

Arquitecturas de Software para Aplicaciones Empresariales

Detalles de las tablas, ligados y operaciones en cascada en Spring



PROGRAMA DE INGENIERIA DE SISTEMAS

Ing. Daniel Eduardo Paz Perafán (danielp@Unicauca.edu.co)

Ing. Pablo A. Magé (pmage@Unicauca.edu.co)

La anotación de *@Column*

Al igual que la anotación *@Table* , podemos usar la anotación *@Column* para mencionar los detalles de una columna en la tabla. La anotación *@Column* tiene muchos elementos, como *nombre*, *longitud*, *si permite almacenar valores nulos* y *si el valor del campo es único* .

- ❖ El elemento de *name* especifica el nombre de la columna en la tabla.
- ❖ El elemento *length* especifica su longitud.
- ❖ El elemento *anulable* especifica si la columna es null o no.
- ❖ El elemento *unique* especifica si la columna es única.
- ❖ Si no especificamos el elemento *name*, el nombre del atributo se considerará el nombre de la columna en la tabla.

La anotación de `@Column`

Al igual que la anotación `@Table`, podemos usar la anotación `@Column` para mencionar los detalles de una columna en la tabla. La anotación `@Column` tiene muchos elementos, como *nombre*, *longitud*, *si permite almacenar valores nulos* y *si el valor del campo es único*.

```
@Entity
@Table(name = "Usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(nullable = false, length = 45)
    private String username;
    @Column(name="nombre", nullable = false, length = 45)
    private String nombreUsuario;
    @Column(unique=true, nullable = false, length = 100)
    private String email;
    @Column(unique=true, nullable = false, scale = 2)
    private float salario;
    private Integer estatus;
    @Column(nullable = true)
    private Date fechaRegistro;
```

Crear tablas automáticamente al lanzar la aplicación

Se debe configurar el programa para que genere las tablas:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost/bdvacantes2?useSSL=false&serverTimezone=GMT&allowPublicKeyRetrieval=true
spring.datasource.username=root
spring.datasource.password=root
#JPA
```

```
spring.jpa.hibernate.ddl-auto=create
spring.jpa.generate-ddl=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.jpa.show-sql=true
logging.level.org.hibernate.SQL=debug
# Table names physically
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

Crear tablas automáticamente al lanzar la aplicación

Se debe configurar el programa para que genere las tablas:

La propiedad `spring.jpa.hibernate.ddl-auto` puede tener asignados uno de los siguientes valores: `none`, `validate`, `update`, `create` y `create-drop`. Al especificar explícitamente una de estas opciones, está indicando a Spring Boot cómo inicializar el esquema.

Opción	Efecto
none	Sin inicialización del esquema de base de datos
create	Quita y crea el esquema al iniciar la aplicación. Con esta opción, todos sus datos desaparecerán en cada inicio.
create-drop	Crea un esquema en el inicio y destruye el esquema al cerrar el contexto. Útil para pruebas unitarias.
validate	Sólo comprueba si el esquema coincide con las entidades. Si el esquema no coincide, se producirá un error en el inicio de la aplicación. No realiza cambios en la base de datos.
update	Actualiza el esquema sólo si es necesario. Por ejemplo, si se agregó un nuevo campo en una entidad, simplemente alterará la tabla para crear una nueva columna sin destruir los datos.

Crear tablas automáticamente al lanzar la aplicación

Al ejecutar el programa se muestran las sentencias asociadas a crear la tabla usuario

```
20 @Entity
21 @Table(name = "Usuarios")
22 public class Usuario {
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     private Integer id;
26     @Column(nullable = false, length = 45)
27     private String username;
28     @Column(name="nombre", nullable = false, length = 45)
29     private String nombreUsuario;
30     @Column(unique=true, nullable = false, length = 100)
31     private String email;
```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL

