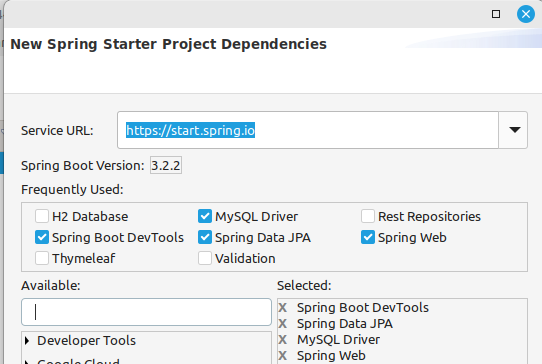
**Tarea 1 – UD5**

**Empezando con Base de Datos**

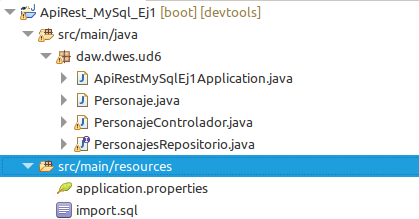
[**https://github.com/AranchaC/D1-UD5.git**](https://github.com/AranchaC/D1-UD5.git)

**Apartado 1.- API REST con MySql: cambiar la base de datos H2 por Mysql y así gozar de persistencia de los datos.**

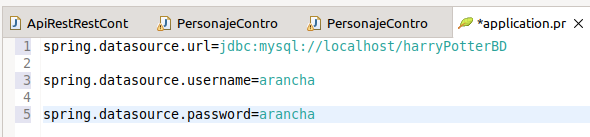
Creo proyecto nuevo con las siguientes dependencias:

****

Copio los archivos .java y el imports.sql de la práctica anterior:



Añado los datos de la BBDD (url, usuario, password) en aplicaciones.properties:



Voy a elegir crear la tabla automáticamente con las siguientes líneas en applicatio.properties:

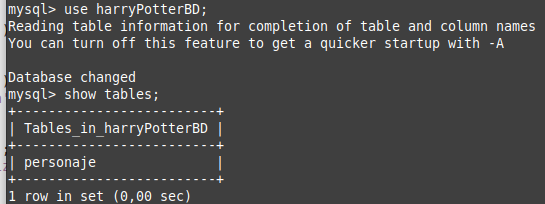


* **spring.jpa.generate-ddl=true:** Indica a Spring Boot que genere el DDL (Data Definition Language) automáticamente.
* **spring.jpa.hibernate.ddl-auto=update:** Le dice a Hibernate que actualice automáticamente la estructura de la base de datos según los cambios en las entidades de JPA.

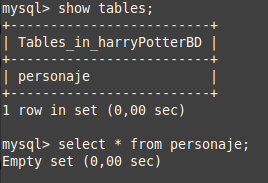
Creo una base de datos y un usuario con permisos totales sobre ella para nuestra aplicación (con los mismos datos indicados en application.properties):



Ejecuto la aplicación desde STS, y viendo que en consola no sale ningún error, visualizo en MySql que se ha autocreado la tabla en mi bbdd, con mi clase entidad como nombre de la tabla:

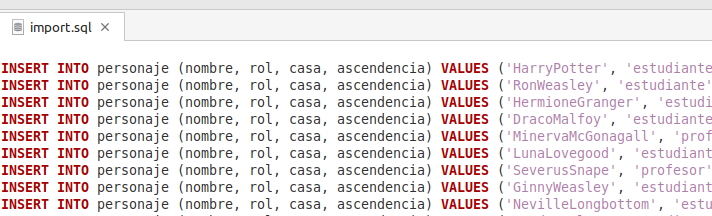


Pero si accedemos a la tabla, está vacía por lo que vemos que no se ha cargado el import.sql:



Me doy cuenta de que en mi archivo imports.sql tengo que cada insert se añadan a una tabla *personajes* y que el código crea una tabla *personaje.*

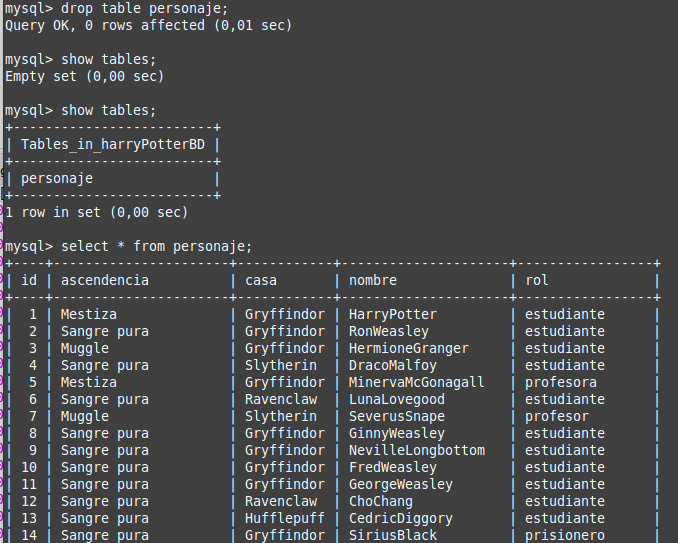
Por lo que modifico el imports.sql cambiando de personajes a personaje:



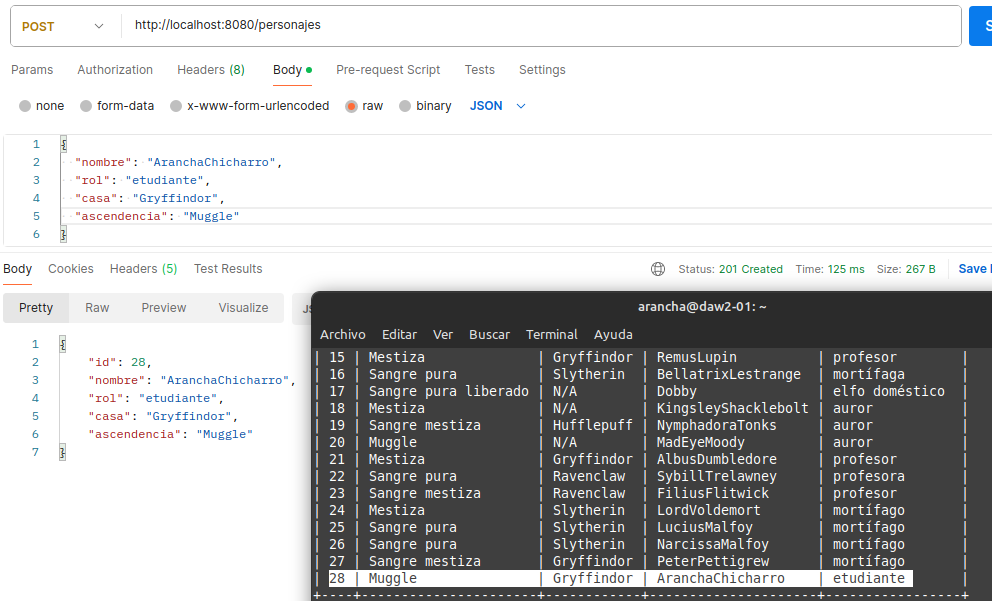
Ejecuto y también sale error. Gracias a los compañeros, me dicen que en la línea de spring.jpa.hibernate.ddl-auto=create de application.properties, cambie update por create.



