



DIPLOMADO

Desarrollo de sistemas con tecnología Java

Módulo 04

Extras Hibernate

Mtro. Alfonso Gregorio Rivero Duarte



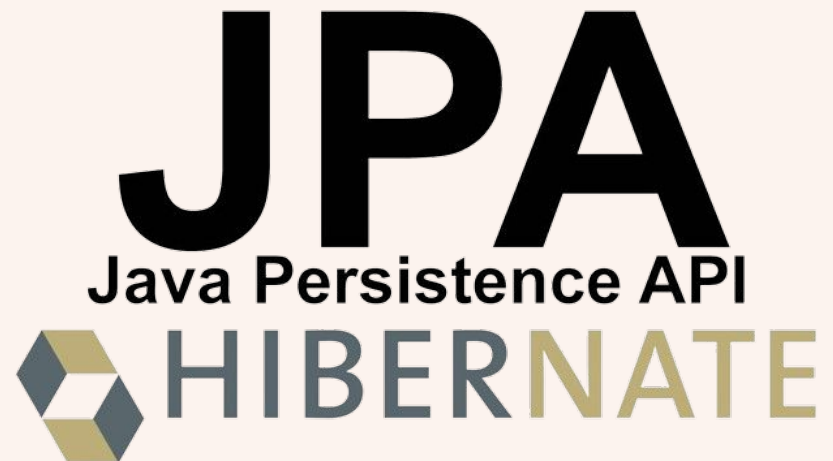
Temario

1. Extras Hibernate

- Maven
- Modelo DAO
- Carga Lazy
- Named Querys
- Caché



1. Extras Hibernate



1.1 Maven

¿Qué es Maven?

Apache Maven es una potente herramienta de gestión de proyectos que se utiliza para gestión de dependencias, como herramienta de compilación e incluso como herramienta de documentación. Es de código abierto y gratuita.

Aunque se puede utilizar con diversos lenguajes (C#, Ruby, Scala...) se usa principalmente en proyectos Java, donde es muy común ya que está escrita en este lenguaje. De hecho, frameworks Java como Spring y Spring Boot la utilizan por defecto, por lo que conviene conocerla si trabajas en la plataforma Java y, desde luego, con Spring.

¿Qué es Maven?

Al contrario que otras herramientas anteriores y más limitadas como Apache Ant (también muy popular y obsoleta), Maven utiliza convenciones sobre dónde colocar ciertos archivos para el proceso de build de un proyecto, por lo que solo debemos establecer las excepciones y por lo tanto simplifica mucho el trabajo.

Es una herramienta declarativa. Es decir, todo lo que definamos (dependencias en módulos y componentes externos, procesos, orden de compilación, plugins del propio Maven...) se almacena en un archivo XML que Maven lee a la hora de funcionar. Otras alternativas, como Gradle no utilizan archivos XML, sino de otro tipo, pero usan los mismos conceptos de Maven.

Con Maven se puede...

- Gestionar las dependencias del proyecto, para descargar e instalar módulos, paquetes y herramientas que sean necesarios para el mismo.
- Compilar el código fuente de la aplicación de manera automática.
- Empaquetar el código en archivos .jar o .zip.
- Instalar los paquetes en un repositorio (local, remoto o en el central de la empresa)
- Generar documentación a partir del código fuente.
- Gestionar las distintas fases del ciclo de vida de las build: validación, generación de código fuente, procesamiento, generación de recursos, compilación, ejecución de test ...

Además, la mayor parte de los entornos de desarrollo y editores de Java disponen de plugins específicos o soporte directo de Maven para facilitarnos el trabajo con esta, puesto que se ha convertido en una herramienta casi universal.

¿Qué es el archivo pom.xml?

La unidad básica de trabajo en Maven es el llamado Modelo de Objetos de Proyecto conocido simplemente como POM (de sus siglas en inglés: Project Object Model).

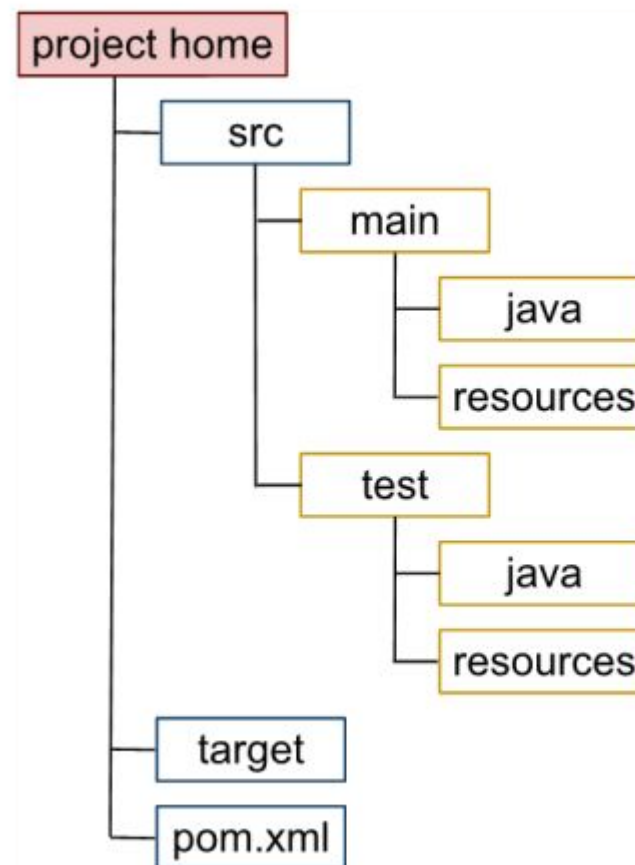
Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, etc...

