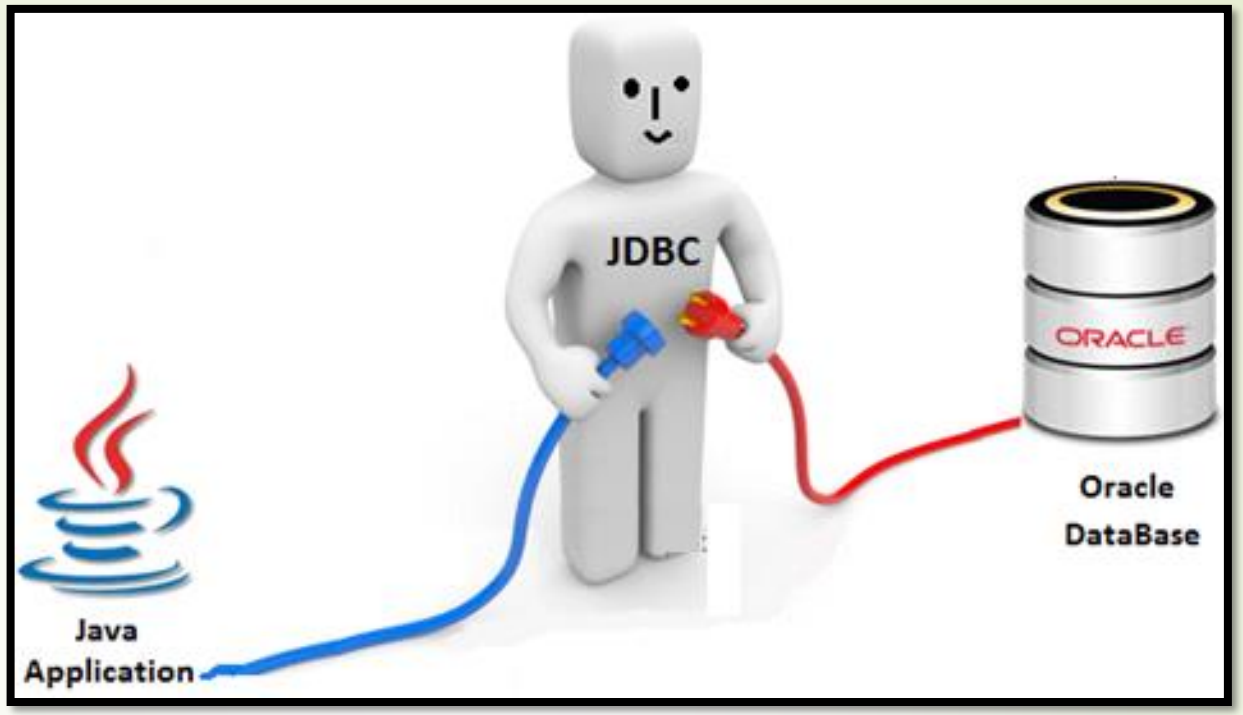

# Desarrollo de Aplicaciones Multiplataforma

**DOCENTE: Daniel López Lozano**

escuelaartegranada




# Tema 2.
# Desarrollo de Aplicaciones con Bases de Datos Relacionales

# Índice de contenidos

- Desfase Objeto-Relacional.
- ODBC y JDBC.
- Ejecución de código SQL desde Java usando MySQL.
- Manejo de fechas en Java con la clase Calendar.

- El desfase **objeto-relacional** consisten en la **diferencia de aspectos** que existen entre la programación orientada a objetos y las bases de datos relacionales.

- El modelo relacional usa **relaciones y conjuntos** con una base matemática.

- La programación orientada a objetos se basa en **clases, objetos** y las **asociaciones** entre ellos (herencia).

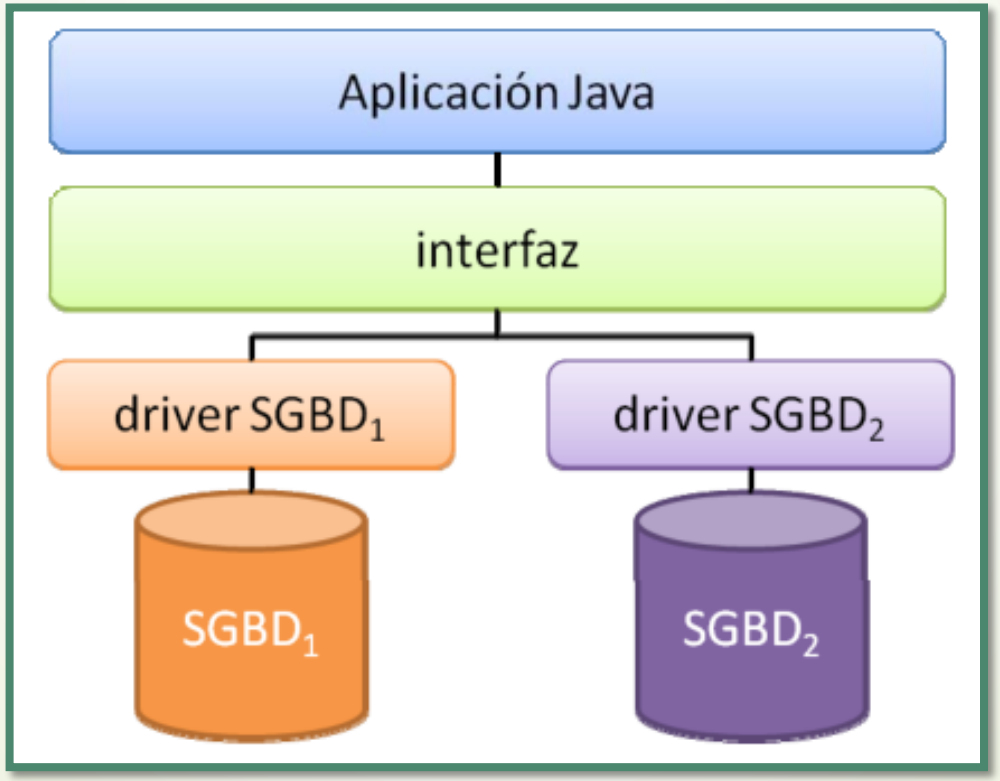
- La diferencia entre objetos y las tablas con filas (tuplas) implica desarrollar un **software adicional** que compenetre ambos esquemas.