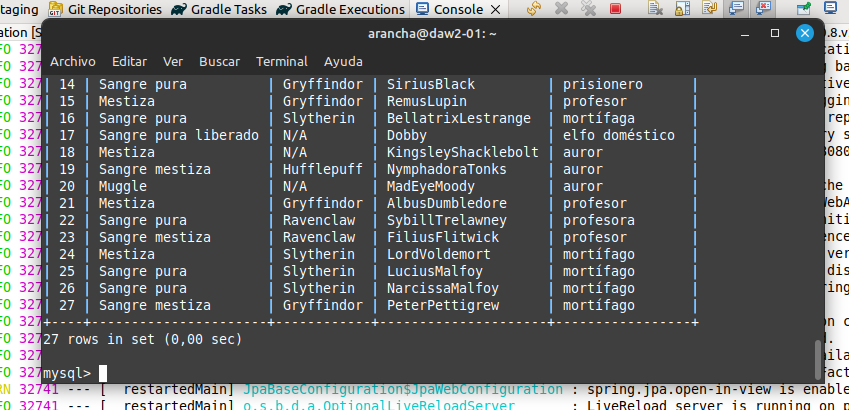
Borro la anterior tabla *personaje,* y ejecuto de nuevo la aplicación y ahora sí que se han cargado los inserts:



Voy a probar a insertar un nuevo personaje con Post crearPersonaje:



Y si paro la aplicación y la vuelvo a ejecutar, este elemento nuevo no se guarda porque vuelve a crear la tabla con los datos anteriores, es decir, la tabla nueva pisa a la que ya estaba:

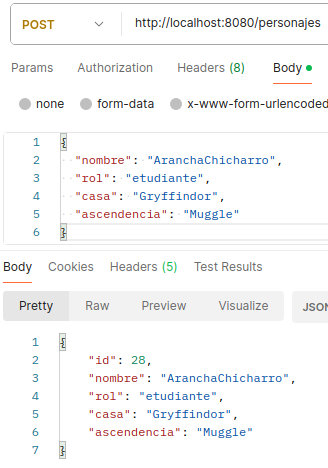
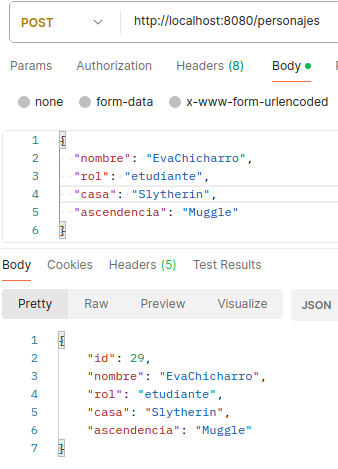


Y entonces, ahora en application.properties cambio el *create* por *update*.

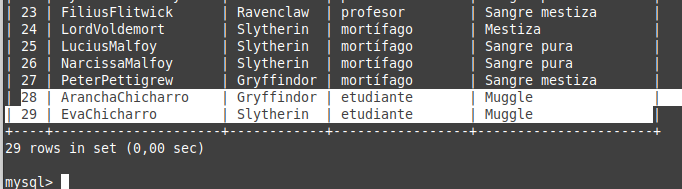


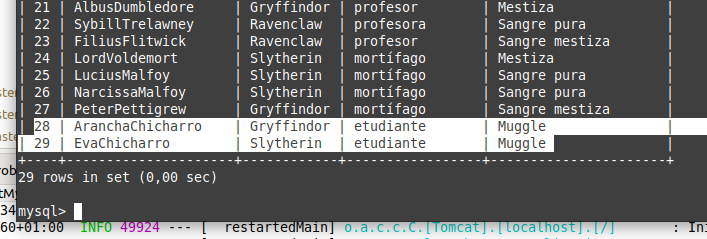
Voy a volver a crear dos nuevos personajes con post crearPersonaje:

* AranchaChicharro
* EvaChicharro

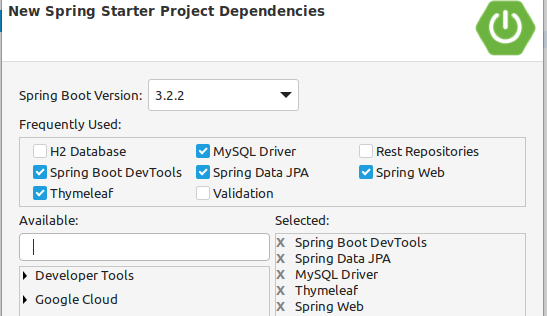
Compruebo en lista:



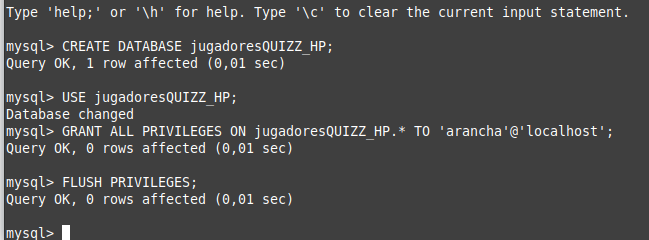
Paro la ejecución de la api y la vuelvo a ejecutar, y vemos que ahora sí que se han guardado los personajes que he añadido nuevos:  


**2.- Continuar el QUIZZ de la UD5 haciendo que las puntuaciones se guarden en base de datos.**

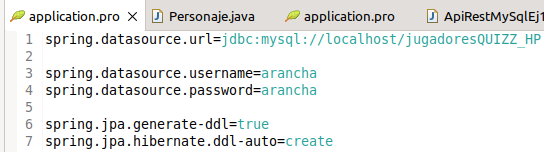
Creo proyecto con las siguientes dependencias:



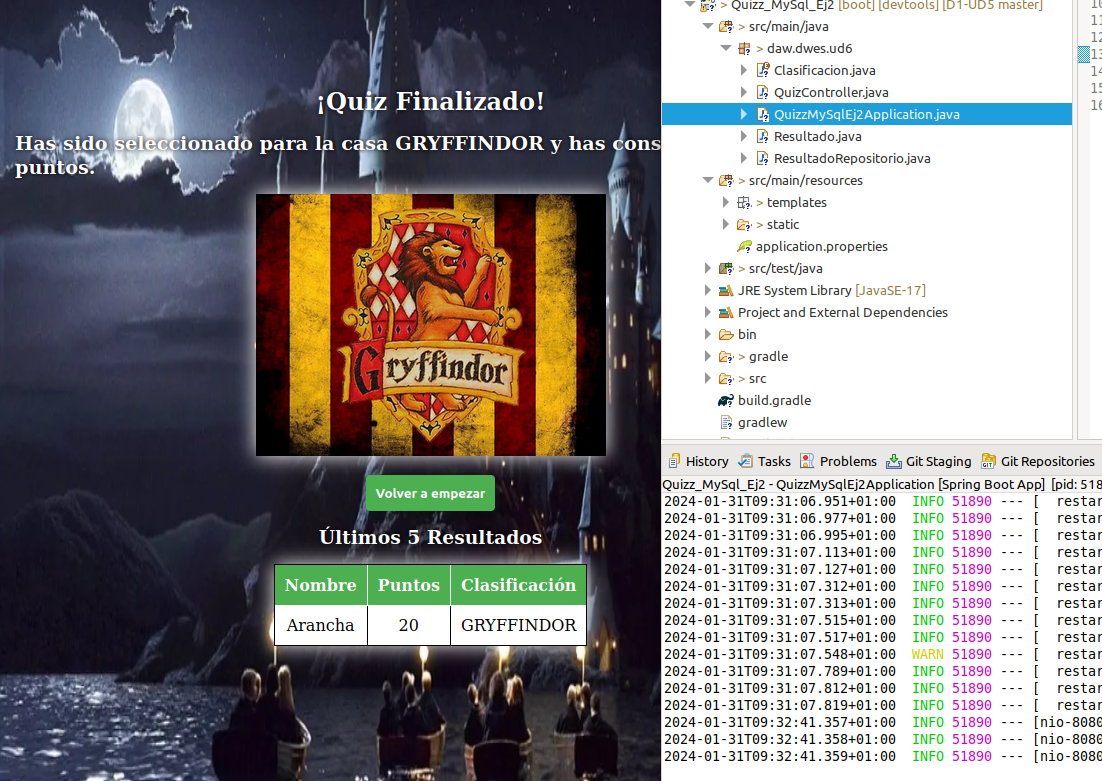
Primero voy a crear la bbdd para esta api y voy a aprovechar el usuario del ej anterior, pero tendré que darle permisos a esta bbdd nueva:



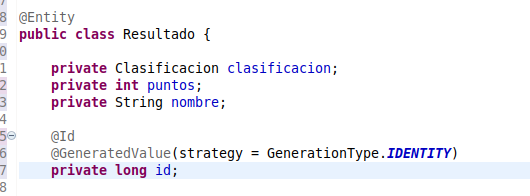
Modifico el application.properties con los datos de esta bbdd, igual que en el ej1:



Copio los archivos del proyecto del quizz del tema anterior y ejecuto la api, asegurándome de que no hay ningún error en consola ni en el navegador:

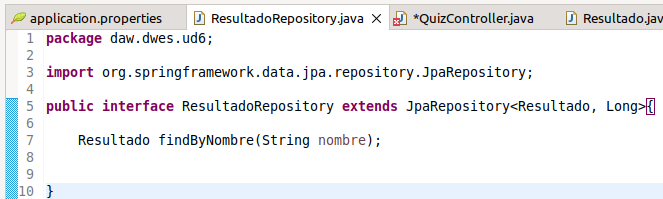


Como en el caso de esta api, mi objeto entidad, *Resultado* no tiene atributo identificador, voy a crear un id automático, igual que hicimos en el reto anterior. También añado la anotación *@Entity* a la clase *Resultado:*



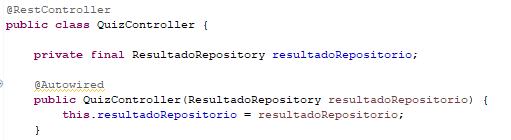
En mi clase *ResultadoRepositorio* la convierto en interface (creando clase nueva), y que extienda de *JpaRepository* sobre mi entidad y la clave, que será Resultado y Long (sobre esto me he fijado en ejemplo de ej anterior)

Y creo función *findByNombre* (que aún no se si será necesario):



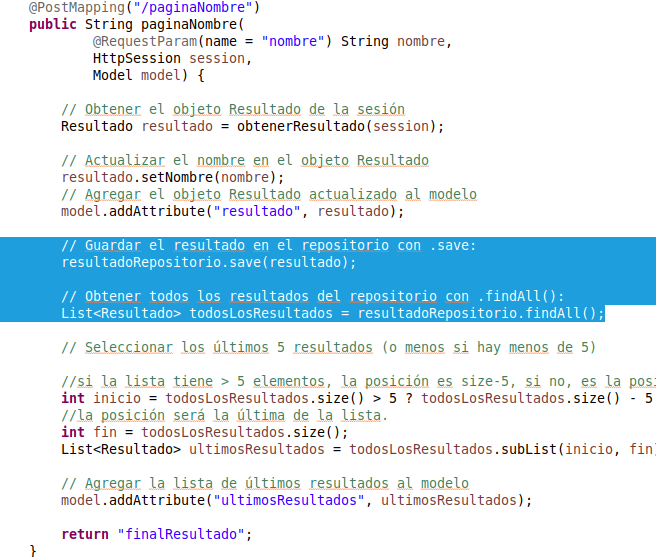
Ahora voy al **Controlador**.

Cambio la anotación *@Controller* de la clase por *@RestController.* Me aseguro de que hay variable de tipo *ResultadoRepository*, y creo constructor con anotación *@Autowired* para iniciar esta variable:



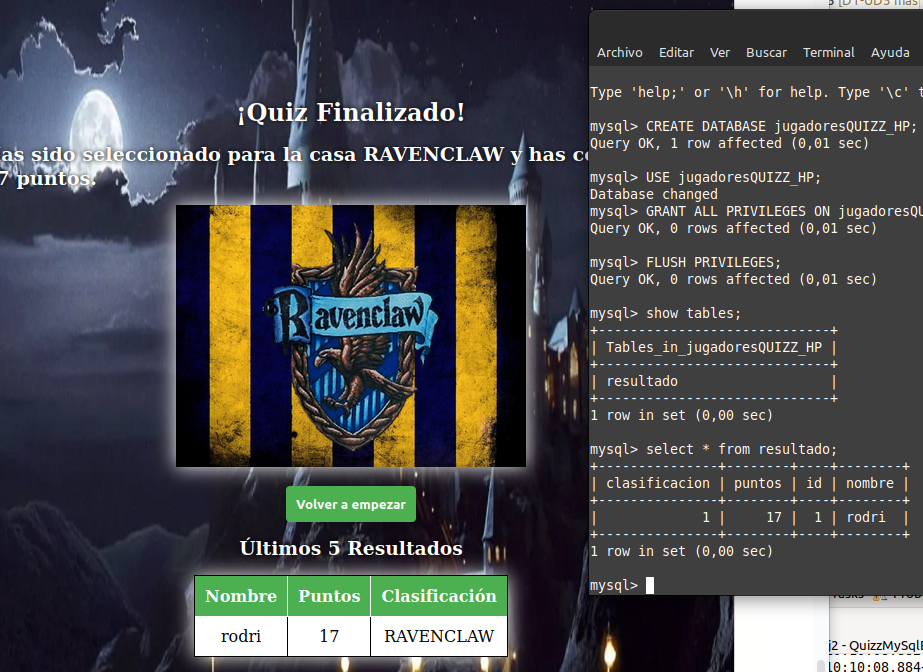
Ahora tengo que cambiar la forma de guardar los datos y acceder a ellos, pues ahora vamos a usar la bbdd.

Me dirijo al método de la última página, donde gestiono esta parte de lógica, y usando las funciones de *JpaRepository*, ahora aplicadas a mi *ResultadoRepository*.