¿Qué es el archivo pom.xml?

El hecho de utilizar un archivo XML revela su edad. Maven se creó en 2002, cuando XML era lo más. Si se hubiese creado unos pocos años después seguramente tendríamos un pom.json y si hubiese sido más reciente un pom.yml. Modas que vienen y van. No obstante, el formato XML, aunque engorroso, es muy útil a la hora de definir con mucho detalle cómo debe ser cada propiedad y, también, para poder comprobarlas.

¿Qué es el archivo pom.xml?

Incluso, aunque nuestro proyecto, que usa Maven, tenga un archivo pom.xml sin opciones propias, prácticamente vacío, estará usando el modelo de objetos para definir los valores por defecto del mismo. Por ejemplo, por defecto, el directorio donde está el código fuente es `src/main/java`, donde se compila el proyecto es `target` y donde ubicamos los test unitarios es en `src/main/test`, etc... Al pom.xml global, con los valores predeterminados se le llama Súper POM.



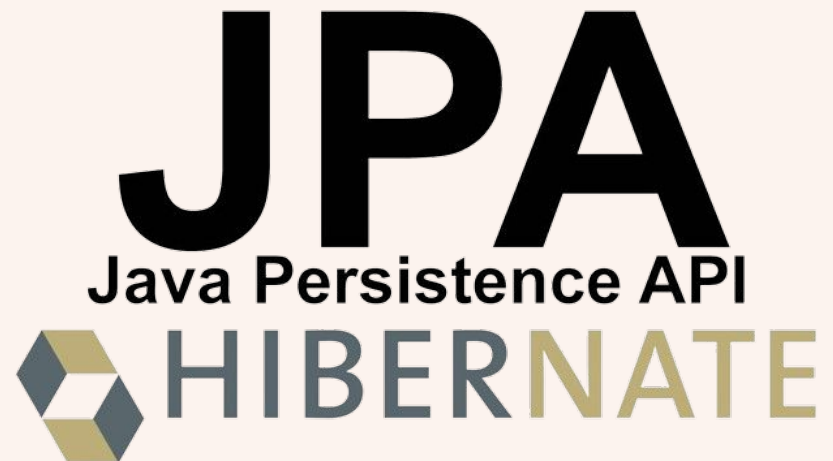
¿Qué es el archivo pom.xml?

Como puedes ver incluye una serie de dependencias: la primera es el starter de Spring Boot para generar una aplicación Web, la segunda es el starter que trae todas las dependencias para poder ejecutar test (por ejemplo JUnit o Mockito) y que excluye todo lo relativo al motor antiguo de JUnit 4. La final añade el soporte para JSP en el servidor Apache Tomcat.

