

# Bases de datos relacionales

## Tema 2. Acceso a datos



# Objetivos

- Conocer las APIs que proporcionan conectores a bases de datos, en particular JDBC.
- Comprender el desfase objeto-relacional.
- Abrir una conexión a una base de datos JDBC mediante drivers.
- Escribir programas en Java utilizando JDBC para ejecutar sentencias SQL: de definición de datos y de modificación y consulta de datos.
- Utilizar sentencias preparadas para ejecutar sentencias de SQL de manera eficiente.
- Trabajar con transacciones.

# Sistemas de bases de datos relacionales

- Un sistema gestor de base de datos (SGBD) es un conjunto de programas que administra y gestiona la información contenida una base de datos.
- El usuario no ve detalles acerca del almacenamiento y los procedimientos para recuperar y actualizar la información.

# Sistemas de bases de datos relacionales

- Dos tipos básicos:
  - Sistemas embebidos a los programas (SQLite, H2)
  - Sistemas independientes que se ejecutan de forma separada.

# Modelo relacional

- Indica como se organizan los datos en un SGBD.

Atributos (columnas)

| IdTipoHabitacion | NumeroCama | TipoCama   | TipoSDB | Descripcion                            |
|------------------|------------|------------|---------|--|
| 1                | 1          | individual | D       | 1 cama individual con ducha            |
| 2                | 2          | individual | D       | 2 camas individuales con ducha         |
| 3                | 2          | individual | DW      | 2 individuales con ducha y WC separado |
| 4                | 1          | doble      | D       | 1 cama doble con ducha                 |
| 5                | 1          | doble      | DW      | 1 cama doble con ducha y WC separado   |
| 6                | 1          | doble      | BW      | 1 cama doble con baño y WC separado    |
| 7                | 1          | cama XL    | BW      | 1 doble grande con baño y WC separado  |

Tuplas (filas)

