TEMA 2: MANEJO DE CONECTORES

¿QUÉ ES UNA BASE DE DATOS?

- Es una colección de datos relacionados entre si, que se almacenan para poder ser utilizados en un futuro.
- A diferencia de los ficheros, con las bases de datos podemos acceder a información repartida por el almacenamiento y agruparla para dar salida a un consulta muy concreta.
- Imaginemos, tenemos un fichero de Empleados y otro fichero de departamentos. Estos pese a poder ver que tienen relación (todo empleado tiene un departamento), no podemos unirlos y sacar un único resultado. Deberemos analizar cada fichero por separado y finalmente unir los datos para obtener la información, es un sistema poco eficiente.
    - Con las bases de datos esto no ocurre, ya que de por si la información está relacionada y permite acceder y recuperar a aquellos registros que deseamos (inclusive uniendo información entre ellos).
    - Si el ejemplo del fichero, lo extrapolamos a un uso intensivo con miles de usuarios el resultado puede ser catastrófico.
    -
- Se podrían catalogar como la evolución lógica de los ficheros.
    - Se pueden restringir datos de entrada
    - Se puede evitar duplicidad
    - Puede ser consultado por miles de usuarios
    - Tiene un motor eficiente interpretando las consultas para obtener resultados muy precisos

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos
    - Selección.
    - Proyección
    - Combinación
    - División
-

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

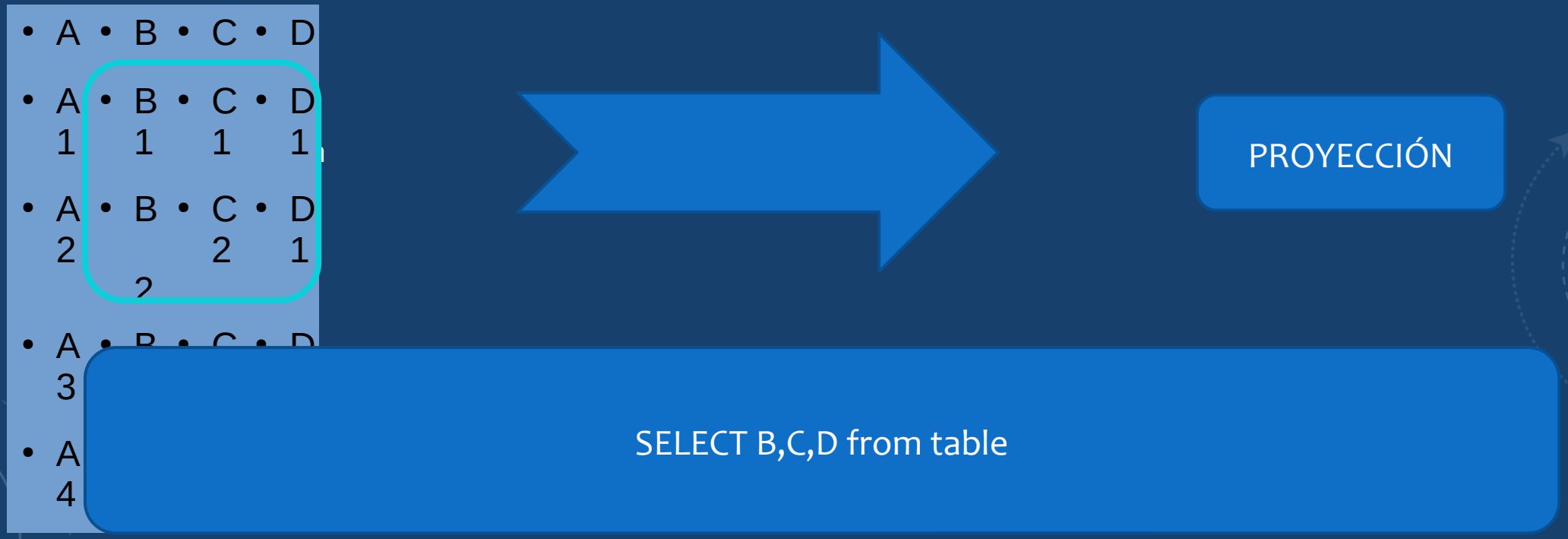
A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D1
A3	B3	C3	D2
A4	B4	C4	D2

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos



¿COMO FUNCIONAN LAS BD?

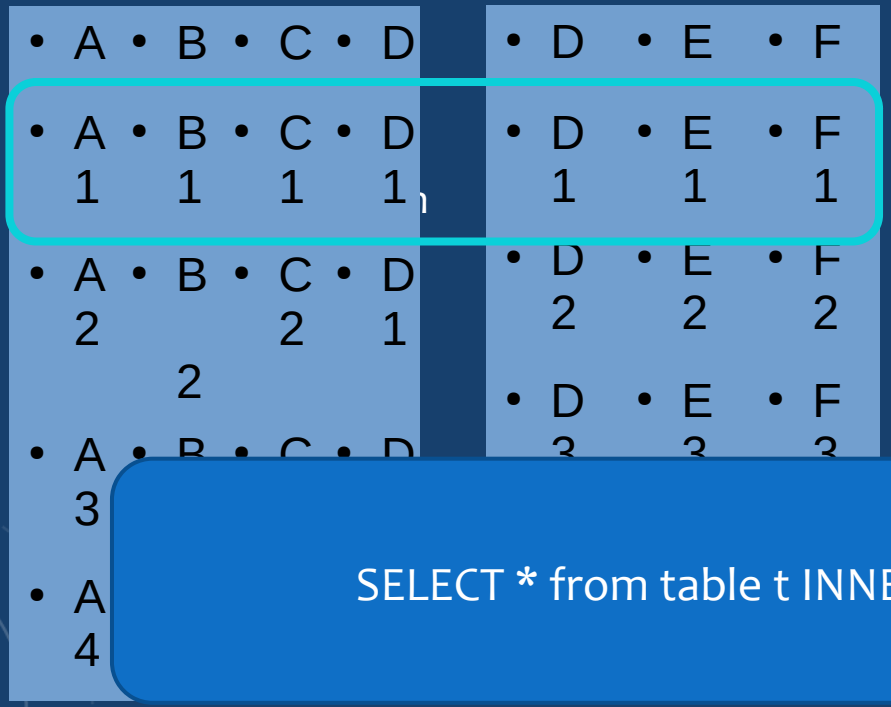
- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D1
A3	B3	C3	D2
A4	B4	C4	D2

D	E	F
D1	E1	F1
D2	E2	F2
D3	E3	F3
D4	E4	F4

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos



Unión

SELECT * from table t INNER JOIN table2 t2 ON t.D=t2.D where t.d="D1"

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B
A1	B1
A1	B2
A1	B3
A2	B1
A3	B1
A3	B2

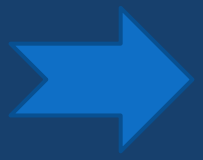
B
B1
B2
B3

¿COMO FUNCIONAN LAS BD?