- Unos de los puntos a tratar de este curso es ver **soluciones** para solventar el problema del desfase objeto-relacional.

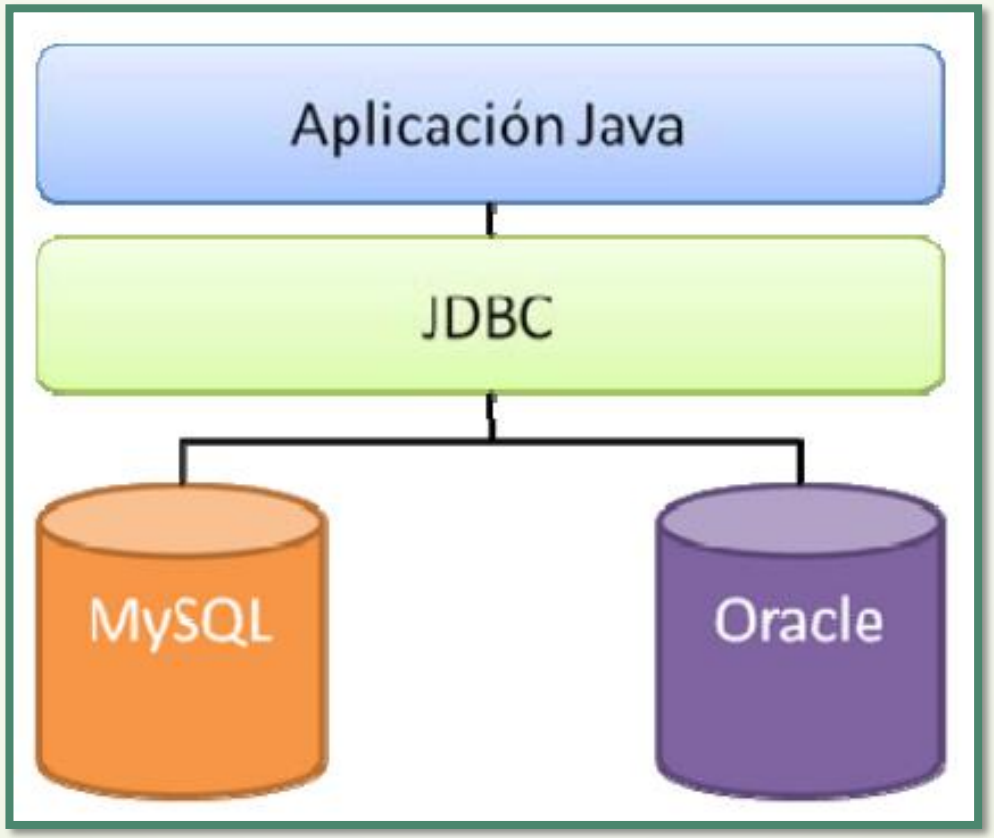
- En este tema veremos como usar **los conectores** que se comunican con el sistema gestor de bases de datos.

- El conector **envía la consulta SQL** a la base de datos.

- El objetivo del conector es **reducir la diversidad** y complejidad del uso de bases de datos colocando una interfaz entre la aplicación y la BD.



- Las normas más comunes que implementan dicha interfaz son **ODBC y JDBC**.



- **ODBC (Open Database Connectivity)** define una API que pueden usar las aplicaciones para abrir una conexión con una base de datos.

- Permitiendo realizar consultas, actualizaciones y obtener los resultados **en el propio lenguaje**.

- Se puede usar dicha API para conectar con cualquier **servidor de bases de datos compatible con ODBC**.