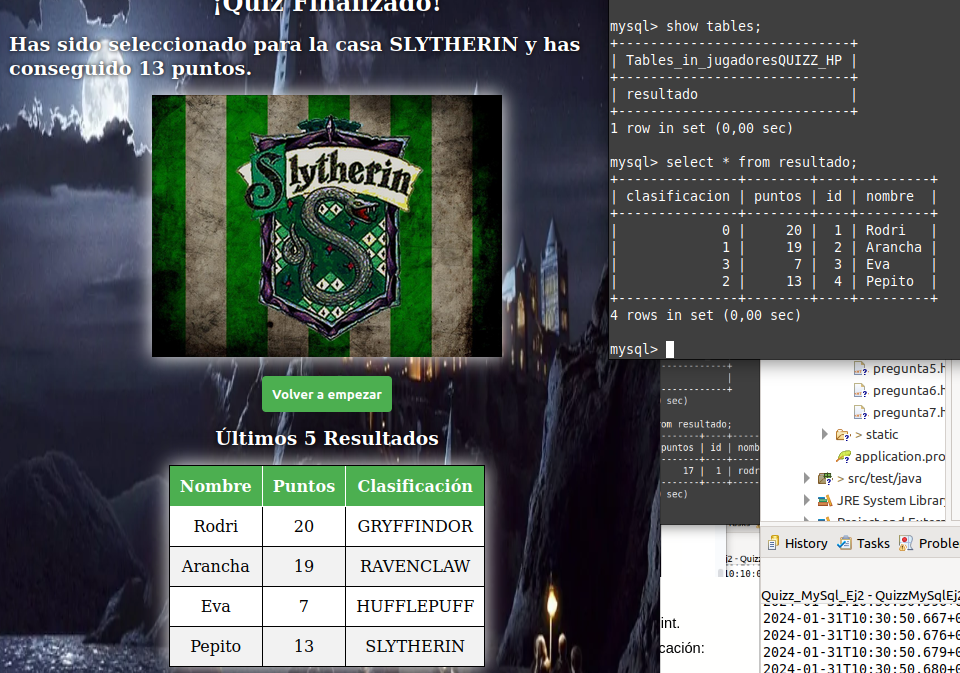


* **resultadoRepositorio.save(resultado);:** con esta función guardo el objeto/entidad resultado, que se compone de nombre, puntuación y clasificación en la bbdd.
* **List<Resultado> todosLosResultados = resultadoRepositorio.findAll();:** creo una lista de todosLosResultados y le asigno todos los valores del repositorio mediante la función .findAll().

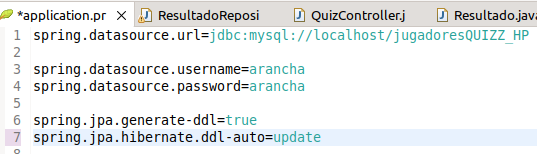
Ahora que no tengo ningún error en el código, ejecuto la api y tras ver que no hay ningún error en la consola, pruebo en el navegador a hacer una vez el quiz y compruebo en MySql que se ha creado la tabla en la bbdd correspondiente y que se ha introducido el jugador del quizz del navegador con su resultado:



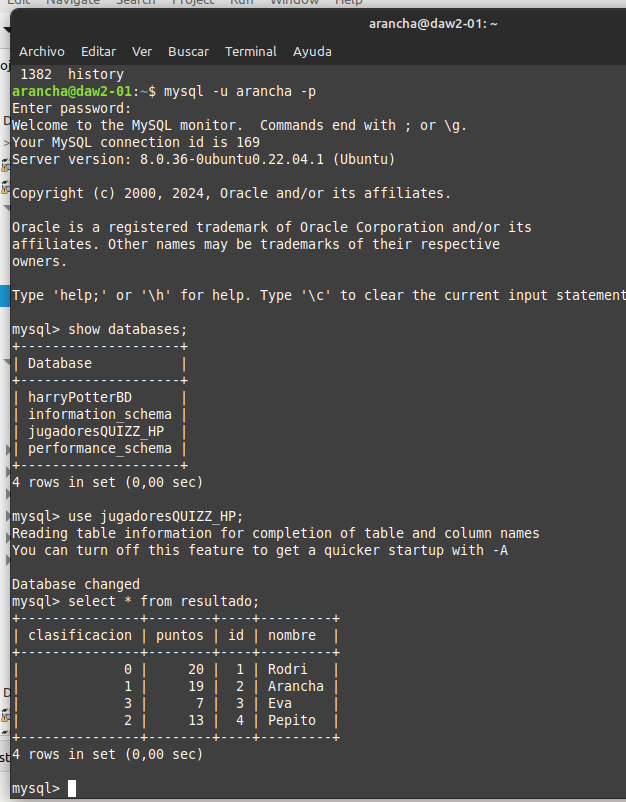
En mi caso, la clasificación es de tipo enum y vemos que en sql aparece de tipo int. Voy a probar a poner más jugadores intentando que salga uno de cada clasificación:



Ahora algo importante, para que se queden estos datos guardados en la bbdd cuando detengamos la api y la volvamos a ejecutar, en *application.properties* cambiamos la anotación **create** por **update**, pues si no, vuelve a hacer la acción de crear tabla y sustituiría a la que teníamos. Y al poner *update* en vez de create, la actualizamos:



Ahora hago la prueba. Detengo la api y la vuelvo a ejecutar, y compruebo en MySql (saliendo y entrando nuevamente) que ahora sí que tenemos los resultados anteriores guardados:

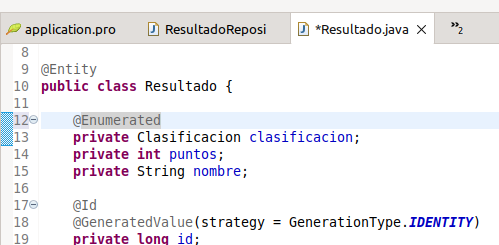
****

Introducimos más jugadores para verificar que se añaden a la tabla anterior:

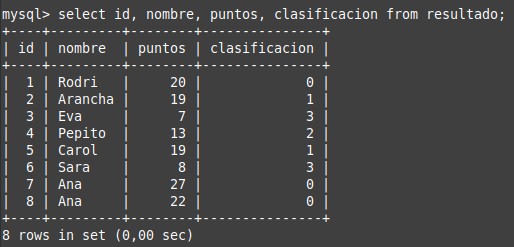


Al ver los materiales, he visto que existe la anotación @*Enumerated* que se usa para indicar que un tipo enumerado se almacene como String de lo contrario se almacena como ordinal, como ha pasado en mi tabla.