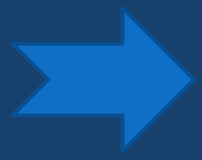
- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos

A	B
A1	B1
A1	B2
A1	B3
A2	B1
A3	B1
A3	B2

B
B1
B2
B3



Divisón



A
A1

¿COMO FUNCIONAN LAS BD?

- Las bases de datos se basan en algebra relacional. Donde disponemos de diferentes conjuntos de datos (tablas con filas) y podemos hacer operaciones con las mismas.
- Operaciones sobre bases de datos
    - Selección.
    - Proyección
    - Combinación
    - División
-

Gracias a este modelo matemático, si la BD está bien construída es muy eficiente y puede gestionar millones de registros

DESFASE OBJETO-RELACIONAL

- Las tecnologías, especialmente los lenguajes de programación han ido avanzando desde su origen, añadiendo más y más características. Dotando a los sistemas de más inteligencia y a su vez haciendo la programación más sencilla.
- Un claro ejemplo son los objetos donde ya no tenemos que trabajar con pseudo-estructuras de datos, si no que podemos crear estás estructuras y mantenerlas en memoria de forma estable haciendo muy sencilla la programación.
- Pero en todo esto existe un gran problema, las bases de datos relacionales no han avanzado tal y como han avanzado los lenguajes de programación, ya que en las bases de datos que tienen una estructura de tabla y filas el concepto objeto no es entendible.
- Todo esto se complica cuando en un objeto nos encontramos otro objeto en su interior
    -

DESFASE OBJETO-RELACIONAL

- Las tecnologías, especialmente los lenguajes de programación han ido avanzando desde su origen, añadiendo más y más características. Dotando a los sistemas de más inteligencia y a su vez haciendo la programación más sencilla.
- Un claro ejemplo son los objetos donde ya no tenemos que trabajar con pseudo-estructuras de datos, si no que podemos crear estás estructuras y mantenerlas en memoria de forma estable haciendo muy sencilla la programación.
- Pero en todo esto existe un gran problema, las bases de datos relacionales no han avanzado tal y como han avanzado los lenguajes de programación, ya que en las bases de datos que tienen una estructura de tabla y filas el concepto objeto no es entendible.
- Todo esto se complica cuando en un objeto nos e
  -

```
public class Persona {
   int id;
   String nombre;
   int telefono;
   ArrayList<String> direcciones;
}
```

DESFASE OBJETO-RELACIONAL

Las BD relacionales no admite que una columna tenga más de un valor, por lo que por ejemplo no podríamos tener varias direcciones tal y como plantea la clase Persona.
Para realizar esta operación debemos crear una segunda tabla donde unir estos datos

- Las tec ción han ido avanzando desde su origen, aña emas de más inteligencia y a su vez hac
- Un cla e trabajar con pseudo-estructuras de dat tenerlas en memoria de forma estable haciendo muy sencilla la programación.
- Pero en todo esto existe un gran problema, las bases de datos relacionales no han avanzado tal y como han avanzado los lenguajes de programación, ya que en las bases de datos que tienen una estructura de tabla y filas el concepto objeto no es entendible.
- Todo esto se complica cuando en un objeto nos e

```
public class Persona {
   int id;
   String nombre;
   int telefono;
   ArrayList<String> direcciones;
}
```

OBJETO VS TABLA

```
public class Persona {
   int id;
   String nombre;
   int telefono;
   ArrayList<String> direcciones;

   public Persona(int id, String nombre, int
telefono, ArrayList<String> direcciones) {
      this.id = id;
      this.nombre = nombre;
      this.telefono = telefono;
      this.direcciones = direcciones;
   }
}
```

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

OBJETO VS TABLA

```
public class Persona {
   int id;
   String nombre;
   int telefono;
   ArrayList<String> direcciones;

   public Persona(int id, String nombre, int
telefono, ArrayList<String> direcciones) {
      this.id = id;
      this.nombre = nombre;
      this.telefono = telefono;
      this.direcciones = direcciones;
   }
}
```

¡De forma natural esto no puede cohesionar!

• Nombre	• telefono
• Alvaro	• 712123123
• Tania	• 678987987

• id_persona	• direccion
• 1	• C/ Falsa 123
• 1	• C/Verdadera 123
• 2	• C/Test 123

UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.

```
public class Persona {
   int id;
   String nombre;
   int telefono;
   ArrayList<String> direcciones;
}
```

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.