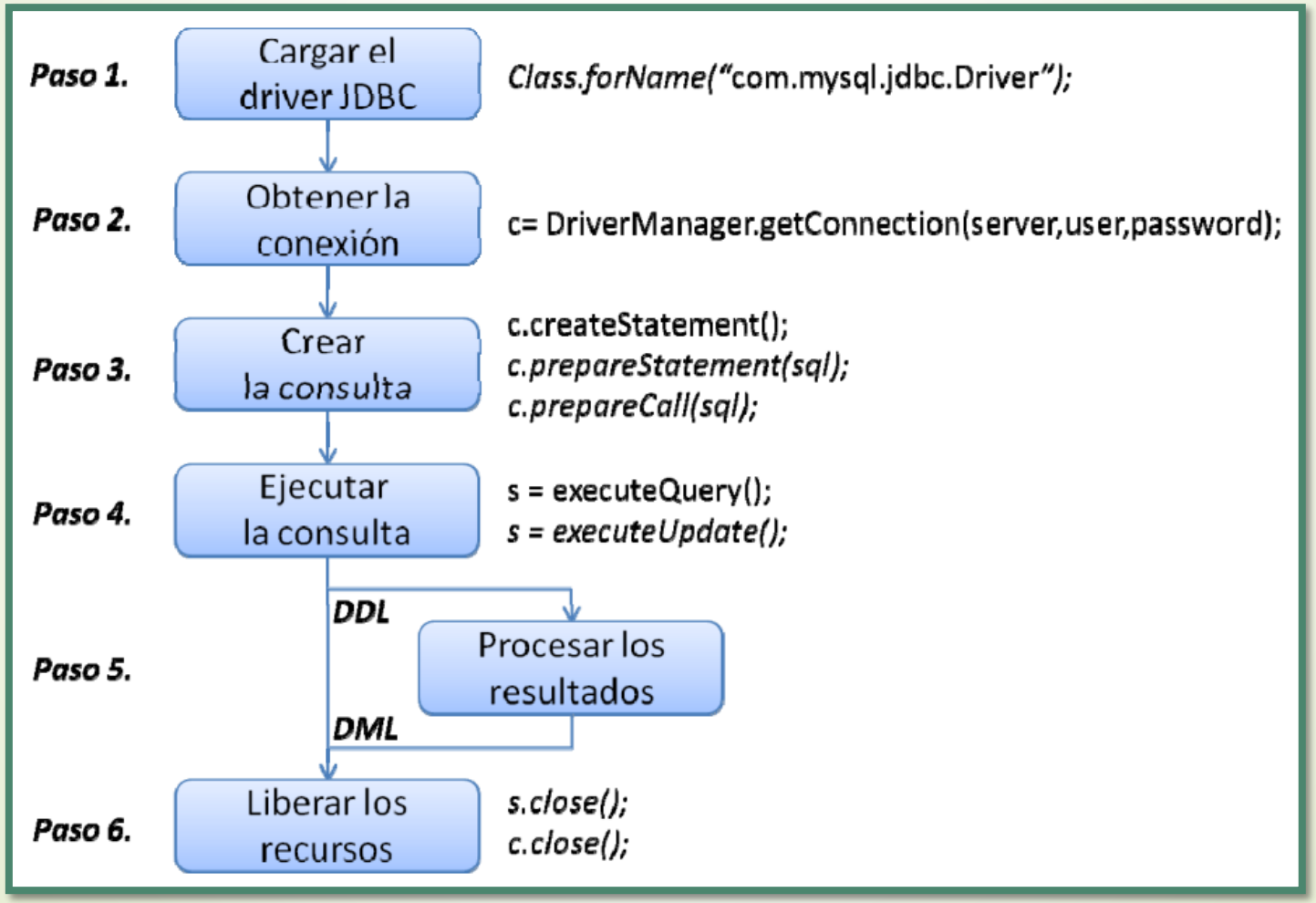
- ODBC fue desarrollado por **Microsoft**.

- **JDBC (Java Database Connectivity)** define una API similar a ODBC pero para el lenguaje Java.

- JDBC además de proveer un interfaz también define una **arquitectura estándar** para que los fabricantes puedan desarrollar la compatibilidad de conexión con Java.

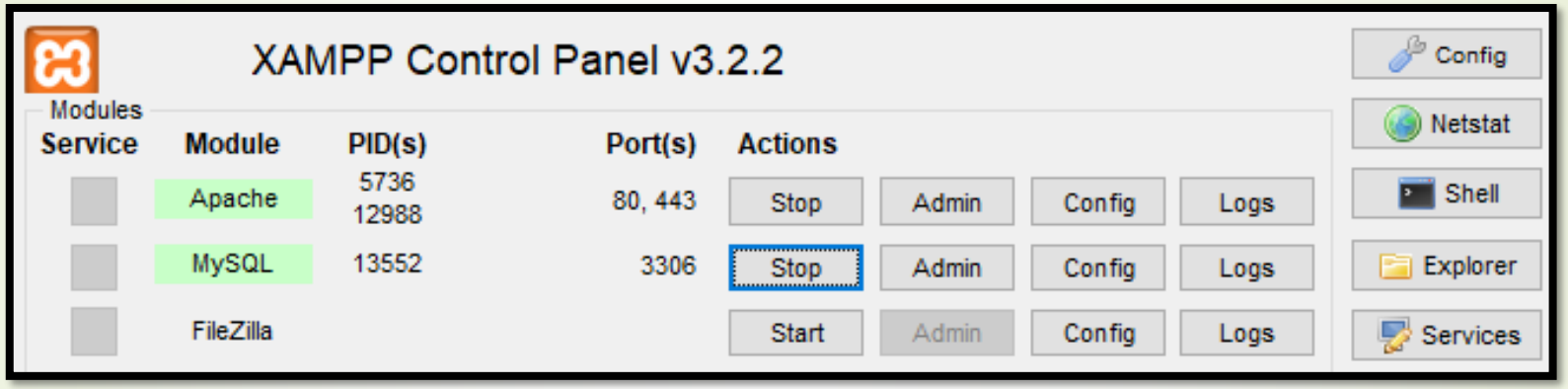
- Dentro del modelo existen alternativas para la compatibilidad donde el conector puede estar escrito **totalmente en Java**, usar un **conector ODBC** como puente, etc.

- Para acceder a bases de datos usando JDBC se debe seguir el siguiente esquema y uso de objetos.