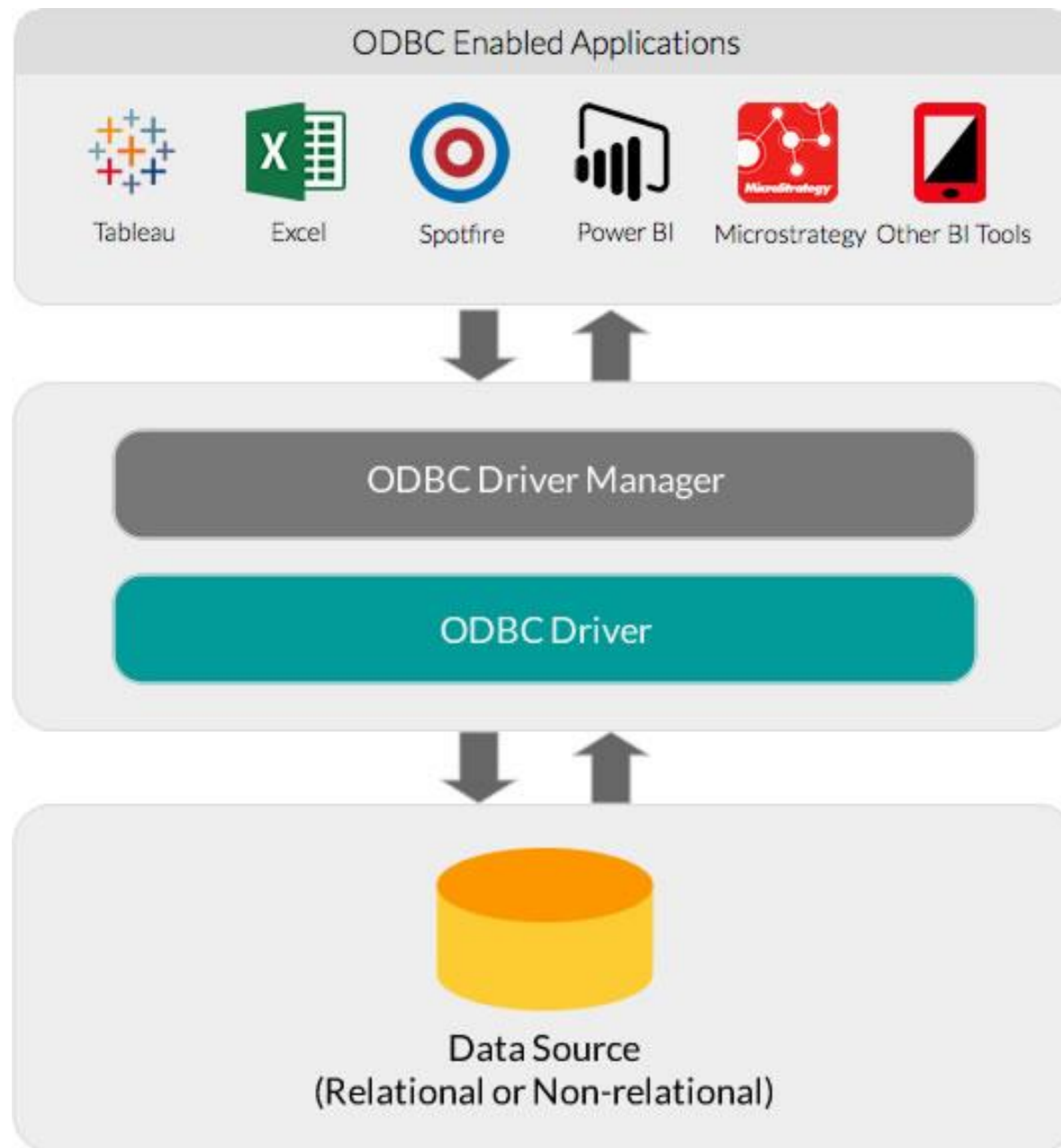
Tabla TiposHabitacion = Conjunto

# Conectores

- Un conector es una API que permite a los programas de aplicación trabajar con bases de datos.
- No son exclusivos de las bases de datos relacionales, también de objetos y No-SQL.

# Standard ODBC

- Open DataBase Connectivity (ODBC) es un estándar de acceso a las bases de datos.
- Permite acceder a cualquier dato desde cualquier aplicación, sin importar qué sistema de gestión de bases de datos almacene los datos
- Todos los SGBD lo implementan.



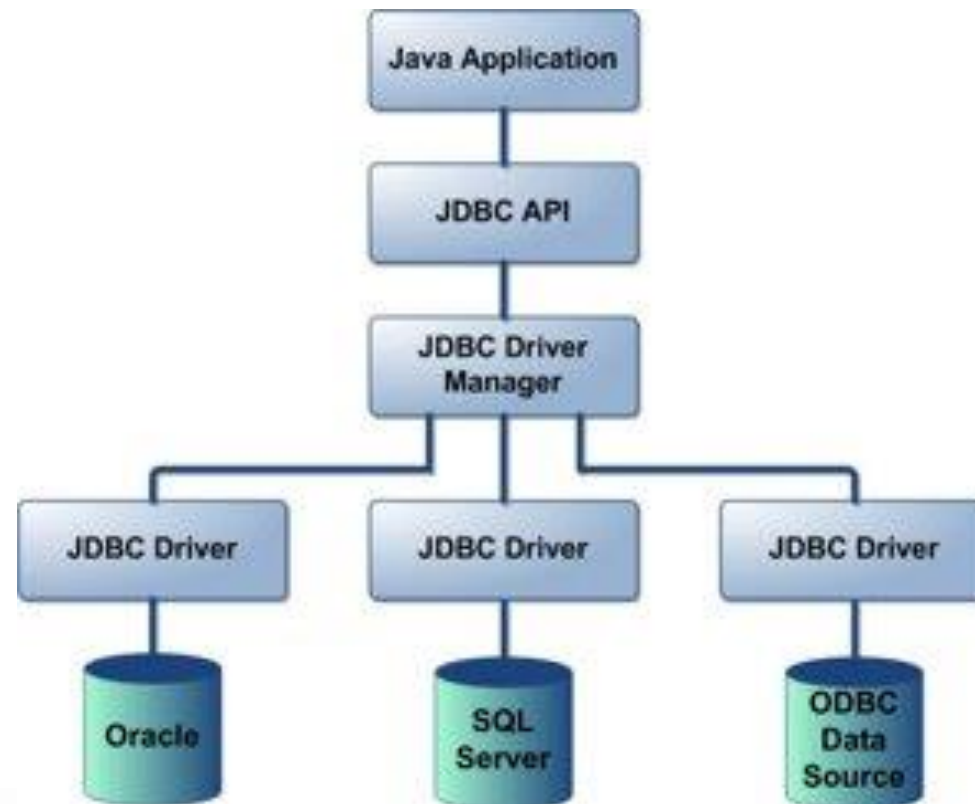


# JDBC (Java DataBase Connectivity)

<http://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>

- JDBC es la API que permite la conexión con las bases de datos utilizando el lenguaje de programación Java, independientemente del sistema operativo que se utilice y de la base de datos a que se quiera acceder.

