BBDD y POO – Teoría (II)

En este documento vamos a tratar de cómo se trabaja con una Base de Datos desde Java. Y donde digo Java, me refiero a casi cualquier otro lenguaje de programación orientado a objetos (POO).

Normalmente, en los proyectos se hace primero el **diseño de la Base de Datos**. Esto es porque necesitamos conocer cómo se va a guardar la información en nuestro sistema antes de decidir cómo vamos a trabajar con ella. Cómo se hace este diseño de la Base de Datos se estudia en otra asignatura, pero requiere hacer las tablas, pasarlas a tercera forma normal, etcétera.

Pero, y aquí está la cosa, nosotros programamos nuestras aplicaciones accediendo a esas Bases de Datos, y nosotros sólo sabemos manipular clases y objetos. Y las Bases de Datos no tienen clases ni objetos.

Es aquí cuando entramos en la segunda fase de diseño: el **diseño de Clases**. Que también se aprende en otra asignatura (Entornos de Desarrollo). Y que, explicado brevemente, consiste en trasladar el diseño de Base de Datos a Clases (de Java). Y más cosas.

Pero esto no nos interesa. Vamos a centrarnos en la mecánica. En cómo se hace para diseñar unos POJOs para la Base de Datos correctamente.

Qué es un POJO y para qué se usan

Un **POJO** (*Plain Old Java Object*) es un término en Java que se refiere a un objeto simple y no especializado. Es una clase suelta que no está relacionada con el resto del código, ni depende de él, ni nada. Contiene unos pocos atributos y unos pocos getters y setters, y ya.

Los POJOs se usan mucho en Java para representar modelos de datos. Es decir, para describir cómo son las tablas de la Base de Datos. Si tú tienes una tabla de Alumnos, tendrás un POJO de Alumno. Cuando accedas a la tabla para leer tres Alumnos, Java generará tres POJOs Alumnos como respuesta. Cuando quieras insertar un POJO, insertarás una línea en la tabla.

Porque, tu programa no entiende de Bases de Datos. Sólo de objetos.

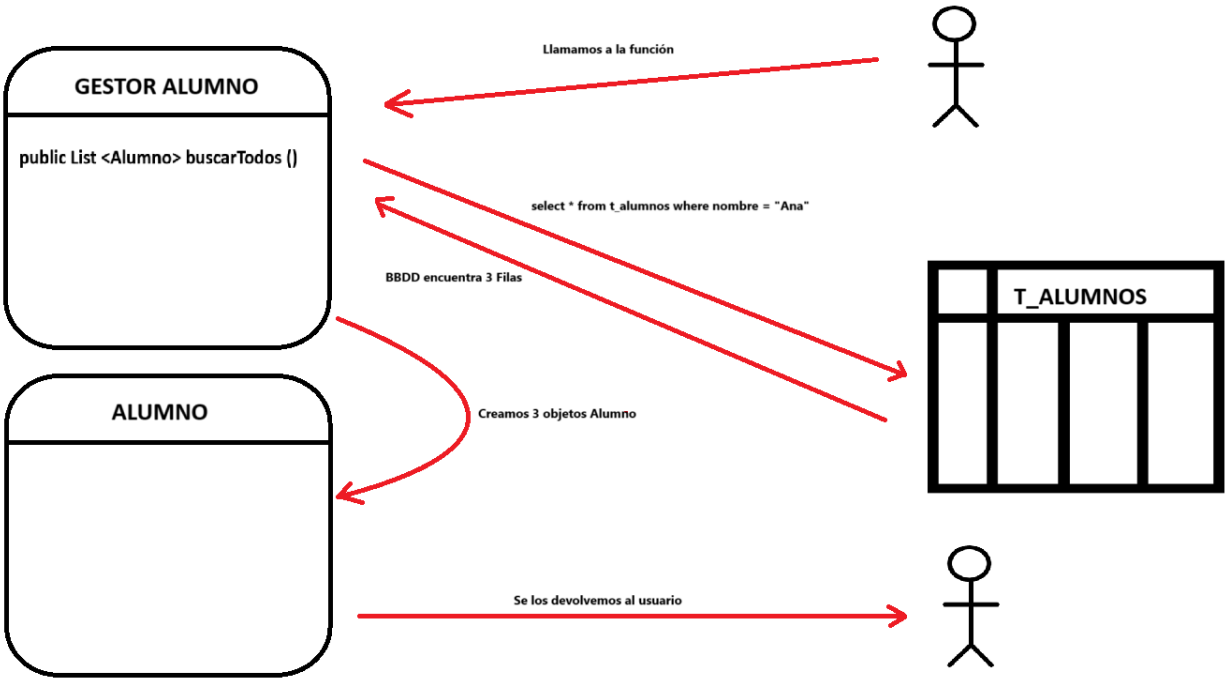
Un ejemplo visual. Supongamos que hacemos esta consulta a Base de Datos:

Consulta: Select * from t_alumnos where nombre = “Ana”

La forma correcta de hacer esto (de libro) es la siguiente:

1- Hago un POJO que me describa la tabla t_alumnos. Lo llamo Alumno.
2- Hago una clase **GestorAlumno** donde meto la función de buscar.
    a. La función la llamamos buscarTodos ()
    b. La función devuelve una lista de Alumnos
    c. La función no necesita recibir parámetros de entrada
3- Meto en la **buscarTodos ()** el código necesario para que funcione la consulta.