Unimos el ID de la clase con el ID de la tabla

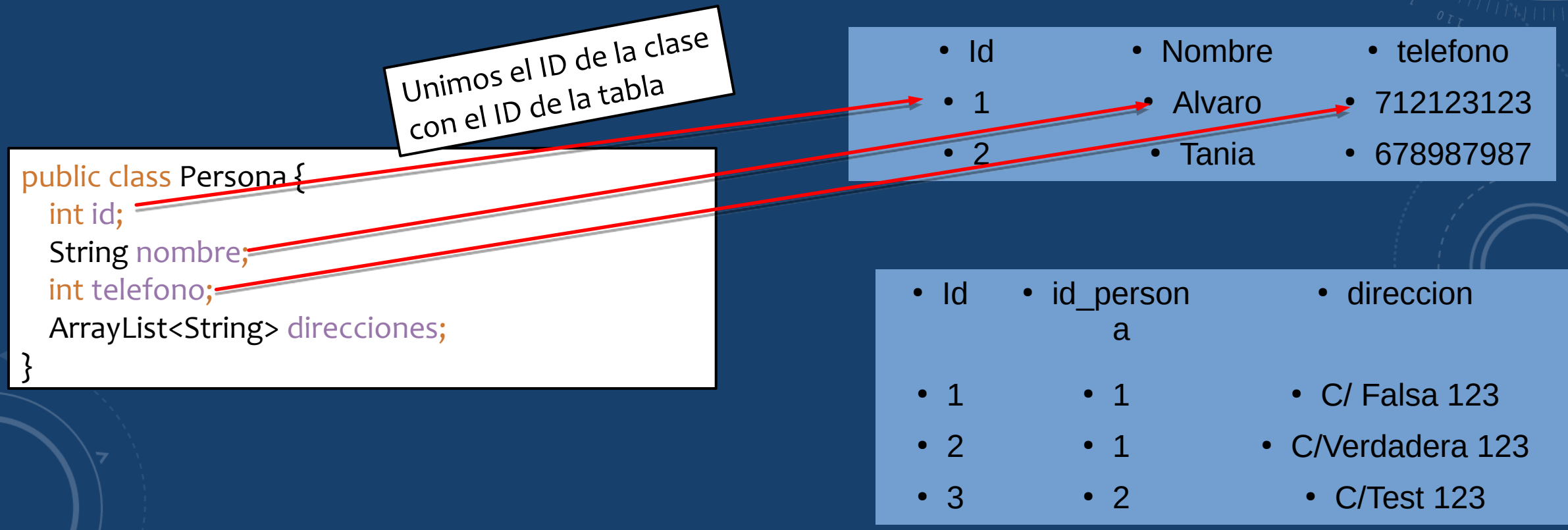
```
public class Persona {
   int id;
   String nombre;
   int telefono;
   ArrayList<String> direcciones;
}
```

Id	Nombre	telefono
1	Alvaro	712123123
2	Tania	678987987

Id	id_persona	direccion
1	1	C/ Falsa 123
2	1	C/Verdadera 123
3	2	C/Test 123

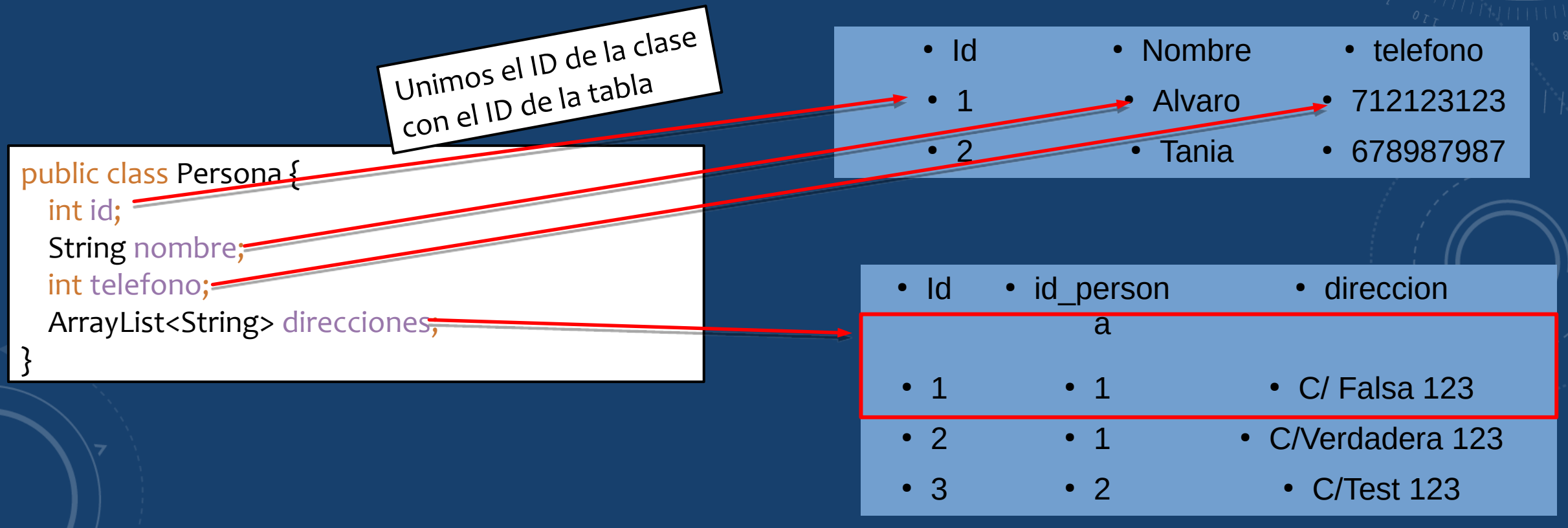
UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.



UNIFICACIÓN

- Para unir estas dos estructuras tablas y objetos (o clases mejor dicho), es necesario un mapeo de datos. En otras palabras indicar de forma manual como están unidos estos datos.



- Para unir estas dos estru
  datos. En otras palabras

Deberemos crear unas consultas que nos consigan todos estos datos y actuar en consecuencia.

```
select p.id, p.nombre, p.telefono, d.direccion from personas
p
INNER JOIN direcciones d ON p.id = d.id_persona
where p.id=1;
```

```
public class Persona {
   int id;
   String nombre;
   int telefono;
   ArrayList<String> direccio
}
```

• Id	• Nombre	• telefono
• 1	• Alvaro	• 712123123
• 2	• Tania	• 678987987

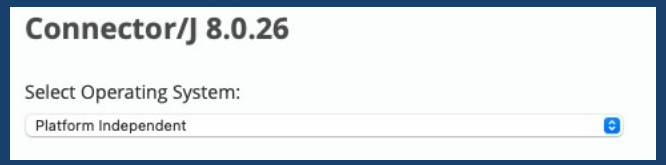
• Id	• id_persona	• direccion
• 1	• 1	• C/ Falsa 123
• 2	• 1	• C/Verdadera 123
• 3	• 2	• C/Test 123

CONNECTOR JDBC

- Para realizar las conexiones con la base de datos, utilizaremos JDBC (Java DataBase Connectivity), una librería encargada de darnos todas las funcionalidades de la base de datos
    - Conectarnos y Desconectarnos
    - Ejecutar consultas
        - Recuperar datos si fuese necesario
- No podemos utilizar JDBC directamente, ya que la programación interna se verá alterada según el motor de BD
    - Para utilizarlo deberemos buscar la librería de cada fabricante según el motor de base de datos utilizado.
    - Cambiar entre diferentes motores, serán tan sencillo como cambiar de librería, ya que utilizan la interfaz JDBC todas las peticiones se harán de la misma manera

INSTALAR JDBC PARA MYSQL - TRADICIONAL

- Buscar el connector de mysql
    - https://dev.mysql.com/downloads/connector/j/
    -
- Seleccionar la opción de platform independent



-
- Descargar cualquiera de las dos opciones, la diferencia está en la comprensión tar o zip.
- Por defecto os pedirá login, pero existe la opción de descargar sin loguearse "No thanks, just start my download"
- Una vez descargado deberemos añadirlo a nuestro IDE, este dependerá del mismo.