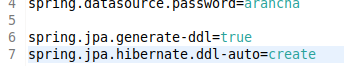
Voy incluir esta anotación en mi clase entidad *Resultado,* en el atributo *clasificacion:*



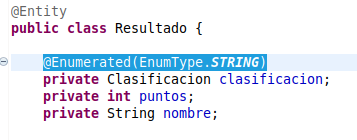
Ejecuto la api de nuevo y compruebo los datos de la tabla en MySql, pero no se actualiza, seguramente sea porque la columna ya esté creada y reconocida como int:



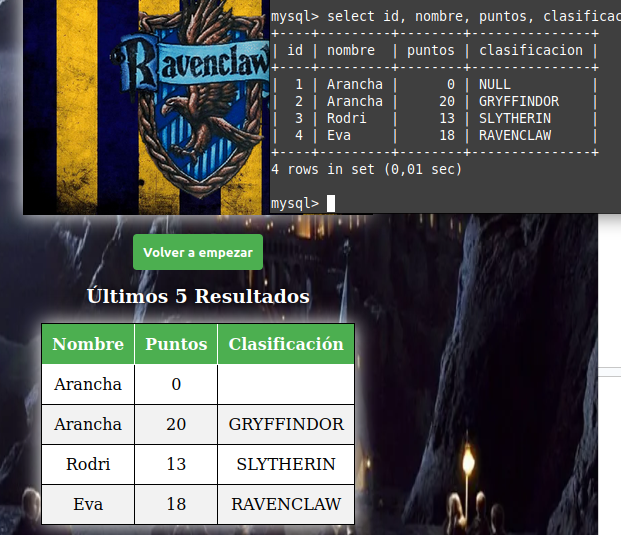
Voy a probar a eliminar la tabla, poniendo en application.properties otra vez create:



Pero tampoco salía la clasificación como string, viendo otra vez los materiales hay un añadido a la anotación @*Enumerated* que es para especificar el tipo de dato enumerado. Habría que añadir EnunType y en mi caso String:



Y vuelvo a iniciar la api, con create en properties, y realizo el quiz varias veces para generar usuarios, y ahora sí que salen los valores de la columna clasificación como string. *(Importante! Volver a cambiar de create a update en properties):*



**3.- Para el bien (dos de los tres puntos)**

* **Documentar las pruebas de qué pasa si no existe la tabla y qué ajusten sirven para que se cree automáticamente y si da algún problema si ya está creada.**

*Esta parte se ha ido documentando a medida que se ha hecho la documentación anterior con pruebas realizadas.*

La clave está en el archivo application.properties. Si dejamos el estado de generate-ddl en create, cada vez que ejecutamos la api la tabla se vuelve a crear, sustituyendo la tabla anterior, si la hubiera, por una nueva y por lo tanto se borran los datos. Para que esto no pase, y se actualicen los datos, tenemos que dejar en generate-ddl en update.

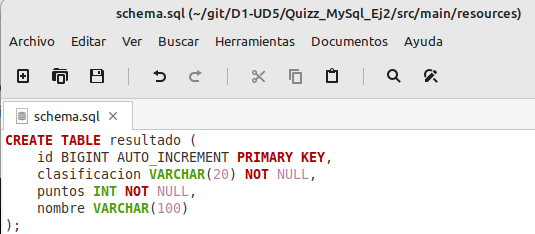
Con create, la tabla se crea automáticamente.

**Ver la creación mediante archivo schema.sql y la carga inicial de datos de las distintas formas posibles: archivo data o import.sql si no existen.**

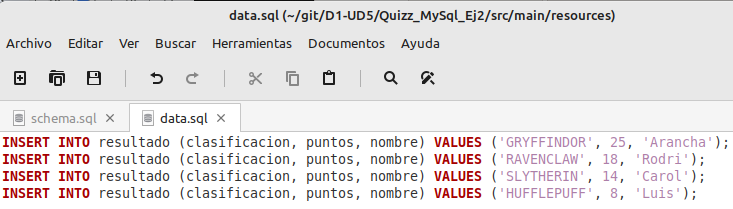
Para hacer estas pruebas voy a borrar la tabla.



Y en mi proyecto, en la carpeta resources voy a crear archivo *schema.sql* con las líneas de sql para crear la tabla:



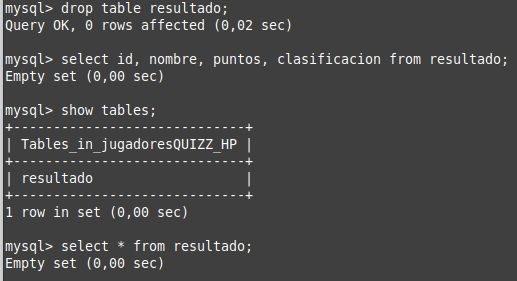
Y para cargar los datos iniciales, lo voy a hacer desde *data.sql*, pues este tipo de archivo genera las las consultas después de crear el esquema de la bbdd, y si lo hiciera desde *imports.sql*, ejecutaría las consultas antes de crear el esquema:



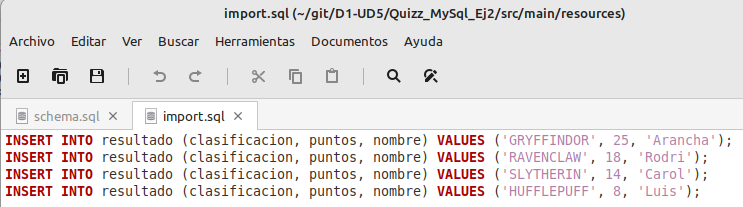
En properties, en la linea de hibernate.ddl-auto pongo el estado en none:



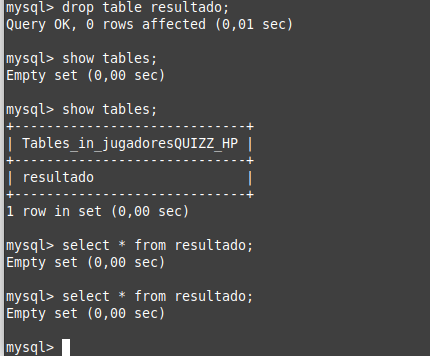
Y ejecutamos la api, se crea la tabla pero no se importan los datos:



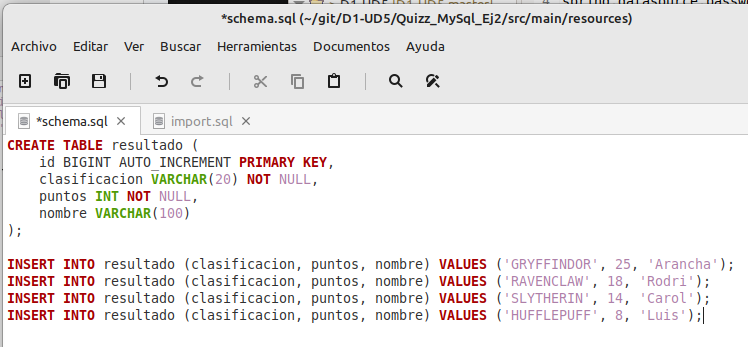
Voy a probar a cargar los datos con el archivo import.sql en lugar de data:



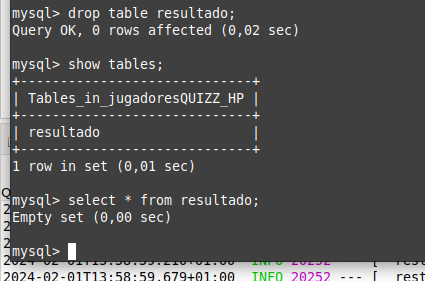
Y vuelvo a empezar, borro la tabla de bbdd, detengo la api y la vuelvo a ejecutar. Y tampoco se cargan los datos, la tabla está vacía:



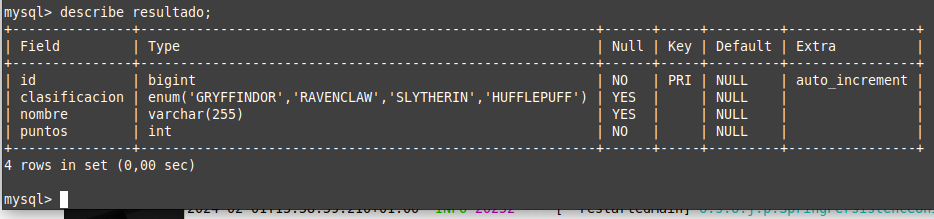
Como solución, voy a poner los imports en el mismo schema.sql, seguidamente de la creación de la tabla:



Y de nuevo, detengo la api, borro tabla de la bbdd y ejecuto api, y tampoco se añaden los datos:



Como verificación extra, accedo a la info de la tabla y se puede ver que la tabla se crea correctamente, con los tipos de datos correctos:

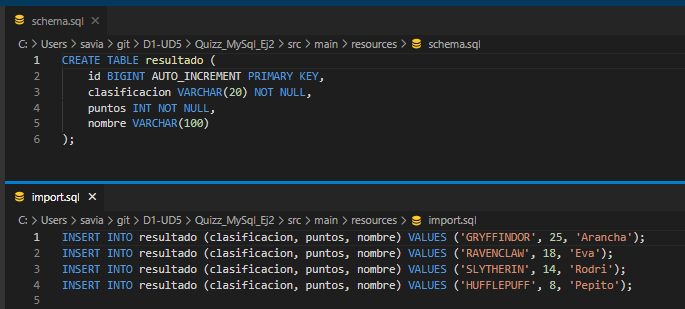


Pero desde casa, al crear la bbdd desde ahí, pruebo algo diferente.

En application.properties la línea generate-ddl pongo el estado en **false** , pues esta línea le indica a Spring Boot que debe generar el esquema de la base de datos (DDL) en función de las entidades JPA detectadas en la aplicación. Cuando se establece en true, Spring Boot generará automáticamente el DDL necesario para crear las tablas de la base de datos en función de las clases de entidad JPA. Y si se establece en false, se tendrá que crear el esquema manualmente.



Por lo que vuelvo a la idea inicial, en el archivo **schema.sql** pongo las líneas sql para crear la tabla y en el archivo **import.sql** los inserts:



Ejecuto la api y compruebo en MySql que se genera tanto la tabla como los valores:

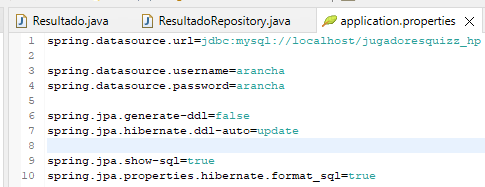


* **Activar y documentar la depuración para ver las consultas SQL que está generando el ORM automáticamente:  
  spring.jpa.show-sql=true  
  spring.jpa.properties.hibernate.format\_sql=true**

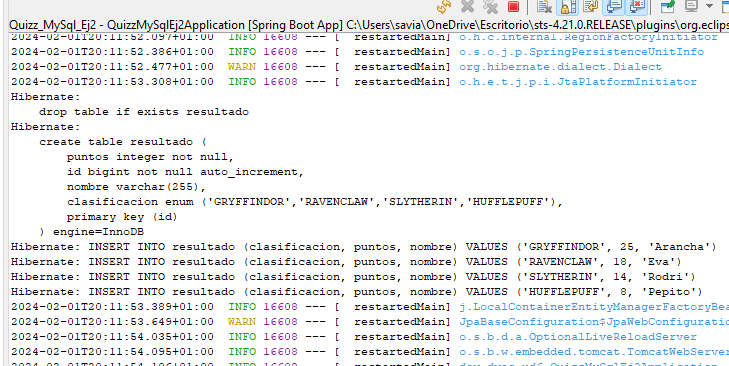
Estas líneas de configuración se utilizan para controlar la visualización y el formato de las consultas SQL generadas por Hibernate.

1. **spring.jpa.show-sql=true**: indica a Spring Boot que muestre las consultas SQL generadas por Hibernate en la consola de registro. Esto puede ser útil para depurar y comprender qué consultas se están ejecutando en la base de datos.
2. **spring.jpa.properties.hibernate.format\_sql=true**: le indica a Hibernate que formatee las consultas SQL generadas para que sean más legibles cuando se imprimen en la consola de registro.. Esto es especialmente útil cuando se trabaja con consultas SQL complejas o largas.

Añado estas líneas a mi application.properties:



Ejecuto la api (tras borrar la tabla nuevamente), y vemos el resultado en consola:

****

En esta línea, Hibernate está generando la consulta SQL para crear la tabla resultado con sus columnas. La columna clasificacion está definida como un tipo enum con los valores 'GRYFFINDOR', 'RAVENCLAW', 'SLYTHERIN', y 'HUFFLEPUFF'. Esta línea por lo tanto indica que esta consulta se ha mapeado correctamente.

Y las siguientes líneas indican que los insert con los valores correspondientes se han volcado correctamente.

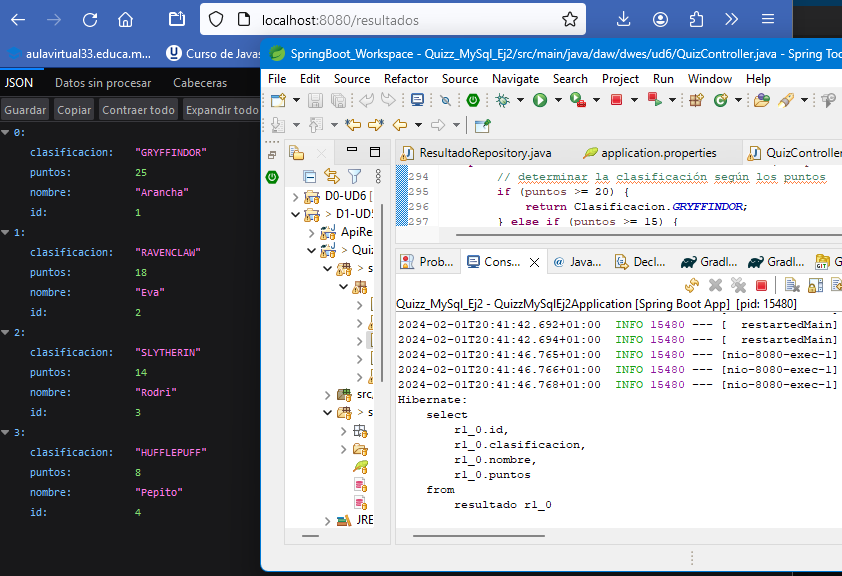
Es decir, vemos las consultas de los dos archivos sql que se están ejecutando, de esta manera vemos más fácilmente que estos archivos sql se ejecutan correctamente.