Los 4 primeros nodos son los únicos obligatorios y definen respectivamente el modelo de objetos que utilizará Maven, el identificador del grupo del proyecto, el id del proyecto (artifact) y su versión. Esto es lo mínimo que debe incluir el archivo si utilizamos Maven y la mayor parte de los editores y herramientas nos lo crearán automáticamente al crear el proyecto.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>es.campusmvp</groupId>
  <artifactId>demo</artifactId>
  <version>1.0</version>
  <!-- Dependencias -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
      <scope>compile</scope>
    </dependency>
  </dependencies>
</project>
```

1. Extras Hibernate



1.2 Modelo DAO

DAO

El patrón Data Access Object (DAO) pretende principalmente independizar la aplicación de la forma de acceder a la base de datos, o cualquier otro tipo de repositorio de datos.

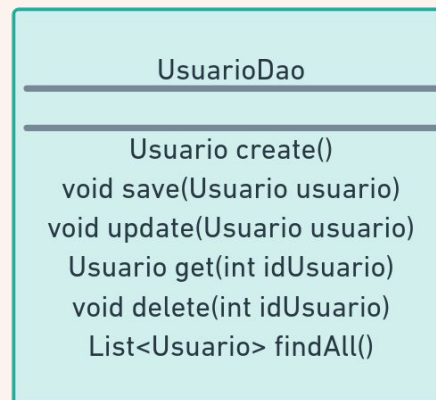
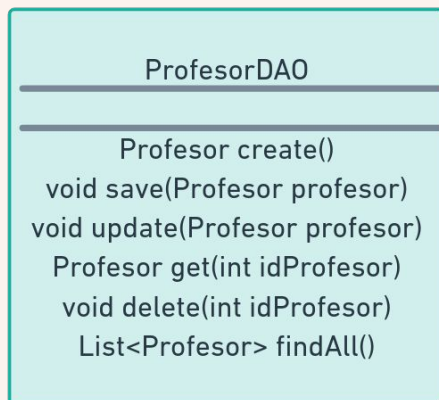
Centraliza el código relativo al acceso al repositorio de datos en las clases llamadas DAO.

Fuera de las clases DAO **no debe** haber ningún tipo de código que acceda al repositorio de datos.

DAO

Las ventajas de usar el patrón DAO son las siguientes:

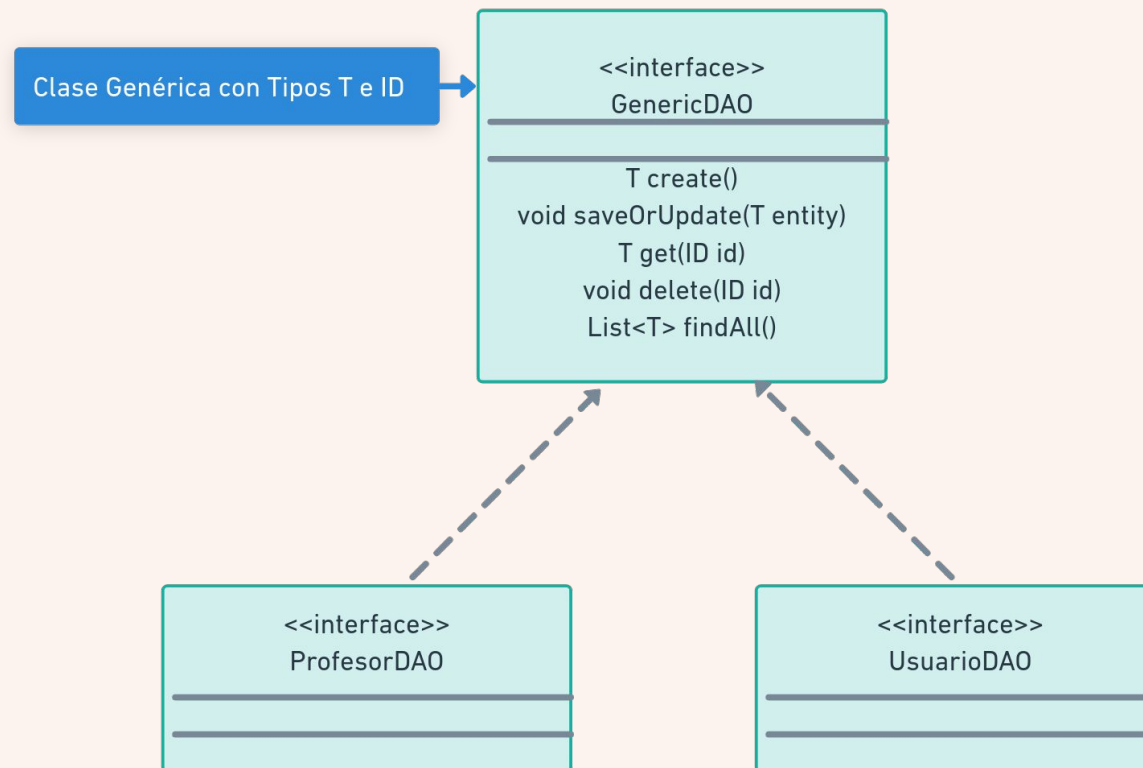
- Modificar el API de acceso
- Modificar el repositorio de datos



El mayor problema del patrón DAO es la repetición de código para cada una de las entidades.

DAO

La solución es crear una **interface genérica** DAO de la cual saldrán varias interfaces DAO hijas y lo que permitirá varias implementaciones.



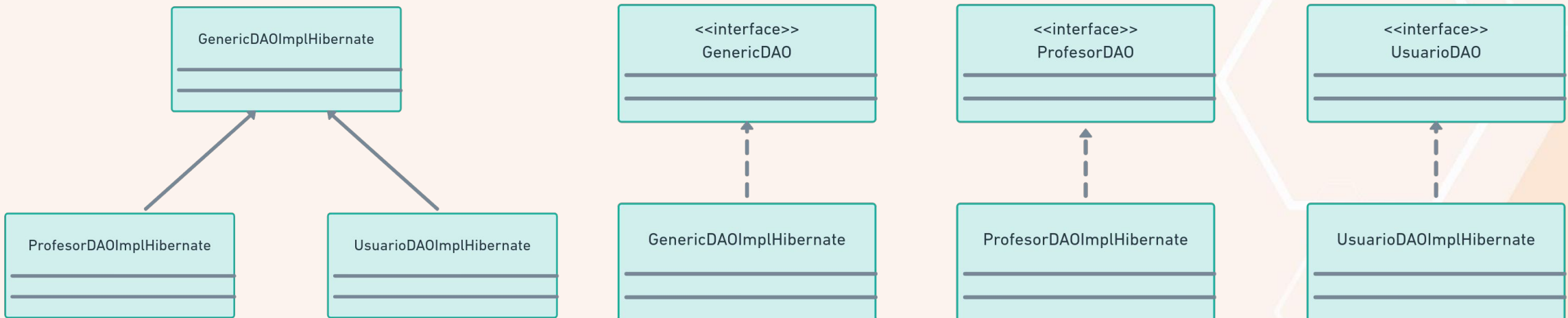
DAO

La definición de los métodos quedaría:

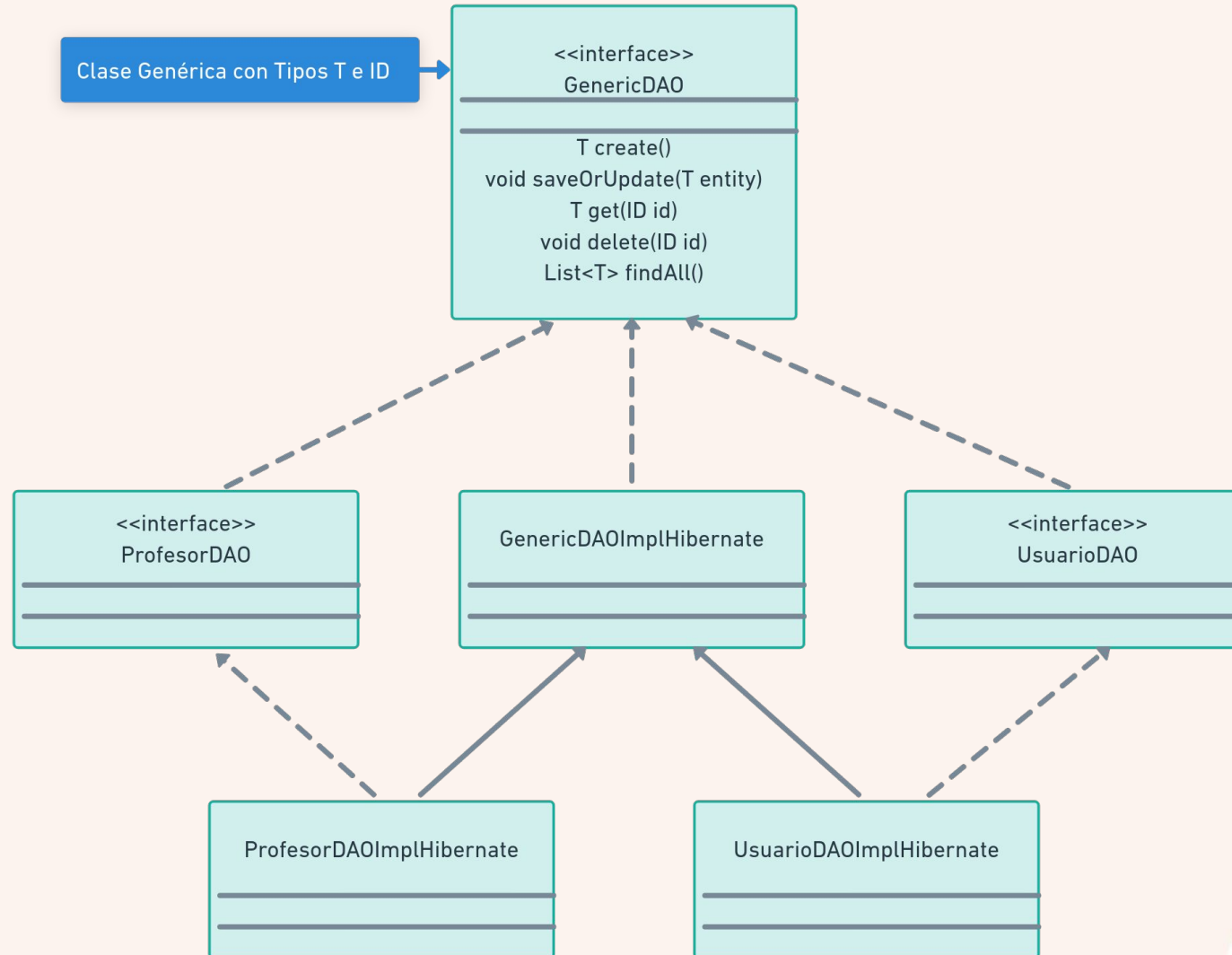
- **T create():** Crea un nuevo objeto de la entidad.