INSTALAR JDBC PARA MYSQL - TRADICIONAL

- Buscar el connector de mysql
  - https://dev.mysql.com/downloads/connector/j/
  -
- Seleccionar la opción de platform independent
-
- Descargar cualquiera de las dos opciones, la difere
- Por defecto os pedirá login, pero existe la op o thanks, just start my download”
- Una vez descargado deberemos añadirlo a nuestro IDE, te dependerá del mismo.

¡JDBC NO ES UN MOTOR DE BASE DE DATOS ES UN CONECTOR!
DEBEREMOS TENER NUESTRO MOTOR DE BASE DE DATOS INSTALADO Y FUNCIONANDO

INSTALAR JDBC - MAVEN

- Maven es una herramienta de software que entre otras utilidades puede gestionar las dependencias de nuestro proyecto.
- Esta herramienta contiene un fichero llamado pom.xml donde definiremos que librerías deseamos y nos las añadirá de forma automática.
    - No solo se pueden definir librerías, también podemos:
        - Pasos de compilación
        - Versión de Java utilizada
        - Autor del proyecto
        - Descripción del proyecto.
        - …
    - Como podemos observar es realmente potente y junto a gradle y npm forman parte de las tres herramientas de software más utilizadas en la actualidad
-

INSTALAR JDBC - MAVEN

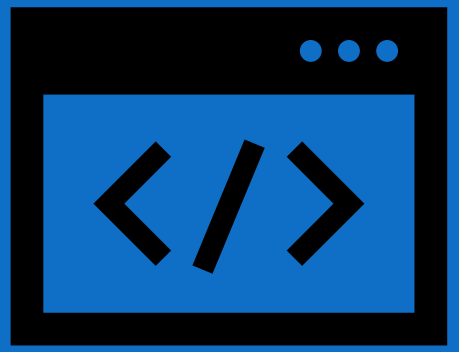
- Pasos para utilizar jdbc con maven
    1. Crear un proyecto Maven
    2. Editar el fichero pom.xml
    3. Añadir las dependencias (librerías necesarias)
    4. Actualizar fichero pom.xml

ARQUITECTURAS SOPORTADAS POR JDBC

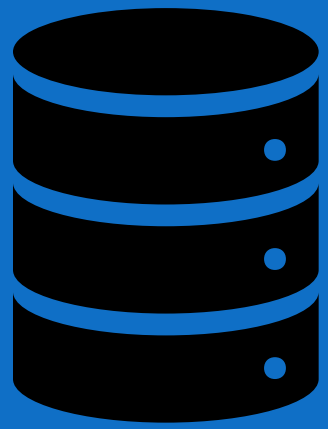
- A diferencia de otras tecnologías como por ejemplo Android, JDBC es compatible con el modelo de 2 y 3 capas
  - **2 capas**, el frontend (lo ve el cliente) y el backend (la lógica) está en la misma capa, por lo que no hay separación alguna, la capa que contiene la aplicación se conecta directamente al Sistema Gestor De Bases de datos.
    - Es el modelo menos común a día de hoy ya que las webs y apps móviles por seguridad no soportan esta arquitectura.
  -
  - **3 capas,** tenemos una clara diferenciación entre frontend, backend y SGDB. El Frontend se connecta mediante una api al backend y este utiliza JDBC para acceder al SGDB. Como podemos ver tenemos 3 capas diferenciadas entre ellas.
    - Es el modelo más común, podemos tener diferentes frontends (web, app) conectados al mismo backend ejecutando las mismas acciones. Es mucho más seguro ya que el frontend únicamente contiene la conexión con el backend y no contra el SGDB