Filter (e.g. text, !exclude)

```
Hibernate: create table Perfiles (id integer not null auto_increment, perfil varchar(255), primary key (id)) engine=MyISAM
Hibernate: create table Solicitudes (id integer not null auto_increment, archivo varchar(255), comentarios varchar(255), fecha datetime, idUsuario integer not null, primary key (id)) engine=MyISAM
Hibernate: create table Usuarios (id integer not null auto_increment, email varchar(100) not null, estatus integer, fechaRegistro datetime, nombre varchar(45) not null, salario float not null, username varchar(45) not null, primary key (id)) engine=MyISAM
Hibernate: create table Vacantes (id integer not null auto_increment, descripcion varchar(255), destacado integer, detalles varchar(255), estatus varchar(255), fecha datetime, imagen varchar(255), nombre varchar(255), salario double precision, primary key (id)) engine=MyISAM
Hibernate: alter table Usuarios drop index UK_7sia0h1arues629dbovxn1xb
Hibernate: alter table Usuarios add constraint UK_7sia0h1arues629dbovxn1xb unique (email)
Hibernate: alter table Direcciones add constraint FK76qj4qt0pqhdpeqxi0c0lwy4t foreign key (idUsuario) references Usuarios (id)
Hibernate: alter table Solicitudes add constraint FK03x21kcxaqneq7hdrjeelcup1 foreign key (idUsuario) references Usuarios (id)
```

Insertar datos automáticamente en una tabla

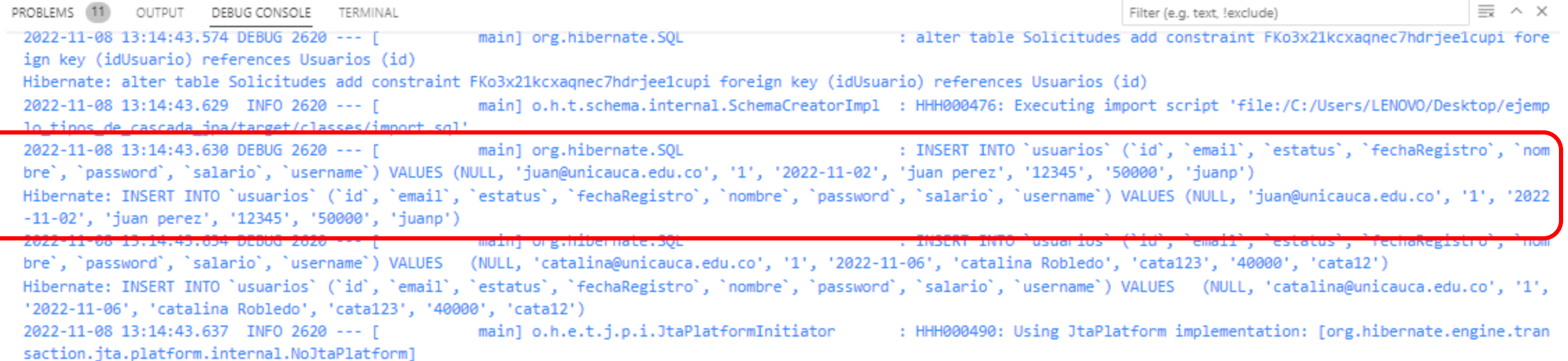
Se debe crear un archivo import.sql en el cual se colocan las sentencias SQL que se ejecutaran cuando la aplicación sea lanzada.

El archivo se crea en la carpeta resources



Insertar datos automáticamente en una tabla

Al revisar la consola se crean ecos que indican que se han insertado un conjunto de datos en la tabla usuarios.



The screenshot shows a Java IDE console window with the following tabs: PROBLEMS (11), OUTPUT, DEBUG CONSOLE, and TERMINAL. The DEBUG CONSOLE tab is active, displaying a log of Hibernate SQL statements. A red box highlights the following log entries:





```
2022-11-08 13:14:43.630 DEBUG 2620 --- [main] org.hibernate.SQL : INSERT INTO `usuarios` (`id`, `email`, `estatus`, `fechaRegistro`, `nombre`, `password`, `salario`, `username`) VALUES (NULL, 'juan@unicauca.edu.co', '1', '2022-11-02', 'juan perez', '12345', '50000', 'juanp')
Hibernate: INSERT INTO `usuarios` (`id`, `email`, `estatus`, `fechaRegistro`, `nombre`, `password`, `salario`, `username`) VALUES (NULL, 'juan@unicauca.edu.co', '1', '2022-11-02', 'juan perez', '12345', '50000', 'juanp')
```

The log also shows other statements, including an ALTER table statement for 'Solicitudes' and another INSERT statement for 'usuarios' with the email 'catalina@unicauca.edu.co'.

Insertar datos automáticamente en una tabla

Al revisar la tabla usuario se han registrado dos usuarios a partir del archivo.