- **void saveOrUpdate(T entity):** Inserta o actualiza un objeto de una entidad en la base de datos.
- **T get(ID id):** Obtiene un objeto de una entidad desde la base de datos en función de la clave primaria.
- **void delete(ID id):** Borra un objeto de una entidad de la base de datos en función de la clave primaria.
- **List<T> findAll():** Obtiene una lista con todos los objetos de una entidad de la base de datos.

DAO

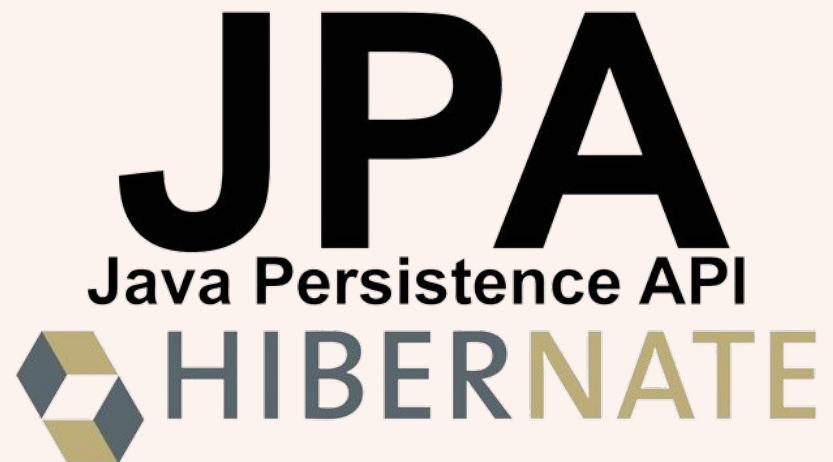
La implementación de este nuevo modelo DAO quedaría:



DAO



1. Extras Hibernate



1.3 Carga Lazy

Carga Lazy

Uno de los puntos más importantes a analizar en un software es el desempeño y optimizar al máximo las queries SQL, esto puede resultar en una ganancia de desempeño considerable.

Un uso incorrecto provoca una gran pérdida en el desempeño de una aplicación, lo que obliga a la base de datos a queries innecesarias.

El tema del **performance** siempre deberá de ser tomado en cuenta por todo tipo de programadores, y no debe dejarse de lado.

Carga Lazy

Cuando modelamos un sistema usando la orientación a objetos, creamos varias relaciones que pueden ser `@OneToOne`, `@ManyToOne`, `@OneToMany` o `@ManyToMany` y para la base de datos, cada relación es una nueva tabla a consultar.

Carga Lazy

Tipos de Inicialización de Propiedades

- **FetchType.LAZY** = Esto no carga las relaciones a menos que lo invoque a través del método getter.
 - Lazy initialization mejora el rendimiento al evitar cálculos innecesarios y reducir los requisitos de memoria.
- **FetchType.EAGER** = Esto carga todas las relaciones.
 - Eager initialization consume más memoria y la velocidad de procesamiento es lenta.

Carga Lazy

Supongamos que tenemos dos entidades, Alumno y Matrícula.
Dícese una matrícula de materias!