# Tipos de drivers JDBC

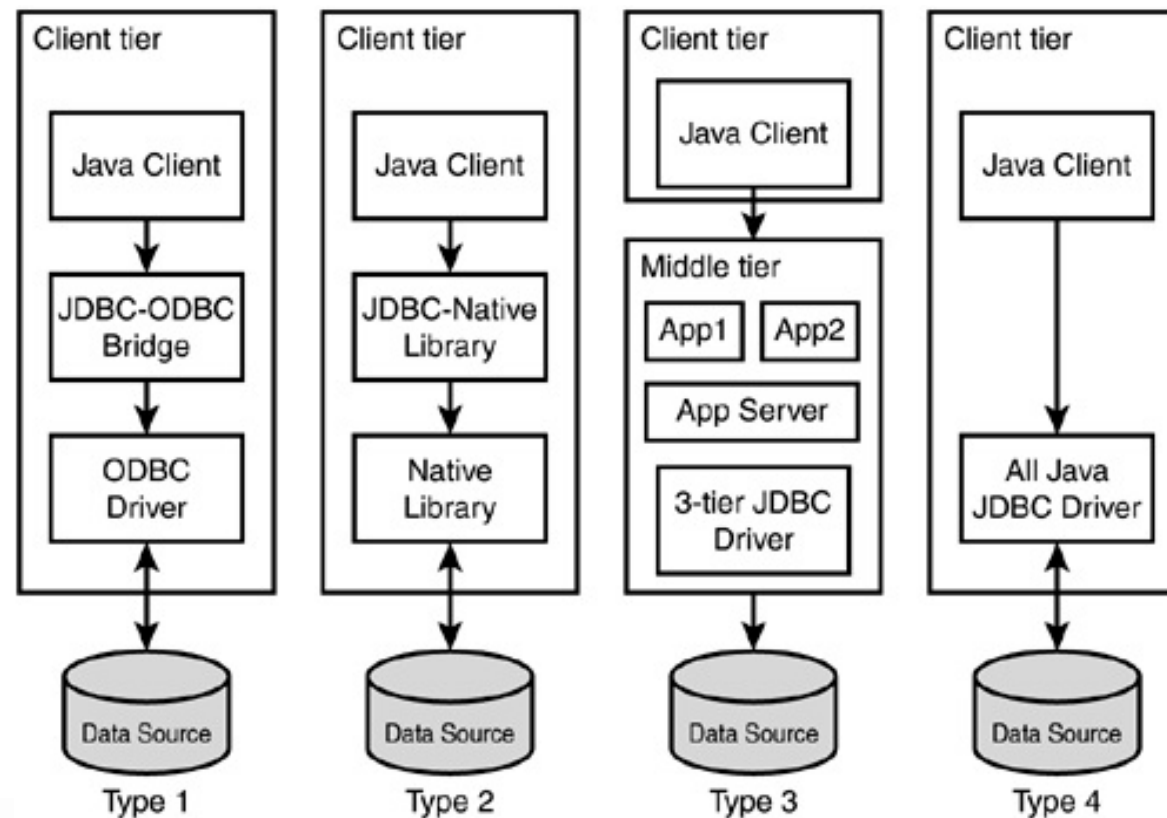


# Tipos de drivers JDBC

Se pueden distinguir cuatro tipos de drivers JDBC:

1. driver puente JDBC-ODBC.
2. driver API nativo / parte Java.
3. driver protocolo de red / todo Java.
4. driver protocolo nativo / todo Java.

# Tipos de drivers JDBC



# Características del driver puente ODBC

- Permite trabajar con multitud de drivers ODBC.
- Son útiles para las empresas que disponen de un driver ODBC instalado
- En ocasiones, es la única forma de conectar con determinadas bases de datos como, por ejemplo, Microsoft Access.
- Baja eficiencia y no soportado por applets

## Driver API nativo / Java

- Este tipo de drivers se encargan de traducir las solicitudes de API JDBC en llamadas específicas para cada base de datos, utilizando la interfaz de métodos nativos que proporciona Java.

# Protocolo de red / Java

Este tipo de drivers están implementados utilizando una arquitectura de tres capas:

- Capa cliente JDBC.
- Capa intermedia.
- Base de datos a la que se accede.

