



Programación

09 Gestión de Bases de Datos mediante código

José Luis González Sánchez





Contenidos

¿Qué voy a aprender?

Contenidos

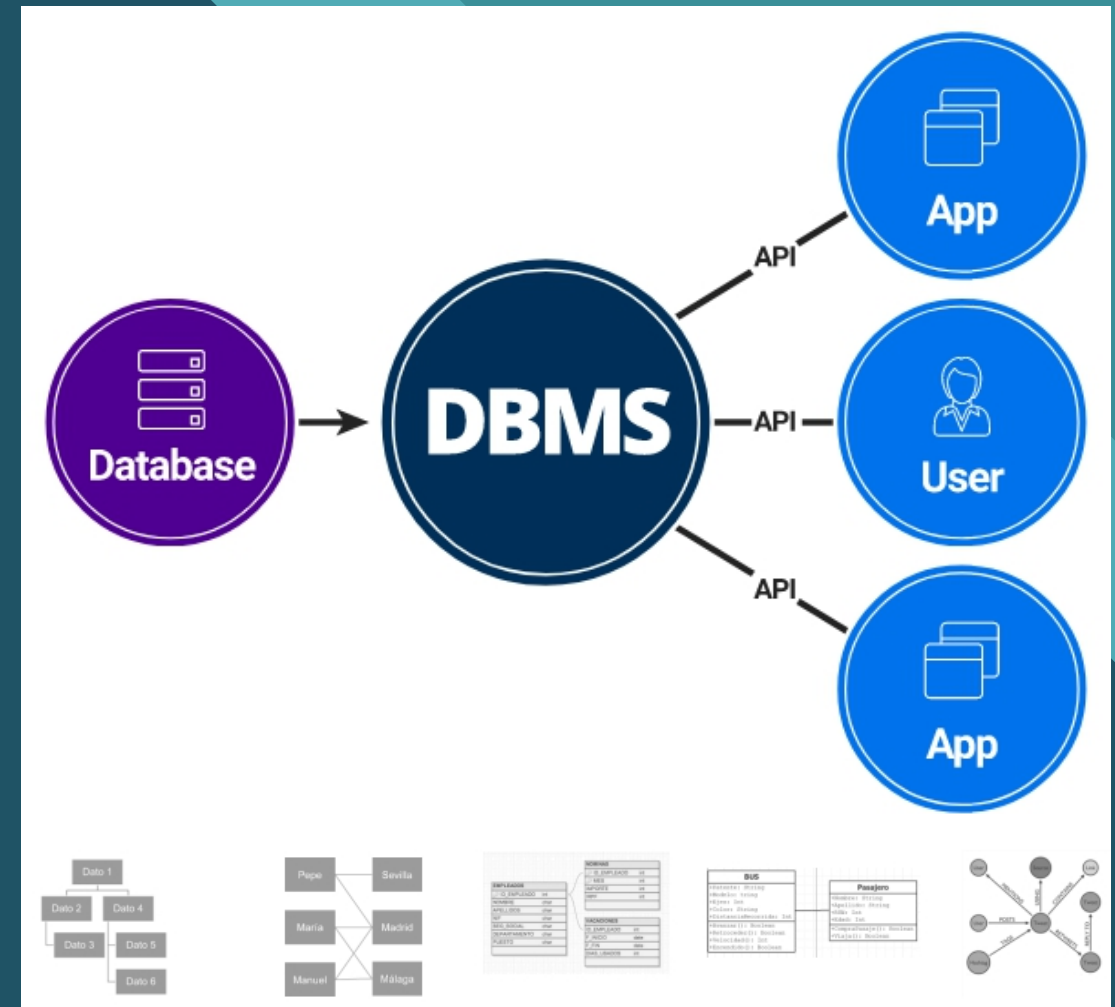
1. Bases de datos Relacionales
2. El Desfase Objeto-Relacional
3. Acceso a Base de Datos Relacionales
4. CRUD con JDBC
5. El problema de Null y el tipo Optional