```
1 @Entity
2 public class Alumno {
3     @Id
4     @GeneratedValue(strategy = GenerationType.IDENTITY)
5     private Long id;
6     private String nombre;
7     @OneToOne
8     private Matricula matricula;
9     // getters y setters omitidos
10 }
```

```
1 @Entity
2 public class Matricula {
3     @Id
4     @GeneratedValue(strategy = GenerationType.IDENTITY)
5     private Long id;
6     @Column(unique = true, nullable = false)
7     private String codigo;
8     // getters y setters omitidos
9 }
```


Carga Lazy

Por estándar, cuando la relación se anota con `@OneToOne` o `@ManyToOne`, se carga en modo Eager, es decir, cuando hicimos algún tipo de búsqueda en la entidad como por ejemplo un `find(Alumno.class, 1)`, se cargará junto con la entidad, en cuyo caso JPA ejecutará una única query.

En el ejemplo anterior colocamos la anotación `@OneToOne` en el atributo Matrícula, ¿tiene sentido? No, pues generalmente un alumno tiene varias matrículas de materias, así que cambiemos la anotación de la relación de clase Alumno a `@OneToMany` y el tipo de List.

Carga Lazy

```
1 @Entity
2 public class Alumno {
3     @Id
4     @GeneratedValue(strategy = GenerationType.IDENTITY)
5     private Long id;
6     private String nombre;
7     @OneToMany
8     private List<Matricula> matriculas;
9     // getters y setters omitidos
10 }
```

Carga Lazy

Por estándar, cuando la relación está anotada con `@OneToMany` o `@ManyToMany`, se carga en modo Lazy, es decir, cuando hicimos algún tipo de búsqueda en la entidad como por ejemplo `find(Alumno.class, 1)`, no se cargará con la entidad, solo cuando ejecutemos el comando `getMatriculas()` se cargará la relación.

Por el modo Lazy, solo cargamos informaciones cuando ejecutamos un **getter** para hacer un `alumno.getMatriculas().getCodigo()` dentro de un for para obtener el código de todas las matrículas del alumno puede traer problema de desempeño a la aplicación, ya que JPA ejecutará varias queries.

Carga Lazy

```
1 for (Alumno alumno : alumnos) {  
2     for (Matricula matricula : alumno.getMatriculas()) {  
3         System.out.println(matricula.getCodigo());  
4     }  
5 }
```

El código desde dentro del for ejecutará el tamaño de la lista veces, por ejemplo, si el tamaño de la lista es N, el código se ejecutará N veces, luego la JPA ejecutará N queries en la base de datos, también podemos contar la query para cargar la entidad.

Este problema se conoce como consultas $[N + 1]$, uno de los problemas que deben evitarse al utilizar JPA e Hibernate.

Carga Lazy

Este problema se conoce como consultas [N + 1]

Además del problema mencionado anteriormente, debemos tener cuidado con `LazyInitializationException`.

Fetch

La mejor manera de resolver la `LazyInitializationException` es usar la directiva JOIN FETCH en las consultas de su entidad.

```
SELECT e FROM Empleado e JOIN FETCH e.direccion
```

@Fetch

- Con la anotación `@Fetch` la recuperación de la base de datos se realiza por JOIN fetch
- `@FetchMode.JOIN`