# Protocolo de red / Java

## Ventajas

- No hay que instalar ninguna biblioteca en el cliente
- Mejor rendimiento
- La mejor opción para aplicaciones distribuidas por internet

## Desventajas

- Capa intermedia específica para cada SGBD



## Protocolo nativo / Java

- Este tipo de driver realiza la conexión con las bases de datos de manera directa utilizando, para esto, el protocolo nativo del servidor.
- No necesitan utilizar ODBC o APIs nativas.

# Protocolo nativo / Java

## Ventajas

- No hay que instalar ninguna biblioteca en el cliente
- Mejor rendimiento
- La mejor opción para aplicaciones distribuidas por internet

## Desventajas

- Capa intermedia específica para cada SGBD

# Conexiones JDBC

El API JDBC permite realizar las conexiones de dos formas distintas:

- Utilizando la clase `java.sql.DriverManager` para aplicaciones en Java.
- Desde aplicaciones J2EE (Java 2 Enterprise Edition).

## Url JDBC

**jdbc:<subprotocolo>:<subnombre>**

- jdbc: indica el protocolo.
- Subprotocolo: indica el driver
- Subnombre: indica la base de datos

jdbc:mysql:empleados ; jdbc:h2:./empleados

# Operaciones básicas JDBC

## **Cargar Driver (no suele ser necesario)**

- `Class.forName("com.mysql.jdbc.Driver");`

## **Crear conexión**

- `c = DriverManager.getConnection(url,...);`

# Operaciones básicas JDBC

## Crear sentencia

- `s = c.createStatement();`

## Ejecutar sentencia

- `s.executeQuery(...)`
- `s.execute(...)`

# Operaciones básicas JDBC

## **Cerrar sentencia**

- `s.close();`

## **Cerrar conexión**

- `c.close();`

Instalación de un servidor de base de datos MySQL

Conexión desde un programa de Java.

Obtener el listado de las tablas disponibles



# Loved by Developers. Trusted by Ops

Bitnami makes it easy to get your favorite open source software up and running on any platform, including your laptop, Kubernetes and all the major clouds. In addition to popular community offerings, Bitnami, now part of VMware, provides IT organizations with an enterprise offering that is secure, compliant, continuously maintained and customizable to your organizational policies.

Community

Enterprise



[Home](#) > [Applications](#) > [Infrastructure](#) > WAMP



# WAMP



4.5 (26 ratings)

php.net

🕒 Updated

27 days ago

## DEPLOYMENT OFFERING



On my computer

|

Win / Mac / Linux



# Download

XAMPP is an easy to install Apache distribution containing MariaDB, PHP, and Perl. Just download and start the installer. It's that easy.



## XAMPP for Windows 7.3.31, 7.4.24 & 8.0.11

| Version             |                                  | Checksum            |                      |                                   | Size   |
|---------------------|----------------------------------|---------------------|----------------------|-----------------------------------|--------|
| 7.3.31 / PHP 7.3.31 | <a href="#">What's Included?</a> | <a href="#">md5</a> | <a href="#">sha1</a> | <a href="#">Download (64 bit)</a> | 158 Mb |
| 7.4.24 / PHP 7.4.24 | <a href="#">What's Included?</a> | <a href="#">md5</a> | <a href="#">sha1</a> | <a href="#">Download (64 bit)</a> | 160 Mb |
| 8.0.11 / PHP 8.0.11 | <a href="#">What's Included?</a> | <a href="#">md5</a> | <a href="#">sha1</a> | <a href="#">Download (64 bit)</a> | 161 Mb |

[Requirements](#) [Add-ons](#) [More Downloads »](#)

## Documentation/FAQs

There is no real manual or handbook for XAMPP. We wrote the documentation in the form of FAQs. Have a burning question that's not answered here? Try the [Forums](#) or [Stack Overflow](#).

- [Linux FAQs](#)
- [Windows FAQs](#)
- [OS X FAQs](#)
- [OS X XAMPP-VM FAQs](#)

## Add-ons





## Configuraciones generales

[Cambio de contraseña](#)

Server connection collation:

utf8mb4\_unicode\_ci

[Más configuraciones](#)

## Configuraciones de apariencia



Idioma - Language

Español - Spanish



Tema:

pmahomme

## Servidor de base de datos

- Servidor: Localhost via UNIX socket
- Tipo de servidor: MySQL
- Conexión del servidor: No se está utilizando SSL
- Versión del servidor: 8.0.21 - MySQL Community Server - GPL
- Versión del protocolo: 10
- Usuario: root@localhost
- Conjunto de caracteres del servidor: UTF-8 Unicode (utf8mb4)