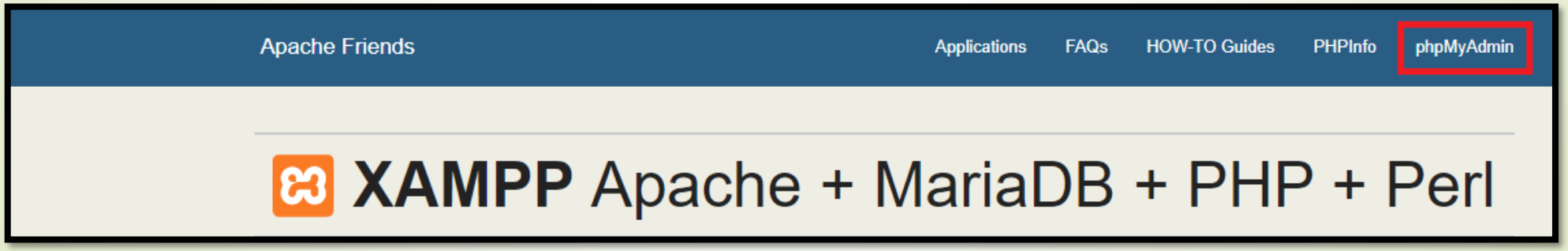


# Cargar código SQL en XAMPP

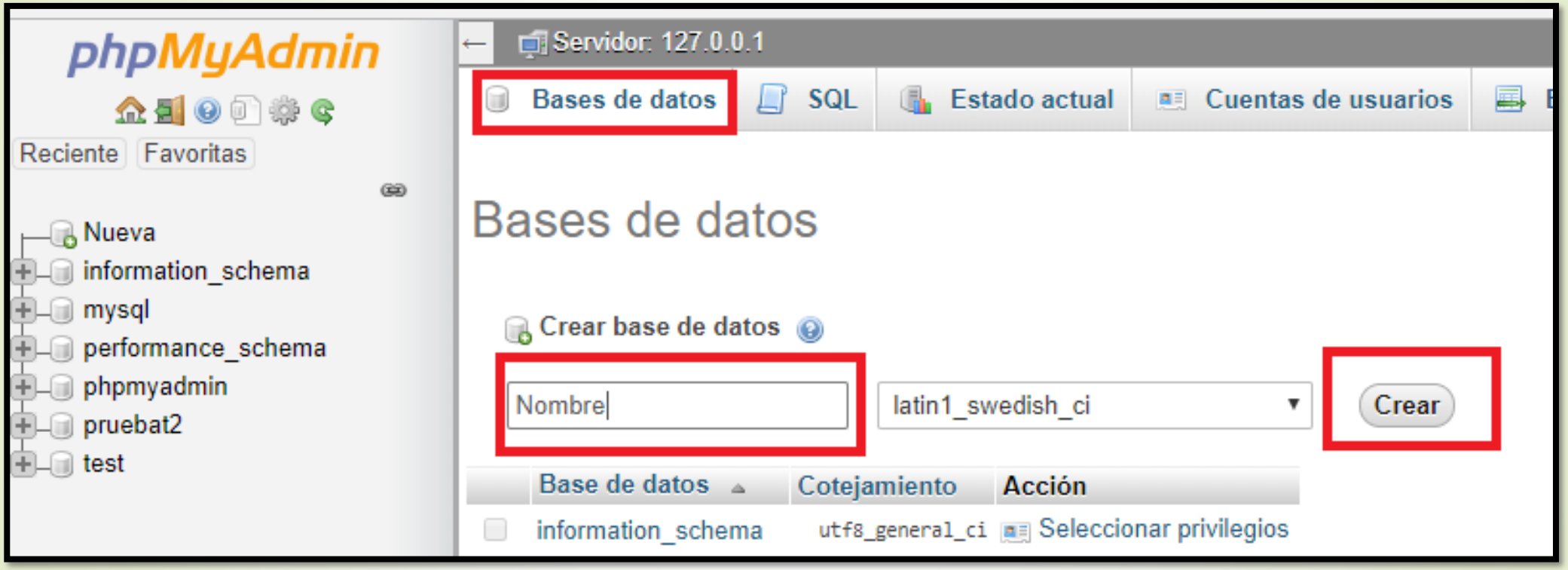
## Lanzar módulos con XAMPP Control Panel