+ Opciones

		id	calle	ciudad	pais	idUsuario
<input type="checkbox"/>	 Editar  Copiar  Borrar	1	calle 8 no 23 A 34	palmira	Colombia	3

+ Opciones

		id	email	estatus	fechaRegistro	nombre	password	salario	username
<input type="checkbox"/>	 Editar  Copiar  Borrar	1	juan@unicauca.edu.co	1	2022-11-02 00:00:00	juan perez	12345	50000	juanp
<input type="checkbox"/>	 Editar  Copiar  Borrar	2	catalina@unicauca.edu.co	1	2022-11-06 00:00:00	catalina Robledo	cata123	40000	cata12
<input type="checkbox"/>	 Editar  Copiar  Borrar	3	Andrea2@unicauca.edu.co	1	2022-11-08 18:14:45	Andrea Sanchez	12345	5000	Andrea2

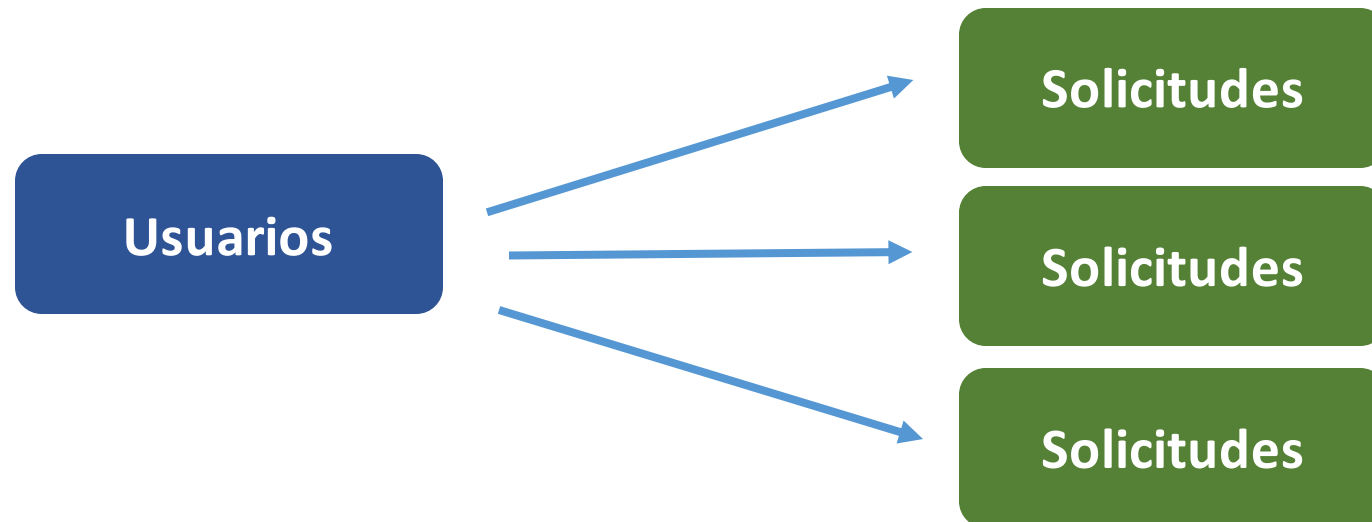
Para probar los ligados lazy eager vamos a almacenar un usuario y un conjunto de solicitudes

```
private void almacenarSolicitudes() {  
    Usuario objUsuario = new Usuario();  
    objUsuario.setNombre(nombre:"Tatiana Acosta");  
    objUsuario.setEmail(email:"tatiana@unicauca.edu.co");  
    objUsuario.setFechaRegistro(new Date());  
    objUsuario.setUsername(username:"Tatiana");  
    objUsuario.setPassword(password:"123456");  
    objUsuario.setEstatus(estatus:2);  
    this.servicioBDUsuarios.save(objUsuario);  
  
    Solicitud objSolicitud1 = new Solicitud();  
    objSolicitud1.setObjUsuario(objUsuario);  
    objSolicitud1.setFecha(new Date());  
    objSolicitud1.setArchivo(archivo:"Ruta al archivo de la solicitud 1");  
    objSolicitud1.setComentarios(comentarios:"Comentarios de la solicitud 1");  
    this.servicioDBSolicitudes.save(objSolicitud1);  
    Solicitud objSolicitud2 = new Solicitud();  
    objSolicitud2.setObjUsuario(objUsuario);  
    objSolicitud2.setFecha(new Date());  
    objSolicitud2.setArchivo(archivo:"Ruta al archivo de la solicitud 2");  
    objSolicitud2.setComentarios(comentarios:"Comentarios de la solicitud 2");  
    this.servicioDBSolicitudes.save(objSolicitud2);  
    Solicitud objSolicitud3 = new Solicitud();  
    objSolicitud3.setObjUsuario(objUsuario);  
    objSolicitud3.setFecha(new Date());  
    objSolicitud3.setArchivo(archivo:"Ruta al archivo de la solicitud 3");  
    objSolicitud3.setComentarios(comentarios:"Comentarios de la solicitud 3");  
    this.servicioDBSolicitudes.save(objSolicitud3);  
}
```