```
1 @OneToMany(mappedBy="tableName", cascade=CascadeType.ALL)  
2 @Column(name="id")  
3 @Fetch(FetchMode.JOIN)
```

Fetch

En JPA, la cláusula JOIN FETCH se utiliza para realizar una consulta que recupera **entidades** junto con sus **relaciones asociadas** en una **sola consulta**, en lugar de generar consultas adicionales para cargar las relaciones (lo que se conoce como carga diferida o lazy loading).

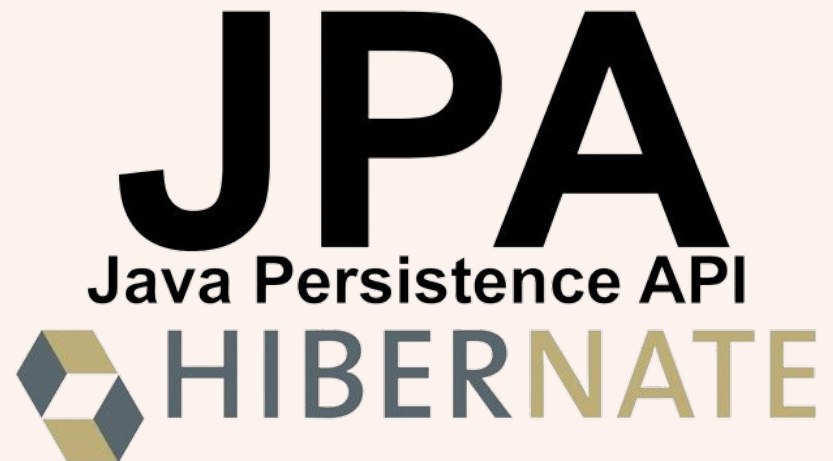
JOIN FETCH es útil para **mejorar el rendimiento de las consultas**, ya que reduce el número de consultas que se ejecutan en la base de datos al traer la información relacionada en una sola consulta. Sin embargo, no necesariamente significa que la base de datos se abra si estuviera cerrada. La apertura y cierre de la conexión a la base de datos generalmente están gestionados por el proveedor JPA (Hibernate, EclipseLink, etc.) y el contenedor de persistencia (como un servidor de aplicaciones).

Fetch

Entonces, cuanto a cerrar conexiones después de usar la cláusula JOIN FETCH, normalmente no es necesario. El manejo de las conexiones y recursos suele ser automático en JPA. Los proveedores JPA administran el ciclo de vida de las entidades y las conexiones de la base de datos. **En un entorno administrado (como una aplicación Java EE o Spring), las conexiones se abren y cierran de manera transparente para el desarrollador.**