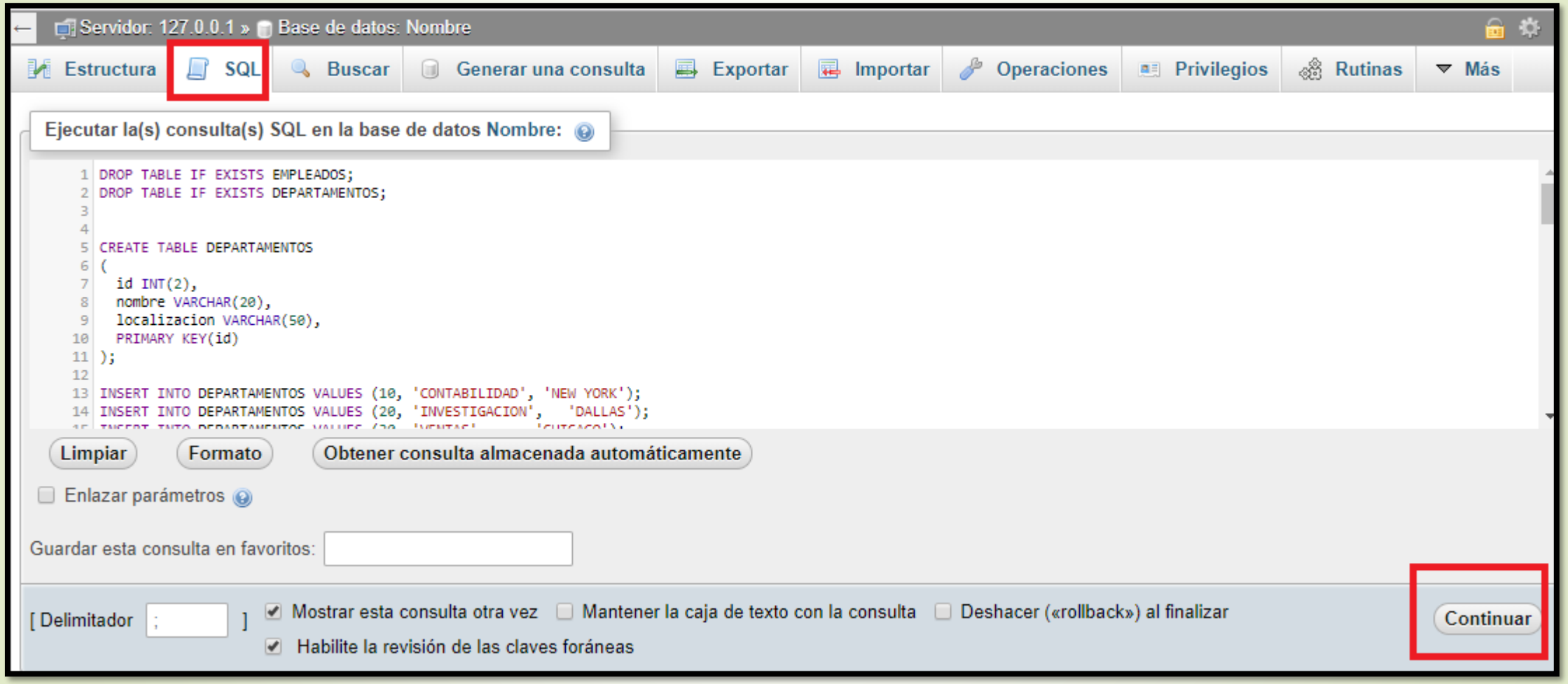

## En el navegador web meternos en localhost



# Crear la base de datos



# Introducir el código SQL



- A continuación presentamos un ejemplo sencillo para MySQL.

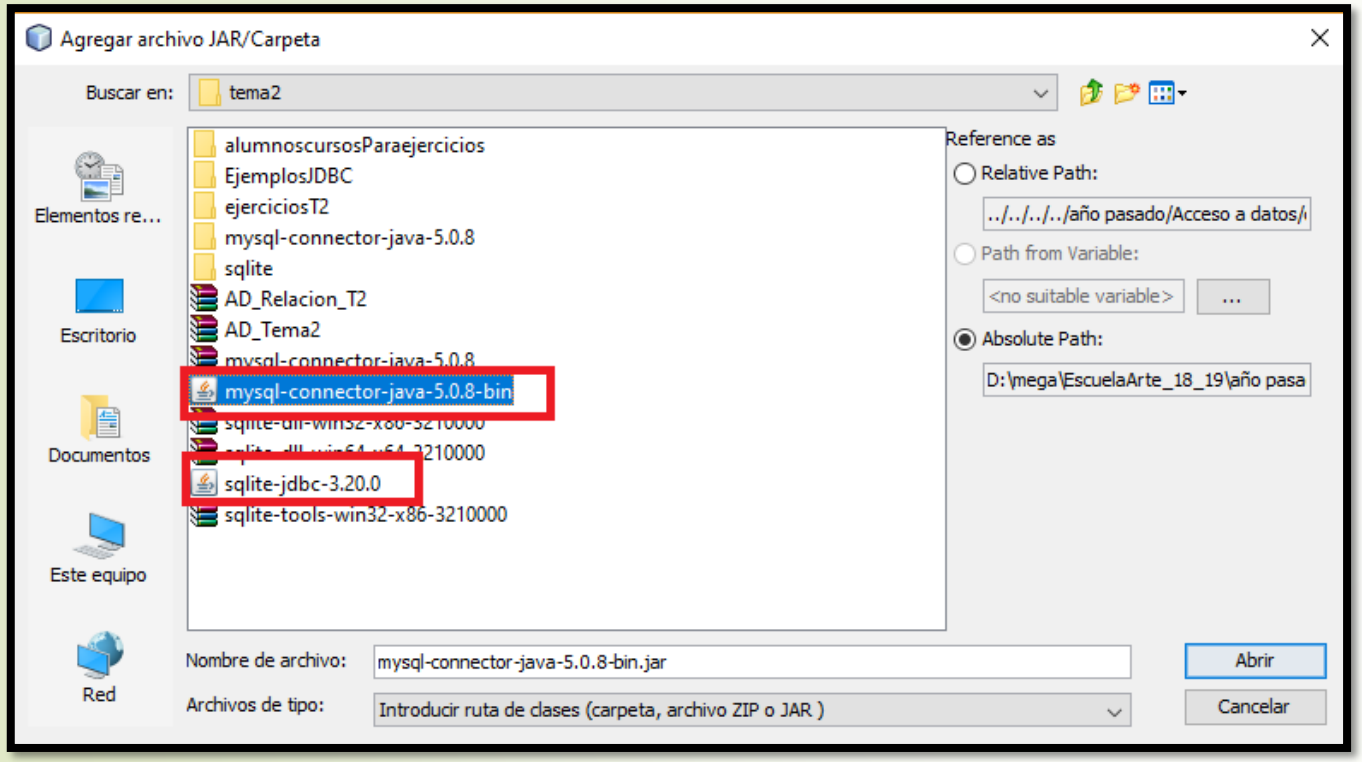
```
CREATE TABLE DEPARTAMENTOS
(
  id INT(2),
  nombre VARCHAR(20),
  localizacion VARCHAR(50),
  PRIMARY KEY(id)
);
```