Cuando se trabaja con un ORM, el ligado de los datos se puede clasificar en dos tipos: perezosa (**Lazy**) y ansiosa (**Eager**).

Eager loading es un patrón de diseño en el que la inicialización de datos se produce en el acto.

Lazy loading es un patrón de diseño que se utiliza para diferir la inicialización de un objeto tanto tiempo como sea posible.



La carga lazy se puede habilitar simplemente usando el siguiente parámetro de anotación @oneToOne, @oneToMany, @ManyToOne, @ManyToMany:

fetch = FetchType.LAZY

La carga eager se puede habilitar simplemente usando el siguiente parámetro de anotación

fetch = FetchType.EAGER

Ejemplo Lazy loading

La carga se habilita mediante la propiedad
fetch = FetchType.LAZY

```
@Entity
@Table(name = "Usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String username;
    private String nombre;
    private String email;
    private String password;
    private Integer estatus;
    private Date fechaRegistro;

    @OneToMany(fetch = FetchType.LAZY, mappedBy="objUsuario")
    private List<Solicitud> solicitudes;

    @OneToOne(mappedBy = "objUsuario")
    private Direccion objDireccion;

    public Usuario()
    {
        this.solicitudes=new ArrayList<Solicitud>();
    }
}
```

Con el enfoque **Lazy Loading**, las *solicitudes* se cargaran solo cuando se llame explícitamente **getSolicitudes()**:

```
private void consultarUsuariosSolicitudes()
{
    Iterable<Usuario> listaUsuarios= this.servicioBDUsuarios.findAll();

    for (Usuario usuario : listaUsuarios) {
        System.out.println("Usuario");
        System.out.println("Nombres: " + usuario.getNombre());
        System.out.println("Solicitudes");
        Iterable<Solicitud> listaSolicitudes=usuario.getSolicitudes();
        for (Solicitud solicitud : listaSolicitudes) {
            System.out.println("id de la solicitud: " + solicitud.getId());
            System.out.println("Fecha de la solicitud: " + solicitud.getFecha());
            System.out.println("Ruta a la hoja de vida: " + solicitud.getArchivo());
            System.out.println("Comentarios: " + solicitud.getComentarios());
        }

        System.out.println(" ---- ---- ----");
    }
}
```

Lazy Loading, usa un objeto proxy y se dispara una consulta SQL separada para cargar las solicitudes .

Consultas que se genera

Consulta 1

```
Hibernate: select usuario0_.id as id1_4_, usuario0_.email as email2_4_, usuario0_.estatus as estatus3_4_, usuario0_.fechaRegistro as fechareg4_4_, usuario0_.nombre as nombre5_4_, usuario0_.password as password6_4_, usuario0_.username as username7_4_ from Usuarios usuario0_
```

Usuario

Nombres: Juan Perez

Solicitudes

Consulta 2

```
Hibernate: select solicitud0_.idUser as idusuari5_3_0_, solicitud0_.id as id1_3_0_, solicitud0_.id as id1_3_1_, solicitud0_.archivo as archivo2_3_1_, solicitud0_.comentarios as comentar3_3_1_, solicitud0_.fecha as fecha4_3_1_, solicitud0_.idUser as idusuari5_3_1_ from Solicitudes solicitud0_ where solicitud0_.idUser=?
```

id de la solicitud: 1

Fecha de la solicitud: 2021-11-09 00:00:00.0

Ruta a la hoja de vida: ruta al archivo de la hoja de vida

Comentarios: Tengo un amplio conocimiento en análisis contable

id de la solicitud: 2

Fecha de la solicitud: 2021-11-10 00:00:00.0

Ruta a la hoja de vida: ruta al archivo de la hoja de vida

Comentarios: Deseo unirme al Socio Logístico con mayor presencia en México,

¿Qué sucede si tenemos un mapper?