**\*\* Para hacer más pruebas voy a generar consultas en el controlador:**

**Método obtener todos los resultados con .findAll():** de tipo List.

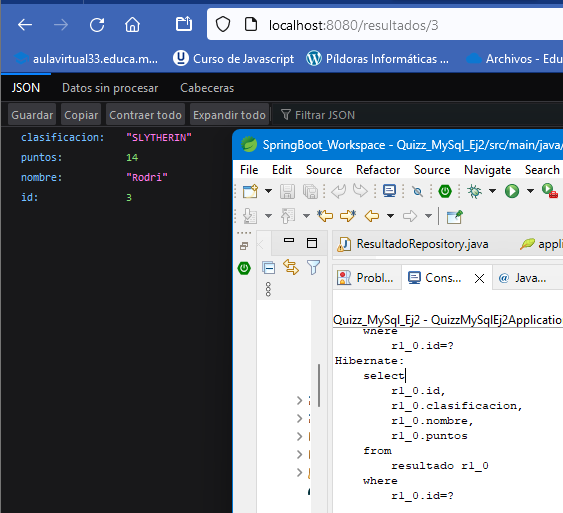
****

Ejecuto en navegador y veo resultado en navegador y consola:

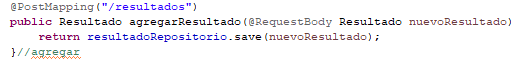


**Método GET buscar un resultado con .findById():** tipo Optional.

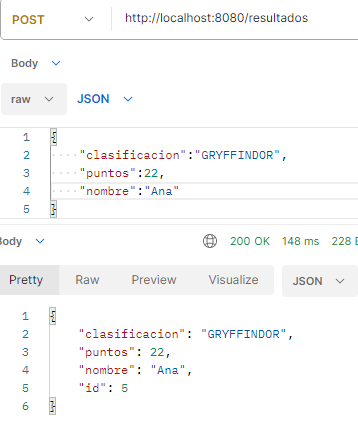
****

****

**Método POST ageregar nuevo resultado con .save().** De tipo Resultado.

****

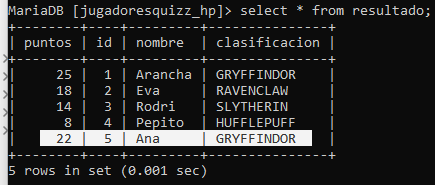
Creo resultado con postman:



Y veo resultado en consola de STS:



Y en mySql:



**4.- Para el excelente (dos de los tres puntos, además del bien)**

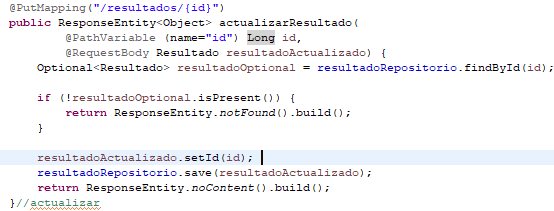
* **Implementar borrado y/o modificación de clasificaciones. Filtrado.**

**Método PUT modificar resultado con .save() y .findById().** De tipo Response Entity para manejar la respuesta.

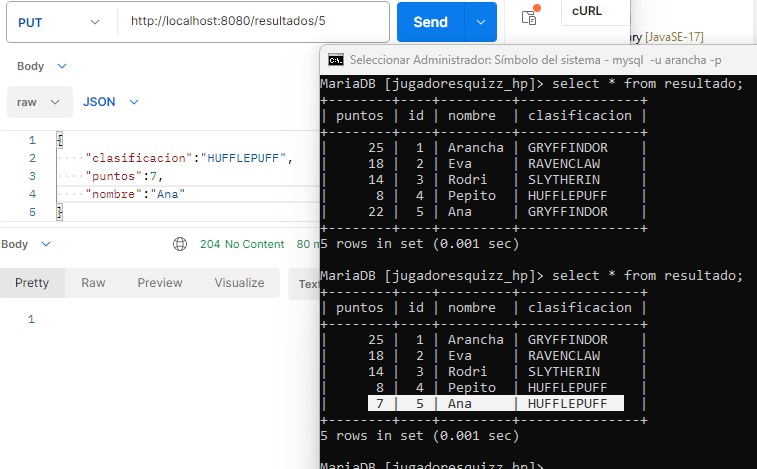
**-**Primero con .findById(id) le asígno el elemento correspondiente a variable resultadoOptional,

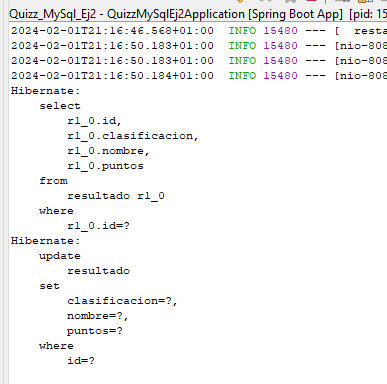
**-**Segundo, si no existe (si no .isPresent) return notFound.

**-**Tercero, si si existe, con .save guardo el resultado en el respositorio y reurn noContent.



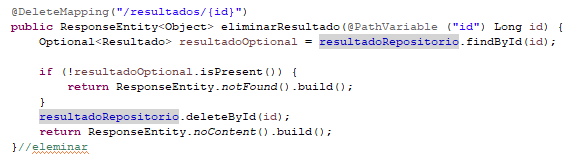
En postman modifico el elemento con id 5, compruebo en MySql que se modifica el elemento y en consola de STS se ve la consulta con formato, pero no se muestran los valores:



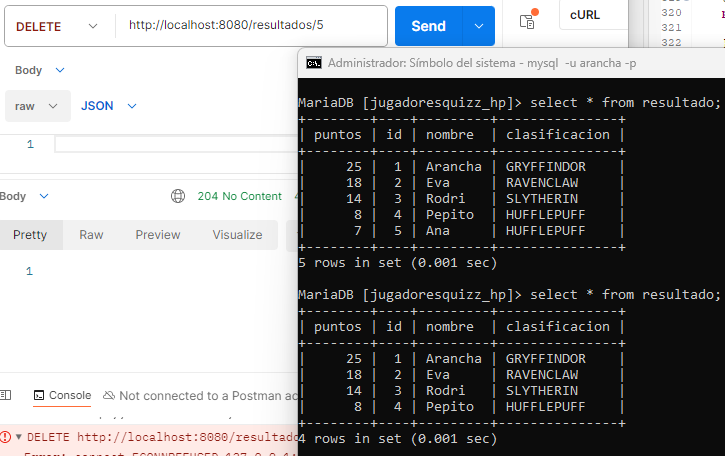


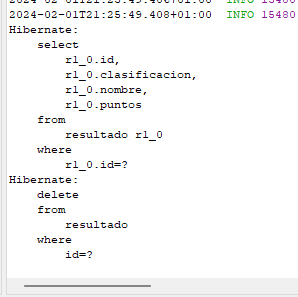
**Método delete borrar resultado con .deleteById() y .findById():** de tipo ResponseEntity para manejar la respuesta.

Los pasos seguidos son los mismos que en el método put, pero si el elemento existe, lo borramos de la bbdd con .deleteById(id).

****

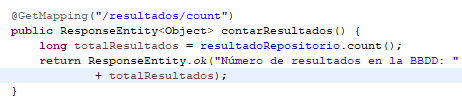
En postman borro el elemento con id 5, nombre Ana, compruebo en MySql que se elimina y en la consola STS la consulta con formato, pero no se ven los valores:

****

****

**Método contar resultados con .count():** de tipo ResponseEntity

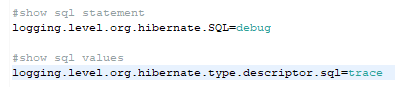
Utilizo el método count() para obtener el número total de resultados en el repositorio y se lo asignamos a variable de tipo long, y lo devolvemos como parte de la respuesta.