ARQUITECTURAS SOPORTADAS POR JDBC

2 CAPAS



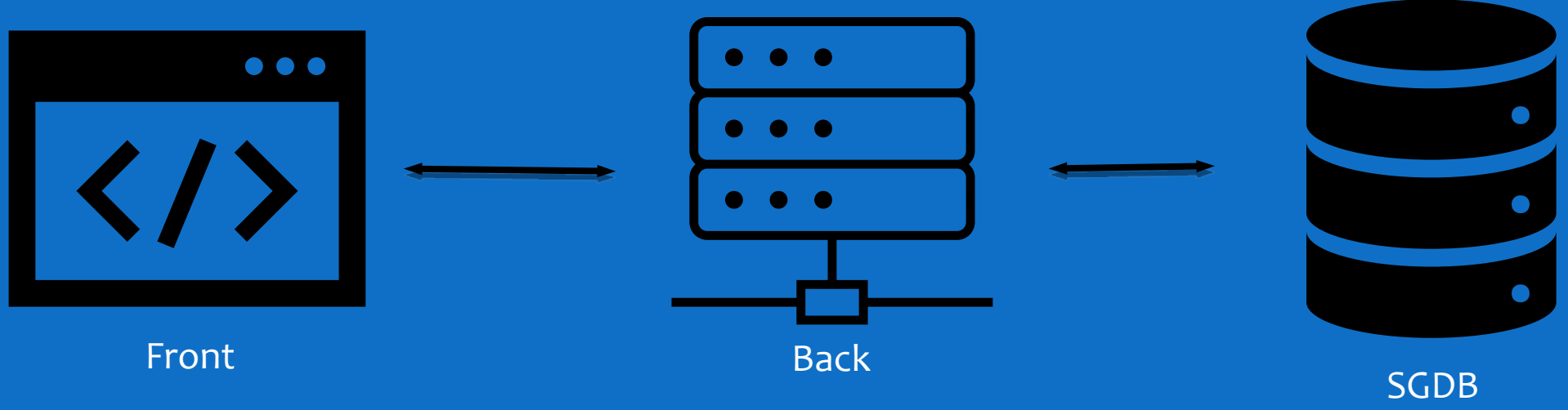
Front+Back unificados



SGDB

ARQUITECTURAS SOPORTADAS POR JDBC

- 3 CAPAS



JDBC - PROGRAMACIÓN

- Para conectarnos a una base de datos mediante JDBC deberemos seguir unos pasos específicos o no funcionará. Estamos accediendo a información no gestionada por nosotros, la gestiona el SGDB y por tanto deberemos delegarle esa confianza.
- Pasos:
    1. Importar las librerías necesarias
    2. Establecer el driver que se va a utilizar, el driver cambiará según el motor utilizado
    3. Establecer la conexión con el servidor
    4. Ejecutar las sentencias deseadas
        1. Las sentencias se ejecutan gracias a una clase llamada Statement
        2. Los resultados de las sentencias se recogen gracias a una clase llamada ResultSet
    5. Finalizar la conexión con el servidor
-

JDBC - PROGRAMACIÓN

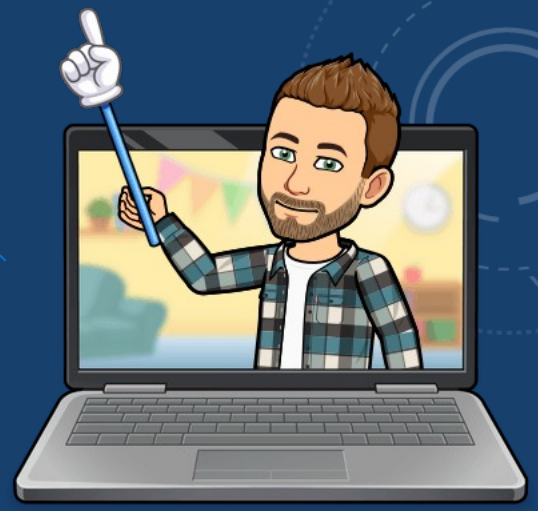
2. Establecer el driver que se va a utilizar, el driver cambiará según el motor utilizado
    - Para saber el driver deberemos mirar la documentación del fabricante
    -
    -
-
-

```
String driver = "com.mysql.cj.jdbc.Driver";
try {
   Class.forName(driver);
} catch (ClassNotFoundException ex) {
   System.err.println("No se ha encontrado el Driver: " +
driver);
   System.exit(-1);
}
```

JDBC - PROGRAMACIÓN

2. Establecer el driver que se va a utilizar, el driver cambiará según el motor utilizado
    - Para saber el driver deberemos mirar la documentación del fabricante
    -
    -
-
-

```
String driver = "com.mysql.cj.jdbc.Driver";
try {
   Class.forName(driver);
} catch (ClassNotFoundException ex) {
   System.err.println("No se ha encontrado el Driver: " +
driver);
   System.exit(-1);
}
```



JDBC - PROGRAMACIÓN

3. Establecer conexión con el servidor
    -
    -
-
-

```
String bdName = "mo6";
String url = "jdbc:mysql://localhost/" + bdName;
String usuario = "root";
String password = "secret";
Connection connection = null;
try {
   connection = DriverManager.getConnection(url, usuario,
password);
} catch (SQLException ex) {
   System.err.println("Error al connectarse con la BD");
   System.exit(-2);
}
```

JDBC - PROGRAMACIÓN

3. Establecer conexión con el servidor
    -
    -
-
-

```
String bdName = "mo6";
String url = "jdbc:mysql://localhost/" + bdName;
String usuario = "root";
String password = "secret";
Connection connection = null;
try {
   connection = DriverManager.getConnection(url, usuario,
password);
} catch (SQLException ex) {
   System.err.println("Error al connectarse con la BD");
   System.exit(-2);
}
```

INSTALACIÓN DE H2 - TRADICIONAL

- Primero de todo es descargar la librería
    - Igual que descargamos el conector de MySQL descargaremos el conector de h2.
        - http://www.h2database.com/html/download.html
    - Una vez descargado debemos añadir el jar a nuestro proyecto igual que hicimos con el conector de Mysql
    - Una vez añadido deberemos establecer la ruta de conexión.
-
    -

INSTALACIÓN DE H2 - TRADICIONAL

- Primero de todo es descargar la librería
  - Igual que descargamos el conector de My
    - http://www.h2database.com/html/d

A diferencia de MySQL no hace falta establecer el Driver, y solo con la URL se conectará a la BD.
La primera vez si esta no existe la creará por nosotros con un username y password especifico

```
String bdName = "mo6";
Path path = Path.of(bdName);
String url = "jdbc:h2:" + path.toAbsolutePath();
String usuario = "admin";
String password = "secret";
Connection connection = null;
try {
   connection = DriverManager.getConnection(url, usuario, password);
} catch (SQLException ex) {
   System.err.println("Error al connectarse con la BD");
   ex.printStackTrace();
   System.exit(-2);
}
```

JDBC - PROGRAMACIÓN

4. Crear objeto Statement para ejecutar sentencias DDL o DML
    -
    -
-
-

```
try {
   Statement estado = connection.createStatement();
   //sentencias
   estado.close();
}catch (SQLException ex){
   System.err.println("Error al crear el Statement");
   System.exit(-3);
}
```

CLASE STATEMENT

- La clase Statement es la que nos permite ejecutar consultas desde Java contra la BD.
- Existren 3 tipos de funciones para lanzar operaciones:
    - execute() → Es la más general permite ejecutar cualquier tipo de consulta DDL o DML. Devuelve un booleano
        - CIERTO → La respuesta es un objeto de tipo ResultSet
        - FALSE → Si la respuesta es el número de filas actualizadas o no tiene resultado
    - executeUpdate() → Ejecución utilizada para operaciones del tipo DML, devuelve un entero con la cantidad de filas afectadas
    - executeQuery() → Ejecución utilizada en operaciones de tipo select, devuelve un objeto ResultSet con los datos de la Respuesta