Bases de Datos Relacionales

La información entre tablas, filas y columnas

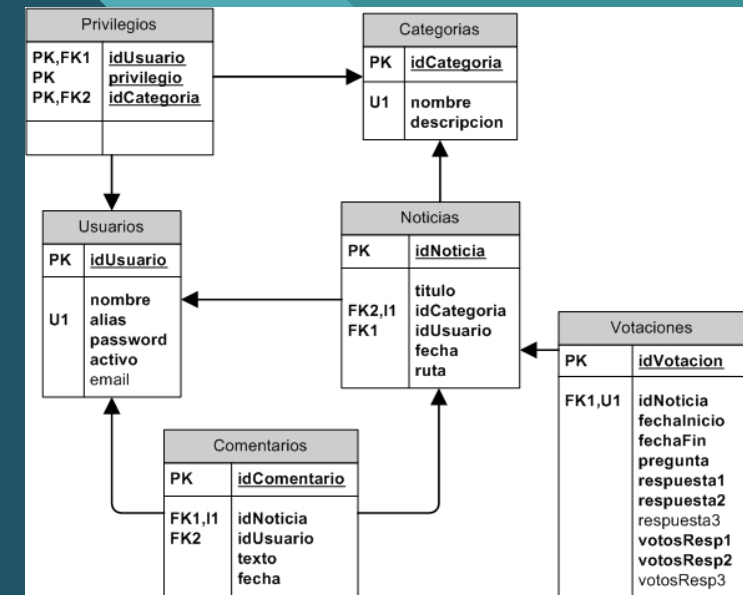
Bases de Datos Relacionales

- Desde que ser humano existe, siempre hemos tenido la necesidad de almacenar la información, ya sea desarrollando la escritura, luego el papel, imprenta, libros, cintas perforadas, disco duro, etc.
- A la misma vez que almacenamos la información debemos organizarla y procesarla ya sea en ficheros de acceso aleatorio, indexados o usando bases de datos.
- Tenemos distintos DBMS según si esquema lógico:
 - Jerárquicos: En árbol, estructura padre/hijo.
 - En Red: Nodos y enlaces. Manejo complejo
 - Relacional. Uso de tablas como única estructura
 - Orientados a objetos: Origen en la POO, para uso concreto.
 - NoSQL, Not Only SQL, Orientado a documentos, no todo es relacional.



Bases de Datos Relacionales

- La base de datos relacional (BDR) es un tipo de base de datos (BD) que cumple con el modelo relacional (el modelo más utilizado actualmente para implementar las BD ya planificadas). Tras ser postuladas sus bases en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos.
- Virtualmente, todos los sistemas de bases de datos relacionales utilizan SQL (Structured Query Language) para consultar y mantener la base de datos.
 - Una base de datos se compone de varias tablas, denominadas relaciones.
 - No pueden existir dos tablas con el mismo nombre ni registro.
 - Cada tabla es a su vez un conjunto de campos (columnas) y registros (filas).
 - La relación entre una tabla padre y un hijo se lleva a cabo por medio de las llaves primarias y llaves foráneas (o ajenas).
 - Las llaves primarias son la clave principal de un registro dentro de una tabla y estas deben cumplir con la integridad de datos.
 - Las llaves ajenas se colocan en la tabla hija, contienen el mismo valor que la llave primaria del registro padre; por medio de estas se hacen las formas relacionales.