## Servidor web

- Apache
- Versión del cliente de base de datos: libmysql - mysqlnd 7.4.10
- extensión PHP: mysqli curl mbstring
- Versión de PHP: 7.4.10

## phpMyAdmin

- Acerca de esta versión: 5.0.2 (actualizada)
- [Documentación](#)

# Información importante para la conexión

- Host
- Port
- User
- Password
- Parametros de conexión

# Driver para la conexión

## MySQL Community Downloads

Connector/J

**General Availability (GA) Releases**Archives*i*


### Connector/J 8.0.21

Select Operating System:

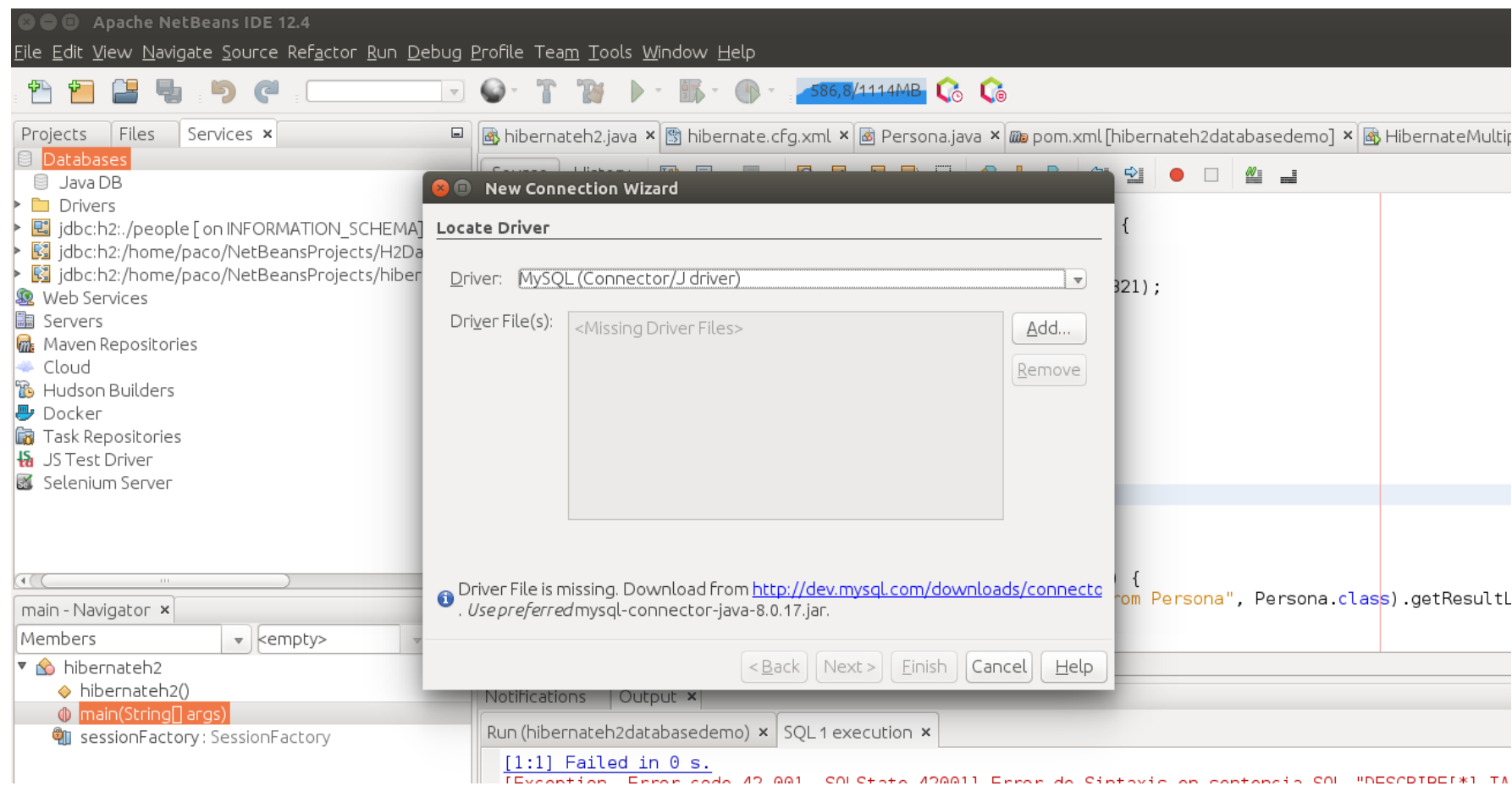
Platform Independent

[Looking for previous GA versions?](#)