En resumen, la cláusula JOIN FETCH en JPA no implica necesariamente la apertura o cierre manual de conexiones de base de datos. Los detalles sobre la gestión de conexiones dependen del entorno en el que estás trabajando y del proveedor JPA que estás utilizando.

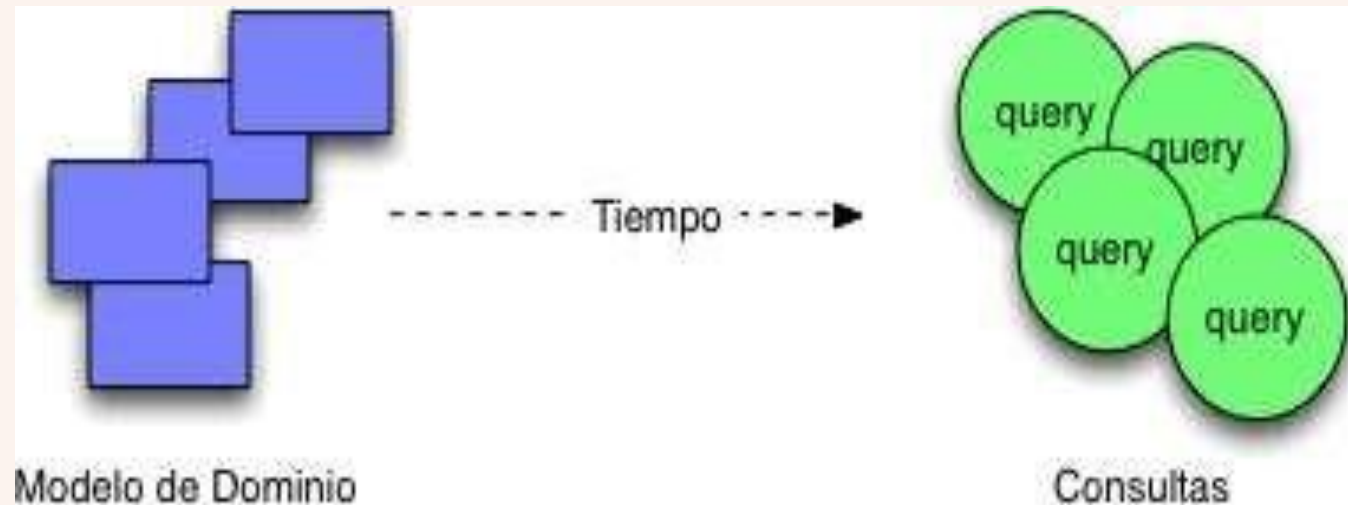
1. Extras Hibernate



1.4 Named Querys

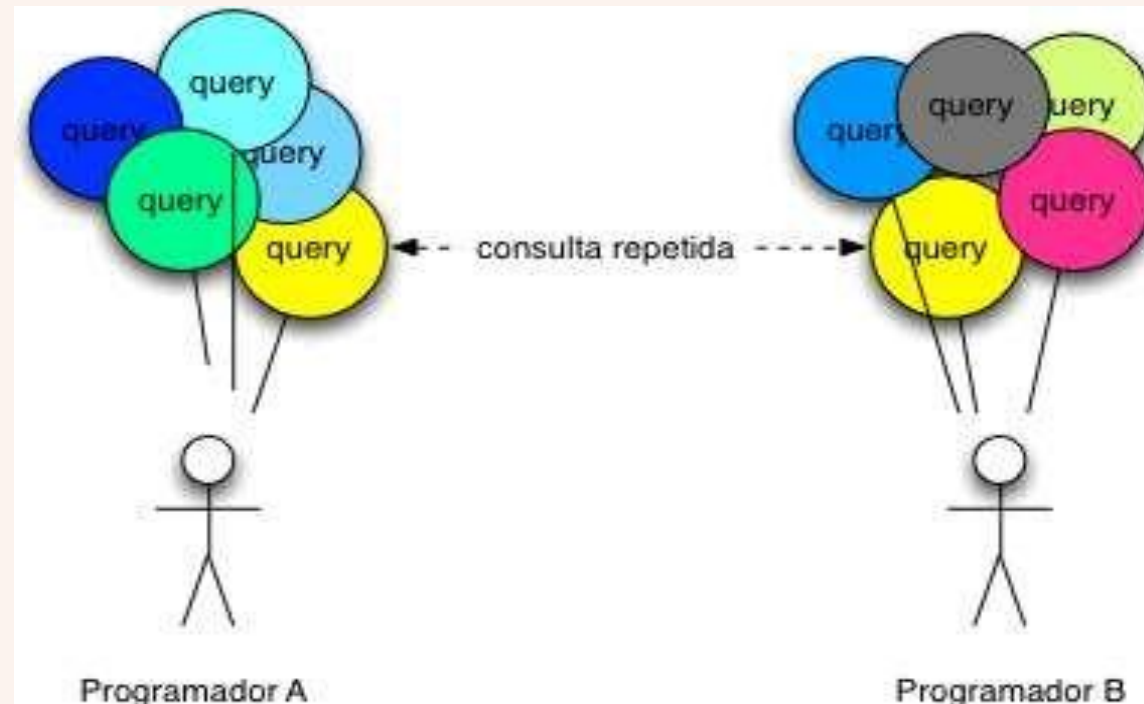
Named Querys

Al realizar el desarrollo de un proyecto una vez que se tiene definido el modelo de dominio el otro gran trabajo importante es construir las diferentes consultas a realizar contra el modelo.



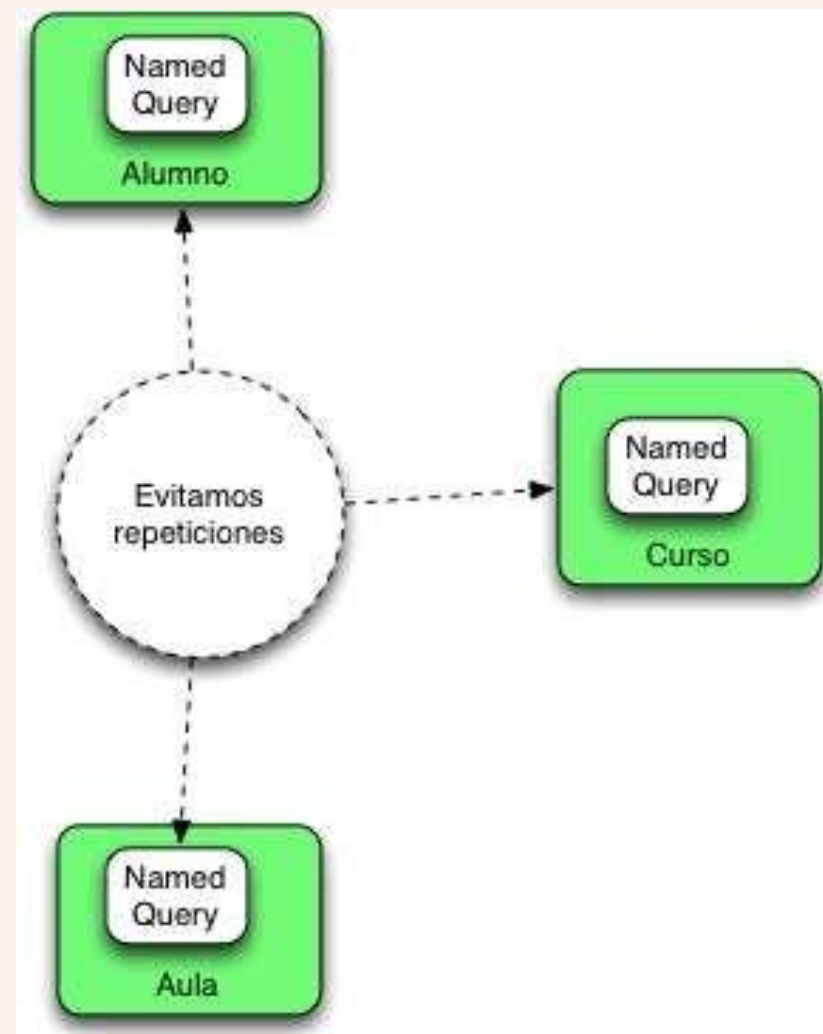
Named Querys

Uno de los problemas que nos podemos encontrar cuando construimos las consultas es que varios desarrolladores van a trabajar en el mismo proyecto construyendo consultas. Con lo cual puede que acabemos con consultas repetidas en distintas partes del código .



Named Querys

Para evitar este problema, Podemos apoyarnos en NamedQueries que nos permiten definir las consultas a nivel de clase de dominio evitando las repeticiones.



Named Querys

@NamedQuery

- Las consultas con nombre se definen junto con la entidad, utilizando la anotación `@NamedQuery`. Mejoran el rendimiento con respecto a las consultas dinámicas, ya que son procesadas una única vez.