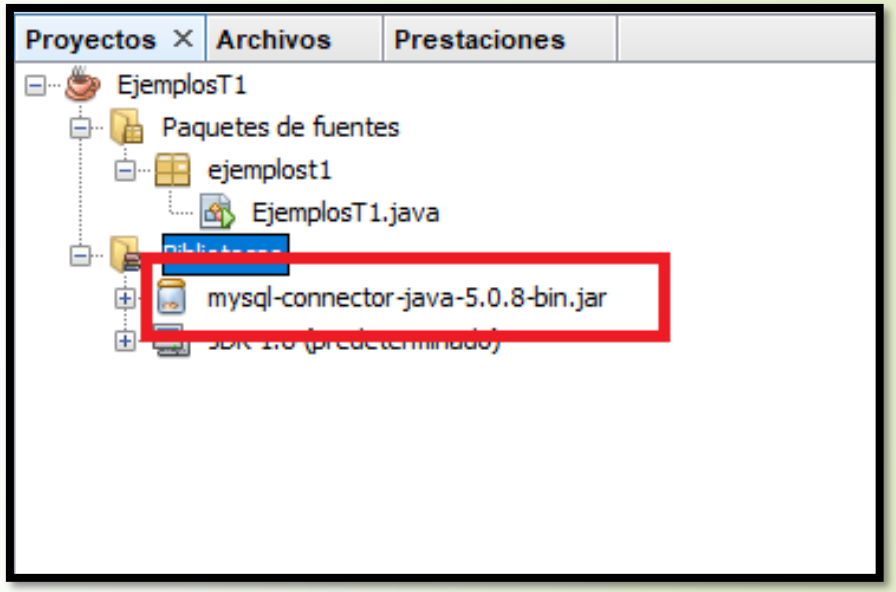
```
CREATE TABLE EMPLEADOS
(
  id INTEGER(4) AUTO_INCREMENT,
  apellido VARCHAR(20) UNIQUE,
  cargo VARCHAR(9),
  jefe INTEGER(4),
  fecha_alta DATE,
  salario DOUBLE,
  comision DOUBLE,
  departamento INTEGER(4),
  FOREIGN KEY (departamento) REFERENCES DEPARTAMENTOS(id),
  PRIMARY KEY(id)
);
```

```
ALTER TABLE EMPLEADOS ADD FOREIGN KEY(jefe) REFERENCES EMPLEADOS(id);
```

# Cargar el conector JDBC en Netbeans

## (como jar)




- Es probable que sea necesario instalar algún software puente con ODBC y las cadenas de conexión desde Java varíen.

- Una vez que tenemos la base de datos cargada y el proyecto preparado para acceso a datos, vamos a introducir el código de la aplicación.

- En primer lugar es necesario cargar el conector y el objeto Connection para realizar operaciones.

```
import java.sql.*;
...
// Cargar el driver
Class.forName("com.mysql.jdbc.Driver");
// Establecemos la conexion con la BD
Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/empresaBD",
                                                  "root", //usuario de la BD
                                                  ""); //contraseña
```

- Posteriormente preparamos la consulta y la ejecutamos

```
// Preparamos la consulta
Statement sentencia = conexion.createStatement();
String sql = "SELECT * FROM departamentos";
ResultSet resul = sentencia.executeQuery(sql);
```

- Eso produce un objeto de la clase ResultSet que contiene los datos pedidos. Dichos datos los recorremos mediante un bucle usando el objeto ResultSet.

```
while (resul.next()) {
        salida+=resul.getInt(1)+"\n"+
                resul.getString(2)+"\n"+
                resul.getString(3);
}
```

- El método next nos da la siguiente fila y getTipo nos devuelve una columna de la tabla.

- Podemos acceder a los campos de la fila poniendo los nombres directamente en lugar de las posiciones.

```
while (resul.next()) {
        salida+=resul.getInt("id")+"\n"+
                resul.getString("nombre")+"\n"+
                resul.getString("localizacion");
}
```

- Por ultimo mostramos los resultados y liberamos recursos.

```
System.out.println("TABLA DEPARTAMENTOS: \n"+salida);
resul.close(); // Cerrar ResultSet
sentencia.close(); // Cerrar Statement
conexion.close(); // Cerrar conexión
```

- Hay que tener en cuenta que dicho código tiene excepciones que hay que tratar.

```
try {
    //Aqui iria todo el codigo ordenado
}catch (ClassNotFoundException cn){
    System.out.println("Error al cargar el driver");
    cn.printStackTrace();
}catch (SQLException e) {
    System.out.println("Error al acceder a BD"+"\n"+e.getMessage());
    e.printStackTrace();
}
```

- Obviamente lo interesante de todo esto es crear consultas SQL parametrizadas.

```
//la variable loc contiene una entrada de datos de usuario
String selectStatement = "SELECT * FROM departamentos WHERE localizacion = "+ loc;
```