- Si el mapper invoca a `getSolicitudes` entonces se realizara la consulta a la BD
- Posteriormente se realiza el mapeo.
- Para evitar este comportamiento hay que configurar el mapeo para que no realice el mapeo

UsuarioDTO

Datos del usuario y
las solicitudes

Mapper

Al realizar el mapeo
invoca a solicitudes

Obtiene las solicitudes

Repositorio

Obtiene solo usuario y no
solicitudes

Ejemplo Eager loading

La carga se habilita mediante la propiedad
fetch = FetchType.EAGER

```
@Entity
@Table(name = "Usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String username;
    private String nombre;
    private String email;
    private String password;
    private Integer estatus;
    private Date fechaRegistro;

    @OneToMany(fetch = FetchType.EAGER, mappedBy="objUsuario")
    private List<Solicitud> solicitudes;
```

Con el enfoque **Eager Loading**, las *solicitudes* se cargaran cuando se listen los usuarios:

Esto será malo si estamos recuperando demasiados objetos en una sesión porque podemos obtener un error de pila de Java.

```
private void consultarUsuariosSolicitudes()
{
    Iterable<Usuario> listaUsuarios= this.servicioBDUsuarios.findAll();

    for (Usuario usuario : listaUsuarios) {
        System.out.println("Usuario");
        System.out.println("Nombres: " + usuario.getNombre());
        System.out.println("Solicitudes");
        Iterable<Solicitud> listaSolicitudes=usuario.getSolicitudes();
        for (Solicitud solicitud : listaSolicitudes) {
            System.out.println("id de la solicitud: " + solicitud.getId());
            System.out.println("Fecha de la solicitud: " + solicitud.getFecha());
            System.out.println("Ruta a la hoja de vida: " + solicitud.getArchivo());
            System.out.println("Comentarios: " + solicitud.getComentarios());
        }

        System.out.println(" ---- ---- ----");
    }
}
```

Eager Loading, usa un objeto proxy y se disparan dos consultas SQL juntas para cargar los usuarios, junto con sus solicitudes

Consultas que se generan

Consulta 1

```
Hibernate: select usuario0_.id as id1_4_, usuario0_.email as email2_4_, usuario0_.estatus as estatus3_4_, usuario0_.fechaRegistro as fechareg4_4_, usuario0_.nombre as nombre5_4_, usuario0_.password as password6_4_, usuario0_.username as username7_4_ from Usuarios usuario0_
```

```
8 Hibernate: select solicitud0_.idUserio as idusuari5_3_0_, solicitud0_.id as id1_3_0_, solicitud0_.archivo as archivo2_3_1_, solicitud0_.comentarios as comentar3_3_1_, solicitud0_.fecha as fecha4_3_1_, solicitud0_.idUserio as idusuari5_3_1_ from Solicitudes solicitud0_ where solicitud0_.idUserio=?
```