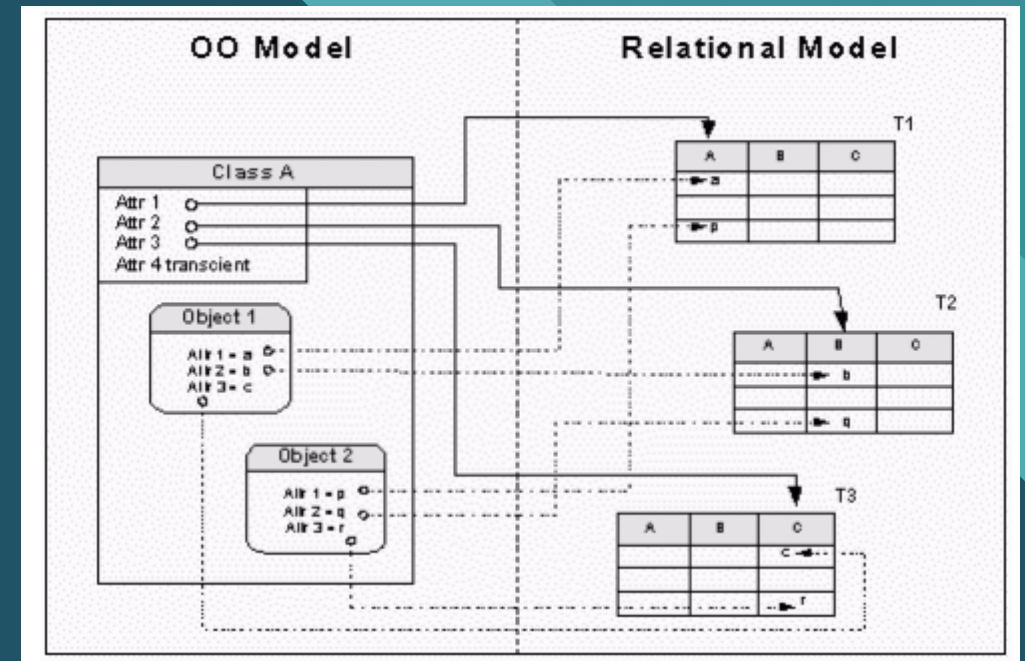


Desfase Objeto-Relacional

De tablas, fuilas y columnas a objetos

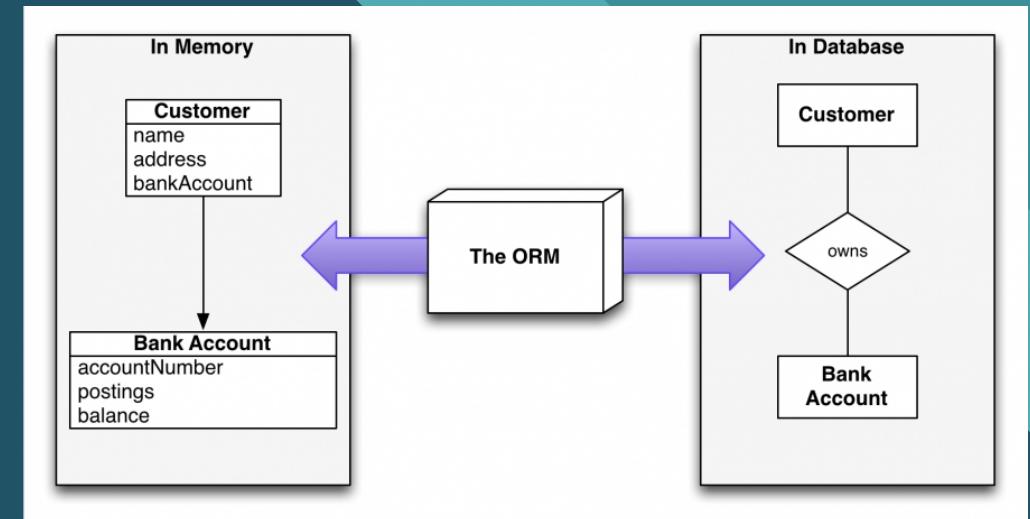
Desfase Objeto-Relacional

- También conocido como **impedancia objeto-relacional**, se trata de la diferencia existente entre la programación orientada a objetos y las bases de datos relacionales. Nos referimos a:
 - Lenguaje: El desarrollador debe conocer dos lenguajes: el de programación y el de la base de datos.
 - Tipos de datos: En la POO el tipo de datos es más complejo.
 - Paradigma de programación: Tiene que haber una traducción del modelo entidad-relación de la base de datos al modelo orientado a objetos del lenguaje de programación.
- Es decir, el modelo relacional trata de relaciones y conjuntos, mientras que POO trata los datos como objetos y asociaciones entre los mismos. Es decir al ahora de programar con BBDD Relacionales, debemos:
 - 1. Abrir una conexión.
 - 2. Crear sentencia SQL
 - 3. Copiar las propiedades del objeto y la sentencia y lanzar la consulta, o objetner de la consulta la información y **mapearla en el objeto**.