|  |        |      |                          |
|--|--------|------|--------------------------|
| <b>Platform Independent (Architecture Independent),<br/>Compressed TAR Archive</b><br>(mysql-connector-java-8.0.21.tar.gz) | 8.0.21 | 3.8M | <a href="#">Download</a> |
| MD5: 971c74093c2ae2a2b5a8dbd65e9ef759   <a href="#">Signature</a>  |        |      |                          |
| <b>Platform Independent (Architecture Independent),<br/>ZIP Archive</b><br>(mysql-connector-java-8.0.21.zip)               | 8.0.21 | 4.5M | <a href="#">Download</a> |
| MD5: 5b6dcd4a1dd6769274822331d5cafe50   <a href="#">Signature</a>  |        |      |                          |

 We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.

# Conectando a la bas de datos desde NetBeans



# Uso con Maven

- **POM.xml**

```
<dependencies>
```

```
  <dependency>
```

```
    <groupId>mysql</groupId>
```

```
    <artifactId>mysql-connector-java</artifactId>
```

```
    <version>8.0.21</version>
```

```
  </dependency>
```

```
</dependencies>
```



# Interfaz Statement

- **ExecuteUpdate():** permite la ejecución de sentencias para la modificación de la base de datos: INSERT, UPDATE, DELETE, etc.
- **ExecuteQuery():** permite la ejecución de consultas: SELECT
- **Execute():** Permite cualquier tipo de consulta.

# Interfaz Statement

```
Statement statement = con.createStatement();  
  
statement.executeUpdate(sentenciaSQL);  
  
statement.close();
```

# Tipos de sentencias

- **DDL:** Lenguaje de definición de datos
- **DML:** Data Manipulation Language

# Tipos de sentencias

## Manipulación de datos

SELECT

Permite recuperar datos de la BD

INSERT

Permite añadir nuevas filas de datos de la BD

DELETE

Permite suprimir filas de la BD

UPDATE

Permite modificar datos existentes en la BD

# Tipos de sentencias


| Definición de datos |  |
|---------------------|--|
| CREATE TABLE        | Permite añadir una base de datos a la BD               |
| DROP TABLE          | Permite suprimir una tabla de la BD                    |
| ALTER TABLE         | Permite modificar la estructura de una tabla existente |
| CREATE VIEW         | Permite añadir una nueva vista a la BD                 |
| DROP VIEW           | Permite suprimir una lista de la BD                    |
| CREATE INDEX        | Permite construir un índice para una columna           |
| DROP INDEX          | Permite suprimir el índice para una columna            |

# Creación de tablas

```
String createTableSQL = "CREATE TABLE TAREAS ("
    + " ID INTEGER NOT NULL AUTO_INCREMENT, "
    + " TAREA CHAR(64) NOT NULL, "
    + " RESPONSABLE VARCHAR(32) NOT NULL, "
    + " PRIORIDAD CHAR(5), "
    + " PRIMARY KEY(ID));";

boolean execution = stmt.execute(createTableSQL);
System.out.println(execution);
```

# Creación de tablas

| v  |  | ejemplo <b>tareas</b> |
|--|---|-----------------------|
|   | ID : int  |                       |
|   | TAREA : char(64)  |                       |
|   | RESPONSABLE : varchar(32)   |                       |
|  | PRIORIDAD : char(5)   |                       |

# Inserción de datos

```
String insertData = "INSERT INTO TAREAS (TAREA,RESPONSABLE,PRIORIDAD) VALUES "  
    + "('Francisco','Preparar Clase','BAJA'),"  
    + "('Cristina','Hacer ejercicio', null),"  
    + "('Alberto','Repasar tema','ALTA'),"  
    + "('Alejandro','Estudiar', null);";  
  
int nFil = stmt.executeUpdate(insertData);  
System.out.println(nFil + " Filas insertadas.");
```



# Consultas

```
String obtenerTareasSQL = "SELECT * FROM TAREAS";  
ResultSet rs = stmt.executeQuery(obtenerTareasSQL);  
  
while (rs.next()) {  
    System.out.println "[" + rs.getInt("ID") + "]";  
    System.out.println "DNI: " + rs.getString("TAREA");  
    System.out.println "Apellidos: " + rs.getString("RESPONSABLE");  
    System.out.println "CP: " + rs.getString("PRIORIDAD");  
}  
rs.close();
```

# Métodos de ResultSet

**.next()**

**.previous()**

**.first()**

**.last()**

**.absolute(int pos)**

**.getXXX(string)**

**.getXXX(int)**

**.close()**

# Sentencias preparadas

Por seguridad y rendimiento, si vamos a usar sentencias SQL que requieran de algun parámetro, debemos usar sentencias preparadas.