JDBC – SENTENCIAS DDL O DML

- Las sentencias DDL (Data Definition Language) son aquellas utilizadas para modificar la estructura de la base de datos
  - Crear base de datos, tabla, vista, etc.
  - Actualizar estructuras de datos
  - Eliminar estructuras de datos
- Las sentencias DML (Data Manipulation Language) son aquellas utilizadas para modificar la información almacenada en la base de datos
  - SELECT
  - UPDATE
  - DELETE
  - INSERT

REPASO SQL DDL

```
CREATE TABLE personas
(
   id      INTEGER PRIMARY KEY AUTO_INCREMENT,
   nombre   VARCHAR(30) NOT NULL,
   telefono VARCHAR(9)  NOT NULL
);

CREATE TABLE direcciones
(
   id        INTEGER PRIMARY KEY
AUTO_INCREMENT,
   id_persona INTEGER     NOT NULL,
   direccion  VARCHAR(50) NOT NULL
);
```

```
ALTER TABLE personas
   ADD COLUMN dni varchar(10);

ALTER TABLE personas ADD COLUMN edad INTEGER DEFAULT
18;

ALTER TABLE direcciones
   ADD FOREIGN KEY (id_persona) REFERENCES personas (id);

ALTER TABLE direcciones
   ADD CONSTRAINT PERSONA_ID_FK FOREIGN KEY
(id_persona) REFERENCES personas (id);

ALTER TABLE direcciones DROP CONSTRAINT
PERSONA_ID_FK;
```

```
DROP TABLE direcciones;
DROP TABLE personas;
```

EJEMPLO CREAR STATEMENT

```
private static Statement crearEstado(Connection connection) {
   try {
      return connection.createStatement();
   } catch (SQLException ex) {
      System.out.println("Error al crear el Statement");
      System.exit(-2);
   }
   return null;
}
```

EJEMPLO CREATE TABLE

```
private static void createTable(Connection connection) {
   Statement estado = crearEstado(connection);
   try {
      String query = "CREATE TABLE personas(id INTEGER PRIMARY KEY
AUTO_INCREMENT," +
            " nombre VARCHAR (30) NOT NULL, telefono VARCHAR (9) NOT NULL);";
      estado.execute(query);
      System.out.println("Tabla Personas creada correctamente");
      estado.close();
   } catch (SQLException ex) {
      if (ex.getMessage().contains("already exists")) {
         System.out.println("La tabla ya existe en la BD");
      } else {
         System.err.println(ex.getMessage());
      }
      System.exit(-3);
   }
}
```

EJEMPLO UPDATE TABLE

```
private static void updateTable(Connection connection) {
    Statement estado = crearEstado(connection);
    try {
        String query = "ALTER TABLE personas ADD COLUMN edad int default 18;";
        estado.execute(query);
        System.out.println("Tabla personas actualizada");
    } catch (SQLException ex) {
        System.err.println("Error actualizar la tabla personas");
        System.out.println(ex.getMessage());
        System.exit(-3);
    }
}
```

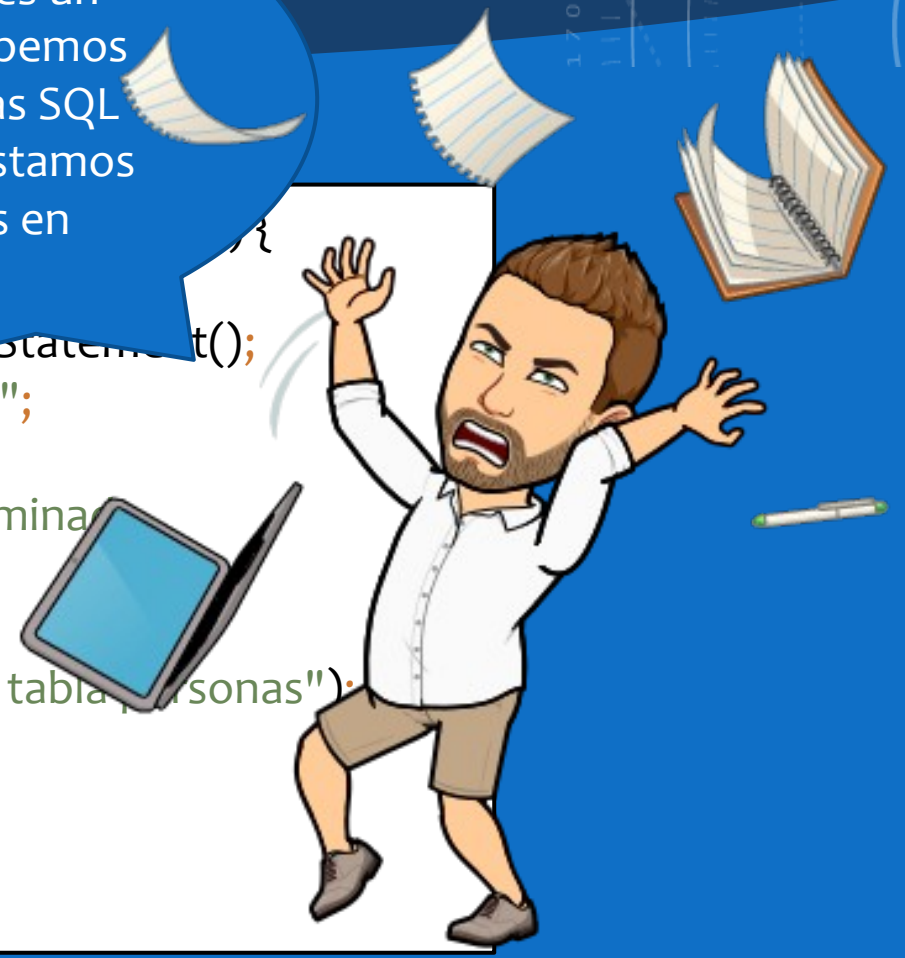
EJEMPLO DELETE TABLE

```
private static void deleteTable(Connection connection) {
   try {
      Statement estado = connection.createStatement();
      String query = "DROP TABLE personas;";
      estado.execute(query);
      System.out.println("Tabla personas eliminada
correctamente");
   } catch (SQLException ex) {
      System.out.println("Error al eliminar la tabla personas");
      System.out.println(ex.getMessage());
      System.exit(-3);
   }
}
```