Desfase Objeto-Relacional. Mapeo Objeto Relacional

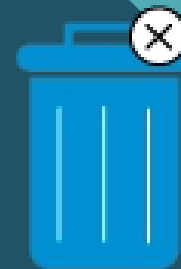
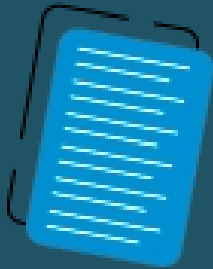
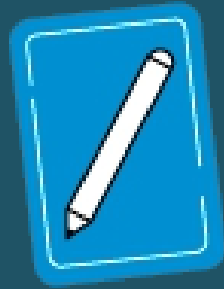
- Al trabajar con lenguajes orientados a objetos y bases de datos relacionales tenemos que estar continuamente descomponiendo los objetos para escribir la sentencia SQL para insertar, modificar o eliminar, o bien recomponer todos los atributos para formar el objeto cuando leamos algo de la base de datos.
- A este conjunto de técnicas que pueden desarrollarse de manera manual por el desarrollador o con apoyo de algún software se conoce como mapeo objeto-relacional.
- Aunque pueda parecer sencillo, es un problema si la base de datos cambia en estructura o tipo de datos, o simplemente el modelo de dominio de nuestro problema muta con el paso del tiempo, lo que puede provocar bastante cambios en cascada en nuestro código.



CRUD con JDBC

Create, Read, Update & Delete

CRUD



CRUD

CREATE

READ

UPDATE

DELETE

CRUD con JDBC. Conexión

- Es necesario **instalar el conector o driver de la base de datos**. En una aplicación JAVA, se instala un conector JDBC, que es el que implementa la funcionalidad de las clases de acceso a datos, y proporciona comunicación entre el API JDBC y el sistema gestor de bases de datos. Traduce los comandos al protocolo nativo del SGBD.
- La versión 3.0 de JDBC proporciona un **pool de conexiones**. Al iniciar un servidor JAVA EE, el pool de conexiones crea un número de conexiones físicas iniciales. Cuando un objeto Java del servidor necesita una conexión (método `dataSource.getConnection()`), la fuente de datos `javax.sql.DataSource` habla con el pool de conexiones y éste le entrega una conexión lógica `java.sql.Connection`. Esta conexión es la que recibe el objeto Java.
- Para cerrar una conexión (método `connection.close()`) la fuente de datos `javax.sql.DataSource` habla con el pool de conexiones y devuelve la conexión lógica.
- Si hay una alta demanda de conexiones a la base de datos, el pool de conexiones crea más conexiones físicas de objetos tipo `Connection`.
- El código básico para conectarnos a una base de datos con pool de conexiones a través de JNDI podría ser:
 - `javax.naming.Context ctx = new InitialContext();`
 - `dataSource = (DataSource) ctx.lookup("java:comp/env/jdbc/Basededatos");`
- Para realizar una operación, obtenemos una conexión lógica:
 - `connection = dataSource.getConnection();`
- Y para cerrarla:
 - `connection.close();`

CRUD con JDBC. Conexión

```
// Conexión a una BBDD
Connection conexion = null;

try {
    conexion = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/basededatos",
        "usuario", "password");
} catch (ClassNotFoundException cnfe) {
    cnfe.printStackTrace();
} catch (SQLException sqle) {
    sqle.printStackTrace();
} catch (InstantiationException ie) {
    ie.printStackTrace();
} catch (IllegalAccessException iae) {
    iae.printStackTrace();
}
```