```
1 @NamedQuery(name="salarioPorNombreDepartamento",  
2     query="SELECT e.salary " +  
3         "FROM Empleado e " +  
4         "WHERE e.departamento.nombre = :deptNombre AND " +  
5         "e.nombre = : empNombre")
```

Named Querys

@NamedQueries

- Si se necesita más de una consulta para una entidad, deben incluirse en la anotación @NamedQueries, que acepta una array de una o más anotaciones @NamedQuery

```
1 @NamedQueries({
2     @NamedQuery(name="Empleado.findAll",
3         query = "SELECT e FROM Empleado e"),
4     @NamedQuery(name="Empleado.findById",
5         query = "SELECT e FROM Empleado e WHERE e.id = :id"),
6     @NamedQuery(name="Empleado.findByName",
7         query = "SELECT e FROM Empleado e WHERE e.nombre = :nombre")
8 })
```


Named Querys

Para ejecutar la consulta hay que llamar al método **createNamedQuery** pasándole como parámetro el nombre de la consulta.

```
1 //Hibernate Named Query
2 TypedQuery query = session.getNamedQuery("findByEmployeeByName");
3 query.setParameter("name","Alfonso");
4 List<Employee> employees = query.getResultList();
```

1. Extras Hibernate



1.5 Caché

Caché

El almacenamiento en caché es un mecanismo para almacenar copias de datos o archivos de tal manera que se puedan servir rápidamente. El almacenamiento en caché está relacionado con un componente de hardware o software que almacena datos para que las solicitudes futuras de esos datos puedan ser atendidas más rápidamente.

Caché

Si estamos hablando de caché de base de datos, el almacenamiento en caché actuará como una memoria intermedia que permanece entre la aplicación y la base de datos.

El almacenamiento en caché de Hibernate actúa como una capa entre la base de datos real y su aplicación. Reduce el tiempo necesario para obtener los datos necesarios, ya que se recuperan de la memoria en lugar de acceder directamente a la base de datos. Es muy útil cuando necesita obtener el mismo tipo de datos varias veces.

Caché

Existen principalmente dos tipos de almacenamiento en caché:

- Caché de primer nivel
- Caché de segundo nivel

La caché de primer nivel está habilitada por defecto por Hibernate.

El objeto de sesión mantiene la caché de primer nivel.

Caché

Una aplicación puede tener muchas sesiones.

Los datos retenidos por un objeto de sesión no son accesibles para toda la aplicación (los datos de una sesión en particular no se comparten con otras sesiones de la aplicación).

Caché

Hibernate es una herramienta ORM que se utiliza para simplificar las operaciones de la base de datos. “Convierte objetos convertidos en relaciones (para almacenar en DB) y viceversa”.

Entonces, cuando consulta una entidad u objeto, por primera vez se recupera de la base de datos y se almacena en el caché de primer nivel (asociado con la sesión de hibernate). Si volvemos a consultar la misma entidad u objeto con el mismo objeto de sesión, se cargará desde la caché y no se ejecutará ninguna consulta SQL.

Caché

Algunos métodos útiles:

- **Session.evict ()**: para eliminar la entidad almacenada en caché.
- **Session.refresh ()**: método para actualizar la caché.
- **Session.clear ()**: método para eliminar todas las entidades del caché.

ENTORNOS DE DESARROLLO DE SOFTWARE


Son las etapas por las que avanza un software mientras se va desarrollando hasta salir a su versión final.



Prof. Alexys Lozada



Comienza a estudiar gratis en ed.team y descubre por qué en español **nadie explica mejor.**

 ed.team/cursos



PORTAFOLIO

Es el espacio donde presentas tus **trabajos** como desarrollador, diseñador, creador de contenido, etc.



Un portfolio **no es un curriculum.**



Son documentos, videos y audio, diseños, sitios web, etc **que demuestran tus conocimientos.**



Tener un portfolio **te da ventaja** frente a los que solo presentan curriculums.



Es muy valorado por las **empresas.**

 **Álvaro Felipe**
CEO EDteam



No te conviertas en **un coleccionista de certificados.** Aprende y aplica tus conocimientos en proyectos.



CERTIFICADO

Solo es un papel impreso o digital que muestra que has terminado o asistido a un curso.



No garantiza que hayas aprendido.



Solo es un efecto placebo.



Mira los certificados **como motivadores.**



Ningún certificado de plataformas de educación online es válido.

Aprende en EDteam con proyectos reales y arma tu portfolio:



ed.team/cursos



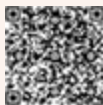
Contacto

Mtro. Alfonso Gregorio Rivero Duarte
Senior Data Manager - CBRE

devil861109@gmail.com

Tels: (+52) 55 289970 69

Redes sociales:



<https://www.linkedin.com/in/alfonso-gregorio-rivero-duarte-139a9225/>