EJEMPLO DELETE TABLE

Como podemos observar es un proceso muy “tedioso”, debemos incluir las propias sentencias SQL dentro de Java por lo que estamos mezclando dos lenguajes en

```
private st                                    ) {
   try {
      Statement estado =                    t();
      String query = "DROP TABLE personas;";
      estado.execute(query);
      System.out.println("Tabla personas elimina
correctamente");
   } catch (SQLException ex) {
      System.out.println("Error al eliminar la tabla personas");
      System.out.println(ex.getMessage());
      System.exit(-3);
   }
}
```



EJERCICIOS

- A partir del siguiente ejemplo https://dev.mysql.com/doc/employee/en/sakila-structure.html
    1. Crea las tablas:
        1. Empleado
            1. (id, nombre varchar(20),fecha_nacimiendo DATE, genero tinyint(1),departamento_id)
        2. Departamento
            1. (id, nombre varchar(20), ubicación varchar(100))
    2. Actualiza la tabla empleado
        1. Añade la columna departamento_id y crea la Foreign Key con departamento
            - 1 departamento muchos empleados
            - 1 empleado 1 departament
    3.
    4.

REPASO SQL DML

```
INSERT INTO personas (nombre, telefono, dni)
VALUES ('ALVARO', '612123123', '32211553-H'),
      ('TANIA', '689789789', '98776654-J')
```

```
UPDATE personas SET nombre = 'Alvaro Ortega' WHERE id=1;
```

```
DELETE FROM personas WHERE id = 2;
```

```
SELECT * FROM personas;

select nombre from personas where edad>=18 order by edad;
```

EJEMPLO SELECCIÓN DATOS

```
private static void selectData(Connection connection) {
   int id = 1;
   String query = "SELECT * from personas where id=" + id +
";";
   Statement estado = crearEstado(connection);
   ResultSet result = null;
   try {
      result = estado.executeQuery(query);
      while (!result.isLast()) {
         result.next();
         Persona persona = new Persona(
               result.getString("nombre"),
               Integer.parseInt(result.getString("telefono")),
               result.getString("dni"));
         System.out.println(persona);
      }
   } catch (SQLException e) {
      System.out.println("Error al realizar la consulta de
datos");
   }
```

```
Persona{ nombre='Alvaro Ortega', telefono=612123123, dni='32211553-H'}
```

RESULTSET

- ResultSet es una clase que contiene la información de toda la consulta de la BD.
- El funcionamiento de ResultSet es muy similar al de un iterador pero con algunas características
    - Un iterador es un objeto que permite recorrer una colección de elementos, no se puede utilizar un bucle tradicional ya que los elementos no están ordenados mediante un índice. El iterador nos proporciona los métodos para comprobar en cada vuelta del bucle si hay más datos (generalmente hasNext()) y recuperar el dato correspondiente (generalmente next()).
    - ResultSet tiene diferentes métodos para obtener los datos:
        - first() → Establece el apuntador en la primera fila recuperada de la BD
        - last() → Establece el apuntador en la última fila recuperada de la BD
        - isLast() → Nos dirá si el apuntador está en la última fila
        - next() → Obtiene los datos de la fila donde se encuentra el apuntador
        - getString(), getInt(), getDate(), etc… → Recupera la información en la columna enviada por parámetros
            - Se le puede enviar el índice de la columna a recuperar
            - Se le puede enviar el nombre de la columna a recuperar

¡¡¡PROBLEMA!!! SQL INJECTION

- Existe una gran vulnerabilidad en SQL cuando se combina código + SQL, por ejemplo en el select anterior:

```
int id = 1;
String query = "SELECT * from personas where id=" + id + ";";
```

- En cualquier momento, un atacante puede modificar el valor de id por cualquier otro valor, por ejemplo añadiendo más campos o información que no se debería de recuperar y ejecutando de esta manera queries que no deberían ser soportadas

```
String id = "1 OR 1=1";
String query = "SELECT * from personas where id=" + id + ";";
```

- En este caso la consulta a ejecutar cambia totalmente, ya que 1=1 es una condición que en SQL siempre es cierta. En el ejemplo es como eliminar la clausula where ya que el resultado a ejecutar será where id=1 or 1=1 y este será where id=1 or true
- El resultado será recuperar todas las filas y no solo la del usuario con el id 1, rompiendo seguridad de nuestra bd, ya que quizás la consulta únciamente debe recuperar la información del usuario actualmente logueado

SOLUCIONAR SQL INJECTION

- Existe otra clase para evitar estos ataques de inyección de datos.
- En lugar de utilizar Statement deberemos utilizar PreparedStatement. Esta clase evitará que hagamos combinaciones de código Java + código SQL.
- Funciona de forma muy sencilla, las variables serán modificadas por “?” un placeholder que determina que allí habrá una variable que posteriormente será reemplazada
    - Cada uno de ellos tiene un índice, que se utilizará para así poder reemplazar de forma correcta. La primera aparición tendrá el índice 1, la segunda el 2, etc.

EJEMPLO CREAR PREPAREDSTATEMENT

```
private static PreparedStatement crearEstado(Connection connection, String query)
{
   try {
      return connection.prepareStatement(query);
   } catch (SQLException ex) {
      System.out.println("Error al crear el Statement");
      System.exit(-2);
   }
   return null;
}
```

EJEMPLO SELECCIÓN DATOS

```
private static void selectPreparedData(Connection connection) {
    String nombre = "Alvaro Ortega OR 1=1";
    String query = "SELECT * FROM personas WHERE nombre = ?";
    PreparedStatement estado = crearEstado(connection, query);
    try {
        estado.setString(1, nombre);
        System.out.println(estado);
        ResultSet result = estado.executeQuery();
        int count = 0;
        while (result.next()) {
            count++;
            Persona persona = new Persona(
                    result.getString("nombre"),
                    Integer.parseInt(result.getString("telefono")),
                    result.getString("dni"));
            System.out.println(persona);
        }
        if (count == 0)
            System.out.println("No se han encontrado datos");
        result.close();
        estado.close();
    } catch (SQLException ex) {
        System.out.println("Error en la consulta de los datos");
        System.out.println(ex.getMessage());
    }
}
```

```
No se han encontrado datos
```