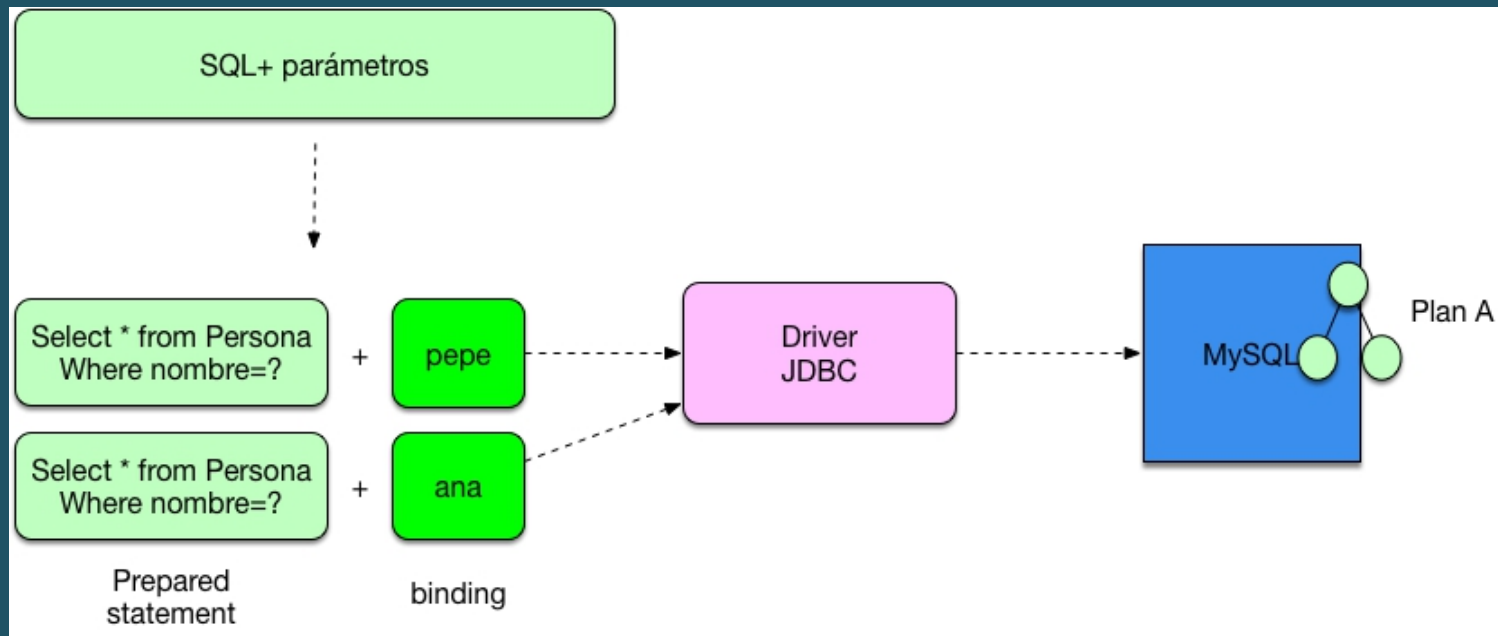
```
// Desconexión a una BBDD
. . .
try {
    conexion.close();
    conexion = null;
} catch (SQLException sqle) {
    sqle.printStackTrace();
}
. . .
```

CRUD con JDBC. Consultas

- Para operar con una base de datos, nuestra aplicación debe:
 - Cargar el driver necesario.
 - Establecer una conexión con la base de datos.
 - Enviar consultas SQL, procesando el resultado.
 - Liberar los recursos al terminar.
 - Gestionar posibles errores.
- Se pueden usar tres tipos de sentencias:
 - **Statement**: Sentencias SQL
 - **PreparedStatement**: Consultas preparadas, como las que tienen parámetros. Recomendado
 - **CallableStatement**: Ejecutar procedimientos almacenados en la base de datos.
- JDBC distingue dos tipos de consultas:
 - Consultas: SELECT
 - Actualizaciones: UPDATE, INSERT, DELETE y sentencias DDL.

CRUD con JDBC. Consultas

- El uso de **JDBC Prepared Statement** es hoy en día prácticamente obligatorio. Son consultas precompiladas por lo que son más eficientes y pueden tener parámetros reutilizando el mismo plan de consulta y evitando ataques por inyección SQL. Se instancia de la siguiente manera (ejemplo con tabla medicamentos):
 - Si hay que emplear parámetros en una consulta, se puede hacer usando el carácter “?”:
 - Establecemos los parámetros de una consulta utilizando métodos set que dependen del tipo SQL de la columna y el orden donde aparecen.
 - Finalmente ejecutamos la consulta con `executeQuery()` o `executeUpdate()`, dependiendo del tipo de consulta que sea.