Esto ya lo hemos visto en los apuntes anteriores. Lo que sucede es lo siguiente:



Por lo tanto, siempre que **busquemos algo** en la base de Datos, lo correcto es devolver uno o varios POJOS. Siempre que **borremos, insertemos o modifiquemos algo**, lo correcto es pasarle a la función uno o varios POJOS.

¿Siempre?

No. Esta es la norma de libro, otro es el mundo real. Pero en general, lo correcto es usar POJOs siempre. No tiene sentido devolver sólo el nombre y el apellido cuando puedes devolver un POJO (una fila de la tabla) entera. A la Base de Datos le cuesta lo mismo devolverte una columna que una fina entera, luego no hay problemas de rendimiento.

Los métodos de los gestores

Lo normal es que los métodos de acceso a Bases de Datos se metan separados en clases a las que yo llamo **GestorAlumno**, **GestorNotas**, etc. Cada gestor tendrá las funciones para acceder a una tabla en concreto.

¿Y si un método accede a dos tablas? ¿Y si uso una join? ¿Y si …?

Pues en este caso, lo metes donde creas que está mejor. No te vuelvas loco.

En general, los métodos que siempre suelen aparecer ahora que estáis aprendiendo, son los siguientes (para GestorAlumno, por ejemplo):

Retorna	Nombre	Parámetros
Alumno	getAlumnoById	Id del Alumno
ArrayList <Alumno>	getAllAlumnos	Nada
Nada	insertAlumno	Alumno
Nada	updateAlumno	Alumno
Nada	deleteAlumno	Alumno

Si luego tenemos que crear nuestros propios métodos, seguimos con esta idea en general. Supongamos que tenemos que hacer un método que retorna todos los alumnos de nombre Ana y Apellido Pérez. ¿Tiene sentido pasar un Alumno? No. Pues vale.

Retorna	Nombre	Parámetros
ArrayList <Alumno>	getAlumnos	Nombre y Apellido a buscar, tipo String

¿Y si queremos insertar de una vez varios Alumnos?

Retorna	Nombre	Parámetros
Nada	insertAlumno	ArrayList <Alumno>

¿Y si queremos borrar todos los alumnos que se son mayores de 25 años?

Retorna	Nombre	Parámetros
Nada	deleteAlumno	Edad, de tipo int

Tú eres el que define cómo son los métodos de acceso a Base de Datos. Tú decides lo que necesitas. Pero de nuevo, en general, mejor trabajas con objetos POJO que con datos sueltos.

Diseñando POJOs

Los POJOs de una tabla se hacen casi sin pensar, siempre y cuando el esquema de Base de Datos esté bien y normalizado. Simplemente haces un POJO con **los mismos atributos** que columnas haya en la tabla, le pones getters y setters, hashCode, equals y toString. Y ya está.

En cuanto a los tipos de datos entre Java y MySQL, hay que sabérselos. Porque sí, es posible leer un BIGINT de BBDD y convertirlo en un String en la misma función, pero NO tiene sentido, es mala idea, no se hace así, te vas a confundir en el futuro.

Te pongo una tabla con las equivalencias más comunes, peor hay más.

MySQL	Java
INT	int
BIGINT	long
VARCHAR	String
TEXT	String
TIME	java.sql.date
BIGBLOB	byte [] (para guardar imágenes)

Esto es importante: si intentas leer un TIME de Base de datos y lo intentas meter en un atributo que NO sea java.sql.date, te va a dar un error.

¿Y tengo que hacer tantos POJOs como tablas?

Si.

¿Seguro?

En realidad, depende. Tú lo que tienes que conseguir es ser capaz de gestionar cada fila de una tabla con un solo objeto. Para eso, hay que conocer el mecanismo de Herencia de Java, que veremos más adelante.

Relaciones entre las tablas

Como ya sabes, las relaciones entre las tablas **también aportan información**. Una relación entre Alumno y Nota podría significar que “este Alumno tiene estas Notas”.

Esto también se refleja en los POJOS. La forma de hacerlo es incluso más sencilla que en las Bases de Datos, siempre y cuando el esquema de Base de Datos esté bien hecho y sobre todo, bien y normalizado.

Las normas son simples. Si tú tienes en el otro lado de la relación un algo, tienes que poner en tu lado una cosa. Punto. No hace falta ni pensar. Supongamos que tenemos dos tablas A y B con una relación entre sí. Si estamos en A y…

En el otro lado (B) tengo…	Entonces en A pongo…
1	B
0..1	B
0…N	ArrayList <B>
1…N	ArrayList <B>
3…7	ArrayList <B>

Luego, no te olvides de poner la relación inversa entre B y A.

¿Para que se usan estas variables nuevas? Bueno pues, para cuando tienes que devolver por ejemplo un alumno llamado Juan con todas sus notas.

En este caso, hacemos una función que haga lo siguiente:

1- Buscamos el Alumno Juan.
2- Creamos un POJO Alumno y le ponemos nombre = Juan, Apellido = …
3- Buscamos las Notas de Juan.
4- Creamos tantos POJOs Nota como sean necesarios, y los metemos en un ArrayList <Nota>.
5- Metemos el ArrayList <Nota> en el POJO Alumno.
6- Devolvemos Alumno.

De esta manera tenemos un Alumno CON todas sus Notas.

¿Estoy obligado a meter cosas en estas relaciones?

NO. Sólo se usan cuando lo necesites. Si solamente quieres devolver Alumno sin Notas, pues no metes las Notas. Obviamente.

¿Y al meter una Nota en Alumno, tengo que meter en Alumno en la Nota?

NO. Porque esa relación no nos interesa usarla. Sólo meteremos un Alumno en la Nota cuando queramos devolver una Nota con su Alumno.