****

****

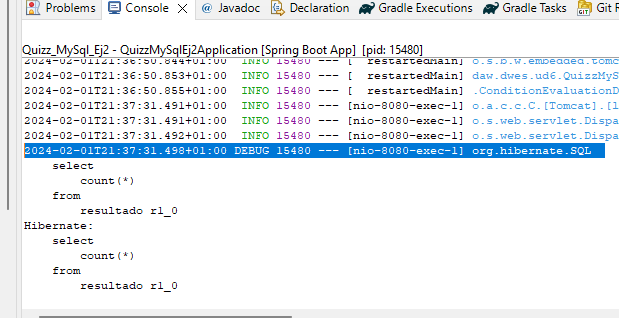
* **Usar el monitor de qué consultas se hacen en tiempo real a la base de datos:** [**https://mkyong.com/spring-boot/spring-boot-show-hibernate-sql-query/**](https://mkyong.com/spring-boot/spring-boot-show-hibernate-sql-query/)

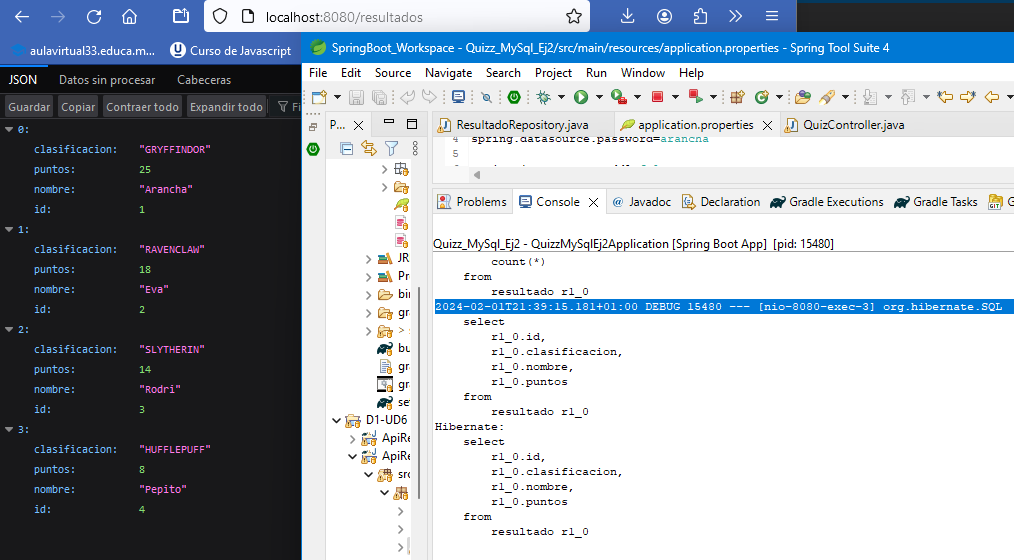
Añado estas líneas en application.properties:

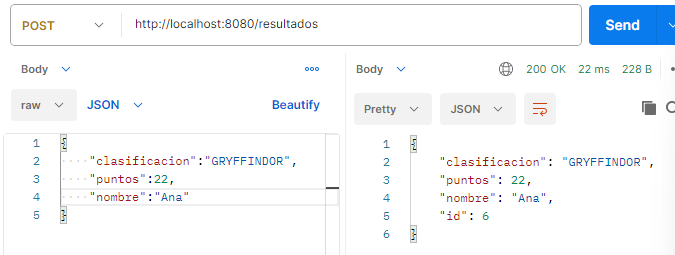


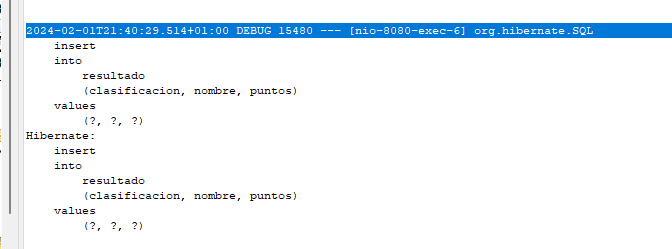
Y al probar a hacer consultas, se puede ver una lína de debug que aparece el tiempo exacto en el que se ha realizado la consulta.

*Pruebo con varias consultas diferentes:*

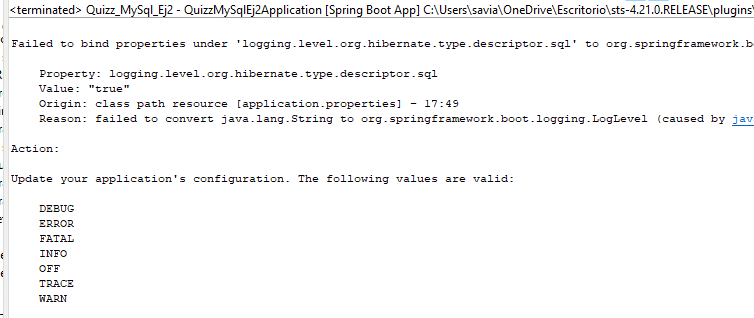
****

****

****

****

La línea *logging.level.org.hibernate.type.descriptor.sql=trace,* creo que es la que hace que se muestren los valores. Cambio el estado *trace* a *true*, y sale error en la consola, pero me da las opciones siguientes:

****

Pruebo con el estado *debug* y tampoco se muestran los valores, siguen mostrándose interrogaciones:  


**CONCLUSIONES**

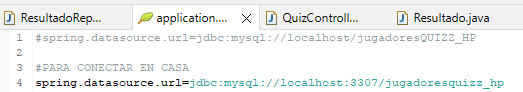
He dejado los dos proyectos usados para esta práctica en mi repositorio de github:

<https://github.com/AranchaC/D1-UD5.git>

### -[ApiRest\_MySql\_Ej1](https://github.com/AranchaC/D1-UD5/tree/master/ApiRest_MySql_Ej1): para el ejercicio 1.

### -[Quizz\_MySql\_Ej2](https://github.com/AranchaC/D1-UD5/tree/master/Quizz_MySql_Ej2): para los ejercicios 2 , 3 y 4.

Me ha gustado mucho hacer esta práctica, sobretodo el hecho de adaptar Apis que ya teníamos hechas pues hemos podido ver que hay muchas maneras de realizar las Apis y sus funciones. Como bbdd me gusta, el probar a hacer consultas con mi propia api y sus valores es muy interesante.

He aprendido mucho sobre las funciones del repositorio en spring gracias a sus dependencias.  
También he aprendido mucho a ver los logs de la consola de STS y manejar los mensajes de error, sobre todo en casa al conectar con mysql de allí. Pues como en casa uso Windows y tengo mysql con xampp, lo uso en el puerto 3307 y como no es puerto por defecto, tuve que especificar en applicatio.propertiies, en la ruta de conexión con la bbdd el puerto, de la siguiete manera:  
  
Pero llegar a esa concluisión me costó mucho, muchas consultas con chatGpt, muchas consultas con los archivos de configuración, tanto de sts como de xamp, y consultas con los logs tanto de la consola de sts como de mysql.

También me ha gustado ver las diferentes formas de crear la tabla, de forma automática y manual con archivo schema.sql.

La forma automática está muy bien, pues se puede personalizar/especificar el nombre y tipo de datos a mapear a la bbdd.  
  
Sobre otros problemas encontrados, los he ido describiendo en el documento pues lo he ido redactando sobre la marcha.