CRUD con JDBC. Selección

- Una consulta de selección sobre una tabla de la base de datos devuelve un conjunto de datos organizados en filas y columnas (registros).
- Usamos `executedQuery`
- En Java almacenamos esa información mediante un objeto de la clase **ResultSet**, el cual está formado por filas y columnas.
- Un **ResultSet** solo puede recorrerse fila por fila, desde la primera hasta la última en una única dirección de su método `next()`.
- Además, podemos acceder a cualquier columna de cada fila mediante algunos de sus métodos (`getInt()`, `getString()`, `getObject()`, etc). Las columnas se numeran empezando en el nº 1.
- Obtener las columnas de una consulta `Select`. Si en algún caso al realizar una consulta no sabemos cuántas columnas nos va a devolver, podemos usar el método `getMetaData()` que devuelve un objeto del tipo `ResultSetMetaData`. Se puede obtener antes si se usa `PreparedStatement`

```
// Consulta tipo Select
...
String sentenciaSql = "SELECT nombre, precio FROM productos " +
    "WHERE precio = ?";
PreparedStatement sentencia = null;
ResultSet resultado = null;

try {
    sentencia = conexion.prepareStatement(sentenciaSql);
    sentencia.setFloat(1, filtroPrecio);
    resultado = sentencia.executeQuery();
    while (resultado.next()) {
        System.out.println("nombre: " + resultado.getString(1));
        System.out.println("precio: " + resultado.getFloat(2));
    }
} catch (SQLException sqle) {
    sqle.printStackTrace();
} finally {
    if (sentencia != null)
        try {
            sentencia.close();
            resultado.close();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
}
...
```


CRUD con JDBC. Insercción, Actualización y Borrado

- Selen devolver un entero con el número de registros afectados
- Se usa `executeUpdate()`

```
// Consulta tipo insert
...
String sentenciaSql = "INSERT INTO productos (nombre, precio) VALUES (?, ?)";
PreparedStatement sentencia = null;

try {
    sentencia = conexion.prepareStatement(sentenciaSql);
    sentencia.setString(1, nombreProducto);
    sentencia.setFloat(2, precioProducto);
    sentencia.executeUpdate();
} catch (SQLException sqle) {
    sqle.printStackTrace();
} finally {
    if (sentencia != null)
        try {
            sentencia.close();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
}
...
```

```
// Consulta tipo update
...
String sentenciaSql = "UPDATE productos SET nombre = ?, precio = ? " +
    "WHERE nombre = ?";
PreparedStatement sentencia = null;

try {
    sentencia = conexion.prepareStatement(sentenciaSql);
    sentencia.setString(1, nuevoNombreProducto);
    sentencia.setFloat(2, precioProducto);
    sentencia.setString(3, nombreProducto);
    sentencia.executeUpdate();
} catch (SQLException sqle) {
    sqle.printStackTrace();
} finally {
    if (sentencia != null)
        try {
            sentencia.close();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
}
...
```

```
// Consulta tipo delete
...
String sentenciaSql = "DELETE productos WHERE nombre = ?";
PreparedStatement sentencia = null;

try {
    sentencia = conexion.prepareStatement(sentenciaSql);
    sentencia.setString(1, nombreProducto);
    sentencia.executeUpdate();
} catch (SQLException sqle) {
    sqle.printStackTrace();
} finally {
    if (sentencia != null)
        try {
            sentencia.close();
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
}
...
```

El problema de Null y el tipo Optional

Evitando NullPointerException y controlando nuestro código

Null y Optional

- El problema de NullPointerException o lidiar con los null es uno de los problemas más importantes y costosos a la hora de programar. Es por ello que debemos llenar nuestro código con condiciones para controlarlo. Si estas comprobaciones se nos olvidan podemos tener problemas, o la famosa excepción.
- Desde Java 8, existe el tipo Optional. Este nos permite crear un “envoltorio” obligándonos a usar sus métodos para evitar el null, o simplemente lanzar una excepción si es necesario.

```
String name = "baeldung";
```

```
Optional<String> opt = Optional.of(name);
```

```
assertTrue(opt.isPresent());
```

```
assertFalse(opt.isEmpty());
```

```
Optional.ofNullable(nullName).orElse("john");
```

```
Optional.ofNullable(nullName).orElseThrow(...);
```

```
Optional<String> opt = Optional.ofNullable(null);
```

```
String name = opt.get();
```



“

"La programación es una carrera entre los desarrolladores, intentando construir mayores y mejores programas a prueba de idiotas, y el universo, intentando producir mayores y mejores idiotas. Por ahora va ganando el Universo"

- Rich Cook



”

Recursos

- Twitter: <https://twitter.com/joseluisgonsan>
- GitHub: <https://github.com/joseluisgs>
- Web: <https://joseluisgs.github.io>
- Discord: <https://discord.gg/eFMKxfPY>
- Aula Virtual: <https://aulavirtual33.educa.madrid.org/ies.luisvives.leganes/course/view.php?id=247>



Gracias

José Luis González Sánchez