- También se pueden usar para esto consultas parametrizadas que se basan en usar una clase PreparedStatement en lugar de clase Statement.

```
String selectStatement = "SELECT * FROM departamentos WHERE localizacion =? ";
PreparedStatement sentencia = con.prepareStatement(selectStatement);
sentencia.setString(1, localizacion);
ResultSet resul = sentencia.executeQuery();
```

- Su principal beneficio es que ignoran los metacaracteres añadiendo seguridad frente a la inyección de código. Otro ejemplo con más parámetros.

```
String selectStatement = "SELECT * FROM empleados WHERE ocupacion =? AND fecha_alta=?
PreparedStatement sentencia = con.prepareStatement(selectStatement);
sentencia.setString(1, ocupacion);
sentencia.setString(2, fecha_alta);
ResultSet resul = sentencia.executeQuery();
```

- Algunos ejemplos de inyección de código para obtener acceso y/o información sin credenciales.
  - https://www.mclibre.org/consultar/php/lecciones/php-db-inyeccion-sql.html#inyeccion-1
  - https://www.securityartwork.es/2013/11/21/evasion-de-autenticacion-con-inyeccion-sql/

- Más métodos de la clase **ResultSet:**
  - ✓ previous(): Retrocede el cursor a la siguiente fila de ResultSet.
  - ✓ first(): Apunta a la primera fila del ResultSet.
  - ✓ last(): Apunta a la última fila del ResultSet.
  - ✓ getRow(): Devuelve el índice(int) de la fila actual.
  - ✓ getDouble(int) /getDouble(String): Devuelve una columna decimal
  - ✓ getBoolean(int) /getBoolean(String): Devuelve una columna de booleanos
  - ✓ getDate(int) /getDate(String): Devuelve una columna de fechas
  - ✓ getMetaData: Devuelve un objeto ResultSetMetaData con información extra
- Métodos de **ResultSetMetaData:**
  - ✓ getColumnCount(): Devuelve el número de columnas de la consulta.
  - ✓ getColumnName(int): Devuelve el nombre de una columna según la posición.
  - ✓ getColumnType(int): Devuelve el tipo de datos de una columna según la posición.

- Para operaciones de manipulación de datos y de definición se usa el método executeUpdate.

## Ejemplo de INSERT INTO

```
String dep = 50; // num. departamento
String dnombre = "MARKETING"; // nombre
String loc = "GRANADA"; // localidad

//construir orden INSERT
String sql = "INSERT INTO departamentos VALUES ('"+nombre+"', '"+localizacion+"')";
System.out.println(sql);
Statement sentencia = conexion.createStatement();
int filas=0;
//Ejecucion orden INSERT
filas = sentencia.executeUpdate(sql);
System.out.println("Filas afectadas: " + filas);
sentencia.close(); // Cerrar Statement
conexion.close(); // Cerrar conexión
```

# Distintos usos del INSERT INTO

```
//LAS FORMAS SON EQUIVALENTES

Si queremos usar el autoincrement

INSERT INTO USUARIOS (LOGIN,PASS,TIPO) VALUES ('PEPE','43242','NORMAL');
INSERT INTO USUARIOS VALUES (NULL,'PEPE','43242','NORMAL');

Si queremos establecer la clave primaria directamente

INSERT INTO USUARIOS VALUES (100,'PEPE','43242','NORMAL');
```

# Ejemplo de UPDATE

```
String dep = "10", subida ="100";
String sql = "UPDATE empleados SET salario = salario + "+subida+"
             WHERE dept_no = "+dep;
Statement sentencia = conexion.createStatement();
int filas = sentencia.executeUpdate(sql);
System.out.println("Empleados modificados: "+filas);
sentencia.close(); // Cerrar Statement
conexion.close(); // Cerrar conexión
```

# Ejemplo de DELETE

```
String localizacion ="GRANADA";
String sql = "DELETE FROM depatamentos
                WHERE localizacion = "+localizacion;

Statement sentencia = conexion.createStatement();
int filas = sentencia.executeUpdate(sql);
System.out.println("Departamentos borrados: "+filas);
sentencia.close(); // Cerrar Statement
conexion.close(); // Cerrar conexión
```

# Ejemplo de INSERT INTO con PreparedStatement

```
String sql = "INSERT INTO departamentos VALUES ('?', '?')";
PreparedStatement sentencia = con.prepareStatement(sql);
sentencia.setString(1, nombre);
sentencia.setString(2, localizacion);
```

# Ejemplo de UPDATE con PreparedStatement

```
String sql = "UPDATE empleados SET salario = ? WHERE dept_no = ?";
PreparedStatement sentencia = con.prepareStatement(sql);
sentencia.setString(1, salario);
sentencia.setString(2, departamento);
```

# Ejemplo de DELETE con PreparedStatement

```
String sql = "DELETE FROM depatamentos
              WHERE localizacion = ?";
PreparedStatement sentencia = con.prepareStatement(sql);
sentencia.setString(1, localizacion);
```

- También podemos controlar transacciones para evitar inconsistencia en la bases de datos.

```
conexion.setAutoCommit(false);
...
try {
    sentencia = conexion.createStatement();
    sentencia.executeUpdate("UPDATE cuentascorrientes SET saldo=saldo + 200" +
                            "WHERE numero_cuenta='11234343'");
    sentencia.executeUpdate("UPDATE cuentascorrientes SET saldo=saldo - 200" +
                            "WHERE numero_cuenta='34365211'");
    conexion.commit();
    ...
}catch SQLException ex) {
    System.out.println("ERROR:al hacer un Insert");
    conexion.rollback();
}
```

```
CREATE TABLE CUENTASCORRIENTES
(
  id INT(2),
  numero_cuenta VARCHAR(20),
  saldo DOUBLE,
  PRIMARY KEY(id)
);
```

- Todas estas operaciones realizadas con MySQL desde Java también de podrían hacer con Oracle u otro sistemas gestor.

