Después habrá que seguir los siguientes pasos:
  - ▶ Generar una plantilla, fichero .jrxml, en el que se configura cómo queremos el informe.
  - ▶ Compilar el fichero .jrxml para obtener un fichero .jasper
  - ▶ Rellenar los datos del informe. Esto generará un fichero .jrprint
  - ▶ Exportar el fichero .jrprint al formato deseado: pdf, html, xml, etc.

## Informes con JasperReports (III)

- ▶ Generar el fichero .jrxml
  - ▶ Se puede generar a mano o mediante herramientas gráficas como *iReport*, o mejor aún con *Jaspersoft Studio*.
  - ▶ Se especifican los parámetros del informe, la sentencia SQL, los datos a visualizar, la cabecera, el detalle y el pie de la página..
- ▶ Compilar la plantilla .jrxml

```
JasperReport report =  
JasperCompileManager.compileReport("./plantilla/plantilla.jrxml");
```
- ▶ Rellenar el informe con datos
- ▶ `JasperPrint MiInforme = JasperFillManager.fillReport(report, params, conexBD);`
- ▶ donde `conexBD` es la conexión con la base de datos. Los parámetros tienen que crearse y almacenarse en un `HashMap`. Ejemplo:

```
Map<String, Object> params = new HashMap<>();  
params.put("titulo", "RESUMEN DATOS DE DEPARTAMENTOS.");  
params.put("autor", "J@");  
params.put("fecha", java.time.LocalDate.now().toString());
```

## Informes con JasperReports (IV)

- ▶ Exportar el fichero JasperPrint, al formato que se desee.
  - ▶ Para visualizarlo con un visor en pantalla:  
`JasperViewer.viewReport(MilInforme);`
  - ▶ Si queremos cerrar el visor sin cerrar la aplicación (por ejemplo en una aplicación con ventanas) se añade *false*:  
`JasperViewer.viewReport(MilInforme, false);`
  - ▶ Para generarlo en html:  
`JasperExportManager.exportReportToHtmlFile(MilInforme, reportHTML);`
  - ▶ Para generarlo en pdf:  
`JasperExportManager.exportReportToPdfFile(MilInforme, reportPDF);`
  - ▶ Para generarlo en xml:  
`JasperExportManager.exportReportToXmlFile(MilInforme, reportXML,`
- ▶ `false); //false para indicar que no hay imágenes`

## Ejemplo proyecto con JasperReports (I)

```
public class EjempJasper {
    public static void main(String[] args) {
        String reportSource = "./plantilla/plantilla.jrxml";
        String reportHTML = "./informes/Informe.html";
        String reportPDF = "./informes/Informe.pdf";
        String reportXML = "./informes/Informe.xml";
        Map<String, Object> params = new HashMap<>();
        params.put("titulo", "RESUMEN DATOS DE DEPARTAMENTOS.");
        params.put("autor", "J@");
        params.put("fecha", java.time.LocalDate.now().toString());
        try {
            JasperReport jasperReport =
JasperCompileManager.compileReport(reportSource);
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost/ejemplo", "ejemplo", "ejemplo");
            JasperPrint MilInforme = JasperFillManager.fillReport(jasperReport, params, conn);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

▶ 76

## Ejemplo proyecto con JasperReports (II)

```
JasperViewer.viewReport(MiInforme);
JasperExportManager.exportReportToHtmlFile(MiInforme, reportHTML);
JasperExportManager.exportReportToPdfFile(MiInforme, reportPDF);
JasperExportManager.exportReportToXmlFile(MiInforme, reportXML, false);
System.out.println("ARCHIVOS CREADOS");
} catch (CommunicationsException c) {
    System.out.println(" Error de comunicación con la BD. No está arrancada.");
} catch (ClassNotFoundException e) {
    System.out.println(" Error driver. ");
} catch (SQLException e) {
    System.out.println(" Error al ejecutar sentencia SQL ");
} catch (JRException ex) {
    System.out.println(" Error Jasper.");
    ex.printStackTrace();
}
}
```

77

## Plantilla .JRXML

Sección	Descripción
title	Su contenido se imprime una vez al principio del informe
pageHeader	Se imprimirá en cada página
columnHeader	Se escribe la cabecera que se va a poner para el detalle. Es decir, los nombres de las columnas que se visualizarán en <i>detail</i>
detail	Es el cuerpo del documento, donde se coloca la información a desplegar en forma tabular obtenida de la select
columnFooter	Aquí se ponen los totales acumulados, y la información para cada una de las columnas de detalle
pageFooter	Es el pie de página que se repite al final de cada página. Por ejemplo el número de página
summary	Se utiliza para concluir el documento, se imprime una sola vez al final del informe

## Gestión de errores (I)

- ▶ Hasta ahora, cuando se producía un error la secuencia de llamadas al método que ha producido la excepción y la línea de código donde se producía el error se visualizaba con `printStackTrace()`.
- ▶ Cuando se produce un error con `SQLException` podemos acceder a cierta información usando los siguientes métodos.
  - ▶ `getMessage()`. Devuelve una cadena que describe el error.
  - ▶ `getSQLState()`. Es una cadena que contiene un estado definido por el estándar X/OPEN SQL.
  - ▶ `getErrorCode()`. Es un entero que proporciona el código de error del fabricante.

▶ 79

## Gestión de errores (II)

```
try
{ // código }
catch (ClassNotFoundException cn) {cn.printStackTrace();}
catch {SQLException e)
{
    System.out.println("HA OCURRIDO UNA EXCEPCIÓN:");
    System.out.println("Mensaje :"+e.getMessage());
    System.out.println("SQL estado:"+e.getSQLState());
    System.out.println("Cod error :"+e.getErrorCode());
}
```

▶ 80