La sentencia es "compilada" por el servidor y luego ejecutada con los diferentes parámetros.

# Sentencias preparadas

```
String query = "UPDATE TAREAS SET PRIORIDAD = ? WHERE ID = ?";  
PreparedStatement preparedStmt = this.connection.prepareStatement(query);  
preparedStmt.setInt(2,3);  
preparedStmt.setString(1,"baja");  
preparedStmt.executeUpdate();
```



## Sentencias preparadas

Siempre que tengamos algun parámetro en las consultas sql, sentencias preparadas **debemos usar.**

# Claves autogeneradas

```
CREATE TABLE `tareas` (  
  `ID` int NOT NULL,  
  `TAREA` char(64) COLLATE utf8mb4_general_ci NOT NULL,  
  `RESPONSABLE` varchar(32) COLLATE utf8mb4_general_ci NOT NULL,  
  `PRIORIDAD` char(5) COLLATE utf8mb4_general_ci DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `tareas`  
  MODIFY `ID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=14;  
COMMIT;  
  
ALTER TABLE `tareas`  
  ADD PRIMARY KEY (`ID`);
```

# Claves autogeneradas

## Métodos de Statement

- **Int ExecuteUpdate(String, int autoGeneratedKeys)**
  - Statement.RETURN\_GENERATED\_KEYS
  - Statement.NO\_GENERATED\_KEYS
- **ResultSet getGeneratedKeys()**

# Claves autogeneradas

```
sInsert.setString(1, "Hacer una ventana con Swing");  
sInsert.setString(2, "David");  
sInsert.setString(3, "Baja");  
  
sInsert.executeUpdate();  
  
ResultSet rs = sInsert.getGeneratedKeys();  
rs.next();  
int num_id = rs.getInt(1);
```



# Multiples tablas

## Consultas complejas con JOIN

```
SELECT * FROM EVALUACIONES  
JOIN ALUMNOS ON EVALUACIONES.alumno_id=ALUMNOS.id  
JOIN EJERCICIOS ON EJERCICIOS.id=EVALUACIONES.tarea_id  
WHERE EVALUACIONES.puntuacion IS NOT NULL
```

# Transacciones

- Una transacción incluye varias instrucciones que se ejecutan de forma secuencial.
- Estas se realizarán al completo de manera exitosa, pero si hay algun fallo, todas deberán ser descartadas.
- MySQL las tiene deshabilitadas por defecto.

# Transacciones

## **START TRANSACTION**

Operacion1

Operacion2

...

## **ROLLBACK**

...

## **COMMIT**

# Transacciones

- **.setAutoCommit(true)**: cada instrucción se ejecuta de forma inmediata.
- **.setAutoCommit(false)**: agrupar varias instrucciones como parte de una sola transacción.
- **.commit()**: ejecuta todas las instrucciones desde el último commit.
- **.rollback()**: deshace los cambios realizados desde el último commit.

# Procesado por lotes

- `PreparedStatement stmt = c.prepareStatement();`
- `stmt.addBatch();` añade una sentencia preparada
- ....
- `stmt.executeBatch();` ejecuta todas las sentencias añadidas

# Base de datos H2

The logo for H2, featuring the letters 'H2' in a bold, white, sans-serif font. The 'H' and '2' are connected. The logo is set against a blue background that has a curved gradient, transitioning from a lighter blue on the left to a darker blue on the right.

H2

# Base de datos H2

```
<dependencies>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>
  </dependency>
</dependencies>
```

# Base de datos H2

```
try(Connection conn=DriverManager.getConnection("jdbc:h2:./ejemplo")){  
    Statement stmt = conn.createStatement();  
    stmt.executeUpdate(QUERY);  
}
```



# Modelo relacional

¿Cómo hacemos para almacenar  
objetos en un SGDB?

# Desfase Objeto-Relacional

Son las diferencias existentes entre la programación orientada a objetos (POO) y las bases de datos relacionales:

- Tipos de datos complejos frente a simples
- Objetos frente a tablas y tuplas
- Asociaciones frente a relaciones

# Desfase Objeto-Relacional

Debemos distinguir claramente entre:

- Modelo: Definición de clases
- Controlador: Operaciones con los objetos
- Vista: Interacción con el usuario

Aunque dependen entre sí, cada componente debe estar completamente separado.