- Enlace tutorial para quien esté interesado en hacerlo.

http://www.forosdelweb.com/f45/conexion-odbc-con-oracle-java-704603/

- En un principio la fechas las almacenaremos en formato ingles YYYY-MM-DD ya que son comparables como String para ordenar por fechas y nos ahorrara problemas en el futuro.
- Si no disponemos de ese formato mediante split debemos conseguirlo transformar

```
String fecha="31-12-2020";
String partes=fecha.split("-");
fecha=partes[2]"-"+partes[1]+"-"+partes[0];

//o tambien
String fecha="31/12/2020";
String partes=fecha.split("/");
fecha=partes[2]"-"+partes[1]+"-"+partes[0];
```

- Podemos obtener la fecha actual con Calendar mediante el método getInstance.
- La clase Calendar es especial ya que no necesitamos crear instancias de las mismas para usar fechas.
- Partiendo de Calendar podemos generar la fecha en formato String partiendo del método get y las constantes de la clase Calendar para obtener las partes que queramos

```
Calendar ahora=Calendar.getInstace();
String fecha=ahora.get(Calendar.YEAR)
                +"-"(ahora.get(Calendar.MONTH)+1)
                +"-"+ahora.get(Calendar.DATE);
```

- Aunque la forma más elegante es usar la clase **SimpleDateFormat** que permite a partir de un Calendar generar fechas en cualquier formato

```
Calendar ahora=Calendar.getInstance();
SimpleDateFormat formateador=new SimpleDateFormat("dd/MM/yyyy");
                                 SimpleDateFormat("yyyy-MM-dd");
String fecha=formateador.format(ahora.getTime());
```

| | |
|---|---|
| y | Year (e.g. 12 or 2012). Use either yy or yyyy. |
| M | Month in year. Number of M's determine length of format (e.g. MM, MMM or MMMMM) |
| d | Day in month. Number of d's determine length of format (e.g. d or dd) |
| h | Hour of day, 1-12 (AM / PM) (normally hh) |
| H | Hour of day, 0-23 (normally HH) |
| m | Minute in hour, 0-59 (normally mm) |
| s | Second in minute, 0-59 (normally ss) |
| S | Millisecond in second, 0-999 (normally SSS) |
| E | Day in week (e.g Monday, Tuesday etc.) |
| D | Day in year (1-366) |
| F | Day of week in month (e.g. 1st Thursday of December) |
| w | Week in year (1-53) |
| W | Week in month (0-5) |
| a | AM / PM marker |
| k | Hour in day (1-24, unlike HH's 0-23) |
| K | Hour in day, AM / PM (0-11) |

- Dada una fecha la podemos avanzar o retrasar mediante el método add y las constantes para referirnos si queremos avanzar por días, meses o años.

```
ahora.add(Calendar.MONTH,5);//Dentro de 5 meses
ahora.add(Calendar.DATE,-21);//Hace 21 dias
ahora.add(Calendar.YEAR,3);//Dentro de 3 años
ahora.add(Calendar.HOUR,-5);//Hace 5 horas
```

- Si queremos saber la diferencia entre 2 fechas tenemos que pasarla a milisegundos hacer la resta y pasarlo a la unidad deseada, por ejemplo.

```
long milisec = aDay.getTimeInMillis()-otherDay.getTimeInMillis();
long dias = milisec/1000/60/60/24;
```

- Para comparar dos fechas tenemos los métodos after, before y el ya clásico compareTo

```
fecha1.after(fecha2); //true si fecha1 es posterior a fecha2
fecha1.before(fecha2); //true si fecha1 es anterior a fecha2
fecha1.compareTo(fecha2);
//0 si son iguales
//negativo si fecha1 es anterior
//positivo si fecha1 es posterior
```

- Si recibimos cualquier otra fecha en un formato cualquiera y queremos aplicar métodos de Calendar para poder avanzar, comparar, etc una fecha:

```
String entrega="13/12/2021";
String[] partes=entrega.split("/");

Calendar entrega_cal=Calendar.getInstance();
entrega_cal.set(Calendar.DATE, Integer.parseInt(partes[0]));
entrega_cal.set(Calendar.MONTH, Integer.parseInt(partes[1])-1);
entrega_cal.set(Calendar.YEAR, Integer.parseInt(partes[2]));
```

# Bibliografía


- **Ramos Martín, Alicia y Ramos Martín, MªJesús:**
  "Acceso a Datos". Editorial Garceta. 2012

- **Córcoles Tendero, J.Ed. y Montero Simarro, Francisco:**
  "Acceso a Datos. CFGS". Editorial Ra-Ma. 2012

- **"Sistemas abiertos". Contenidos de la asignatura.**
  http:http://deim.urv.cat/~pedro.garcia/SOB/
  Última visita: Septiembre 2017.

- **Ejercicios Acceso a Datos 2ª CFGS DAM**
  https://github.com/andresmr/AccesoDatos
  Última visita: Septiembre 2017

- **Master en desarrollo de aplicaciones Android. Universidad Politécnica de Valencia**
  http://www.androidcurso.com/index.php/recursos/42-unidad-9-almacenamiento-de-datos/299-preferencias
  Última visita: Septiembre 2017