EJEMPLO SELECCIÓN DATOS

```
private static void selectPreparedData(Connection connection) {
   String nombre = "Alvaro Ortega";
   String query = "SELECT * FROM personas WHERE nombre = ?";
   PreparedStatement estado = crearEstado(connection, query);
   try {
      estado.setString(1, nombre);
      System.out.println(estado);
      ResultSet result = estado.executeQuery();
      int count = 0;
      while (result.next()) {
         count++;
         Persona persona = new Persona(
               result.getString("nombre"),
               Integer.parseInt(result.getString("telefono")),
               result.getString("dni"));
         System.out.println(persona);
      }
      if (count == 0)
         System.out.println("No se han encontrado datos");
      result.close();
      estado.close();
   } catch (SQLException ex) {
      System.out.println("Error en la consulta de los datos");
      System.out.println(ex.getMessage());
   }
}
```

```
Persona{ nombre='Alvaro Ortega', telefono=612123123, dni='32211553-H'}
```

EJEMPLO SELECCIÓN DATOS

- Un punto muy importante en las consultas del tipo PreparedStatement es como se usan los wildcard.
- Al inyectar un parámetro deberemos hacerlo ya con sus opciones prestablecidas, por ejemplo %Alvaro% para que el texto contenga Alvaro sin importar lo que exista al principio o al final del texto.
- Para ello al momento de inyectar el parámetro estableceremos el wildcard deseado.

```
estado.setString(1, "%" + nombre + "%");
```

EJEMPLO SELECCIÓN DATOS

- Un punto muy import
  wildcard.
- Al inyectar un parán
  %Alvaro% para que
  texto.
- Para ello al p nt de i

Repaso SQL:
Los wildcard son caracteres especiales (comodín) para substituir uno o más caracteres, por ejemplo el "%" es para cualquier texto.
Podeís encontrar más información en:
https://www.w3schools.com/sql/sql_wildcards.asp

```
estado.setString(
```

EJEMPLO INSERTAR DATOS

```
private static void insertData(Connection connection) {
   Persona persona = new Persona("Alvaro", 612123123, "32211553-H");
   Persona persona2 = new Persona("TANIA", 689789789, "98877654-J");
   Statement estado = crearEstado(connection);
   try {
      String query = "INSERT INTO personas (nombre, telefono, dni) " +
            "VALUES ('" + persona.getNombre() + "','" + persona.getTelefono() + "' , '" + persona.getDni() +
"'),"  +
            "      ('" + persona2.getNombre() + "','" + persona2.getTelefono() + "' , '" + persona2.getDni() +
"');";
      estado.executeUpdate(query);
   } catch (SQLException ex) {
      System.out.println("Error al insertar datos");
   }
}
```

EJEMPLO ACTUALIZAR DATOS

```
private static void updateData(Connection connection) {
   String query = "UPDATE personas SET nombre = ? where id=?";
   PreparedStatement estado = crearEstado(connection, query);
   try {
      estado.setString(1, "Alvaro");
      estado.setInt(2, 1);
      estado.executeUpdate();
      estado.close();
   } catch (SQLException ex) {
      System.out.println("Error al acutalizar los datos de la tabla");
      System.out.println(ex.getMessage());

   }
}
```

EJEMPLO ELIMINAR DATOS

```
private static void deleteData(Connection connection) {
   String query = "DELETE FROM personas where id = ?";
   PreparedStatement estado = crearEstado(connection,query);
   try {
      estado.setInt(1, 1);
      estado.executeUpdate();
      estado.close();
   }catch (SQLException ex){
      System.out.println("Error al eliminar de la BD");
      System.out.println(ex.getMessage());
   }
}
```

EJERCICIOS

- Siguiendo el ejemplo anterior https://dev.mysql.com/doc/employee/en/sakila-structure.html

3. Inserta datos en las diferentes tablas, todos los datos deben ir por parámetros y debe utilizarse el PreparedStatement para evitar SQLInjection
    1. Tabla departamento (el id se debe de autogenerar)
        1. 1, contabilidad, PB0
        2. 2, marketing, PB0,
        3. 3, RRHH, PB1
    2. Tabla Empleado (el id se debe de autogenerar)
        1. 1, Alvaro, 25/11/1979, 1, 1
        2. 2, Juan, 07,07,2001,1,2
        3. 3, Marta, 25/12/1997,0,3
        4. 4, Silvia, 01/04/1992,0,2
4. Recupera los datos de los empleados junto con el nombre del departamento al cual pertenecen
5. Recupera todos los usuarios ordenados por fecha de nacimiento, de mayor a menor
6. Edita la fecha de nacimiento del empleado con nombre Marta y establece “04/03/2002
7. Elimina el usuario con el id 1
8.

PRACTICA COMPLETA TEMA 2

- Realiza la siguiente practica, puedes preguntarme todas las dudas que tengas.

1. **Crea un programa que genere la siguiente estructura de packages y clases:**
    - Tema2
        - Common
            - Column
            - DDL
            - DML
        - Entities
            - Persona
            - Direccion
        - H2
            - Main
        - MySQL
            - Main
2. **Define las clases Persona y Direccion:**
    1. Persona (id, nombre, teléfono,dirección,ArrayList<Direccion>)
    2. Direccion(id, idPersona,dirección)
3. **Define la clase Column, esta representa una columna de BD. Decide que variables debe contener para generar una nueva columna en la creación de una nueva tabla**
    - Ejemplo: nombre VARCHAR(30) NOT NULL
    -
    -
4. **Define la clase DDL, debe contener las funciones:**
    1. Crear tabla, debe recibir por parámetros los datos de la tabla (nombre y columna)
        1. Crear ambas tablas
    2. Actualizar tabla
        1. Añade la FK entre Dirección y Persona
    3. Eliminar tabla
        1. Eliminar ambas tablas
5. **Define la clase DML, debe contener las funciones**
    1. Consultar datos
        1. Consultar 1 persona por id
        2. Consultar todas las personas
        3. Consultas 1 persona por id junto a sus direcciones
    2. Insertar datos
        1. Insertar persona
        2. Insertar dirección
    3. Actualizar datos
        1. Modificar una persona
        2. Modificar una dirección
    4. Eliminar datos
        1. Eliminar una persona y su dirección
6. **Debe poder ejecutarse en MySQL y H2, por ello tenemos 2 Main diferentes**
7.