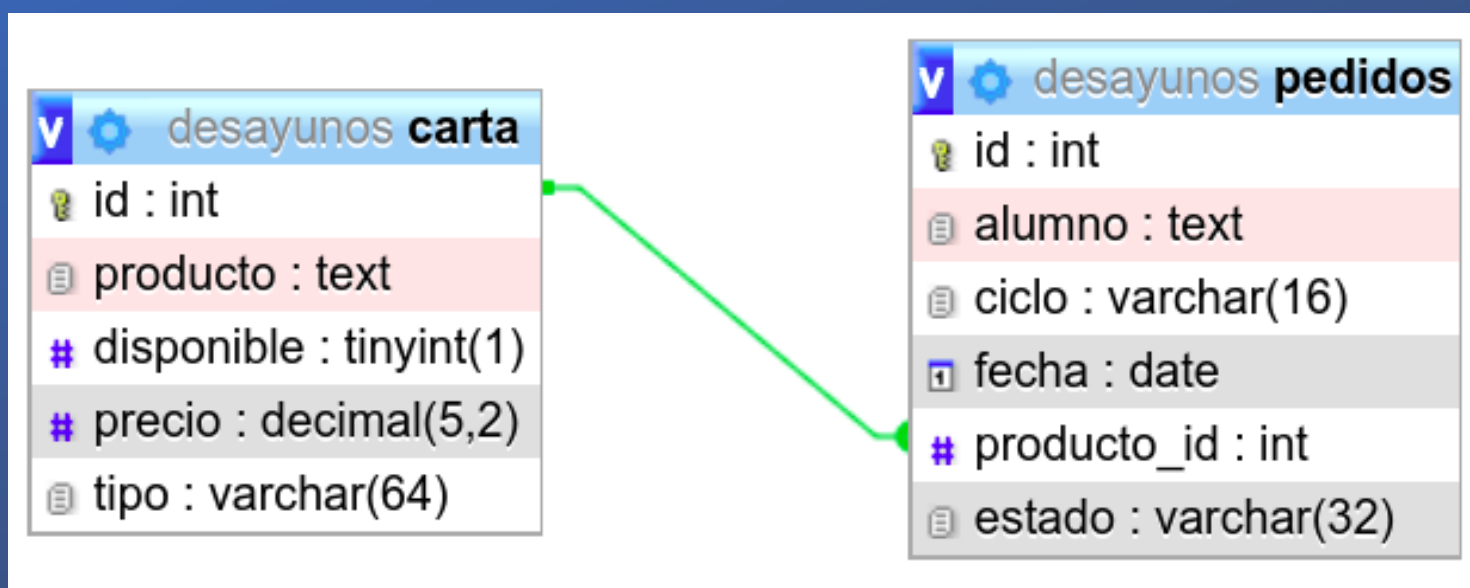
# Desfase Objeto-Relacional

Una primera aproximación es usar una tabla para almacenar los objetos de una clase.

Para casos muy sencillos puede ser suficiente pero para estructuras de datos mínimamente complejas necesitamos gestionar mejor la persistencia.

Crear una aplicación CRUD básica para gestionar los pedidos del desayuno de los alumnos de la clase.

# Modelo de datos a usar



# Interfaz de consola

```
=== Menu de opciones ===
```

```
[p] Mostrar los pedidos para hoy
```

```
[a] Añadir pedido
```

```
[b] Borrar pedido
```

```
[j] Recojer pedido
```

```
[l] Listar pedidos pendientes
```

```
[x] Mostrar el histórico
```

```
[q] Salir
```

```
p
```

```
=====>
```

```
Pedido{id=14, alumno=Juan, ciclo=2DAM, fecha=2020-10-22, producto=Producto{id=1,
```

```
Pedido{id=15, alumno=Juan, ciclo=2DAM, fecha=2020-10-22, producto=Producto{id=1,
```

```
Pedido{id=17, alumno=Juan, ciclo=2DAM, fecha=2020-10-22, producto=Producto{id=1,
```

# Mapeo clases POJO

```
package com.paco.desayunosmysql;

import java.util.Date;
import java.util.Objects;

public class Pedido {

    private int id;
    private String alumno;
    private String ciclo;
    private Date fecha;
    private Producto producto;
    private String estado;

    public Pedido() { ...3 lines }

    public Pedido(String alumno, String ciclo, Producto producto) { ...8 lines }

    public int getId() { ...3 lines }

    public void setId(int id) { ...3 lines }

    public String getAlumno() { ...3 lines }

    public void setAlumno(String alumno) { ...3 lines }
```