Usuario

Nombres: Juan Perez

Solicitudes

id de la solicitud: 1

Fecha de la solicitud: 2021-11-09 00:00:00.0

Ruta a la hoja de vida: ruta al archivo de la hoja de vida

Comentarios: Tengo un amplio conocimiento en análisis contable

id de la solicitud: 2

Fecha de la solicitud: 2021-11-10 00:00:00.0

Ruta a la hoja de vida: ruta al archivo de la hoja de vida

Comentarios: Deseo unirme al Socio Logístico con mayor presencia en México,

id de la solicitud: 3

Fecha de la solicitud: 2021-11-04 00:00:00.0

Ruta a la hoja de vida: ruta a la hoja de vida

Comentarios: Tengo experiencia como coordinador o coordinadora de marketing

Consulta 2

Diferencias entre Lazy Loading y Eager Loading

La principal diferencia entre los dos tipos de búsqueda es un momento en el que los datos se cargan en una memoria.

```
private void consultarUsuariosSolicitudes()
{
    Iterable<Usuario> listaUsuarios= this.servicioBDUsuarios.findAll();

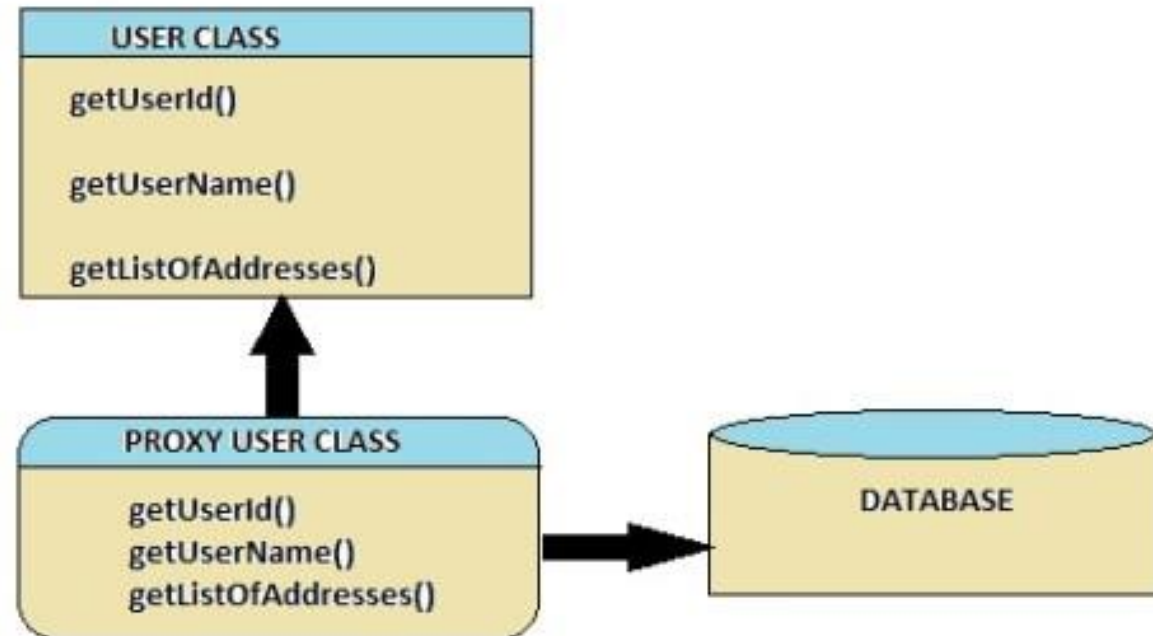
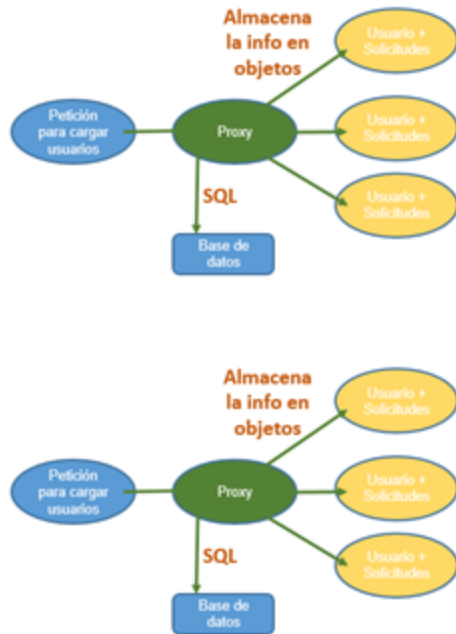
    for (Usuario usuario : listaUsuarios) {
        System.out.println("Usuario");
        System.out.println("Nombres: " + usuario.getNombre());
        System.out.println("Solicitudes");
        Iterable<Solicitud> listaSolicitudes=usuario.getSolicitudes();
        for (Solicitud solicitud : listaSolicitudes) {
            System.out.println("Id de la solicitud: " + solicitud.getId());
            System.out.println("Fecha de la solicitud: " + solicitud.getFecha());
        }
    }
}
```

Con el enfoque de Lazy Loading, *las solicitudes* se cargaran solo cuando se llamen explícitamente mediante un getter.

Pero con un enfoque Eager, las solicitudes se cargaran inmediatamente en la primera línea.

Uso del patrón proxy

Hibernate intercepta las llamadas a una entidad (Por ej Usuario) sustituyéndola por un proxy derivado de la clase de una entidad.



Lazy Loading

Ventajas:

- El tiempo de carga inicial es mucho menor que en el otro enfoque
- Menor consumo de memoria que en el otro enfoque

Desventajas:

- La inicialización retrasada puede afectar el rendimiento durante momentos no deseados
- En algunos casos, debe manejar los objetos inicializados de forma diferida con especial cuidado o podría terminar con una excepción

Eager Loading

Ventajas:

- Al realizar una consulta inicial, ya se tienen todos los datos necesarios

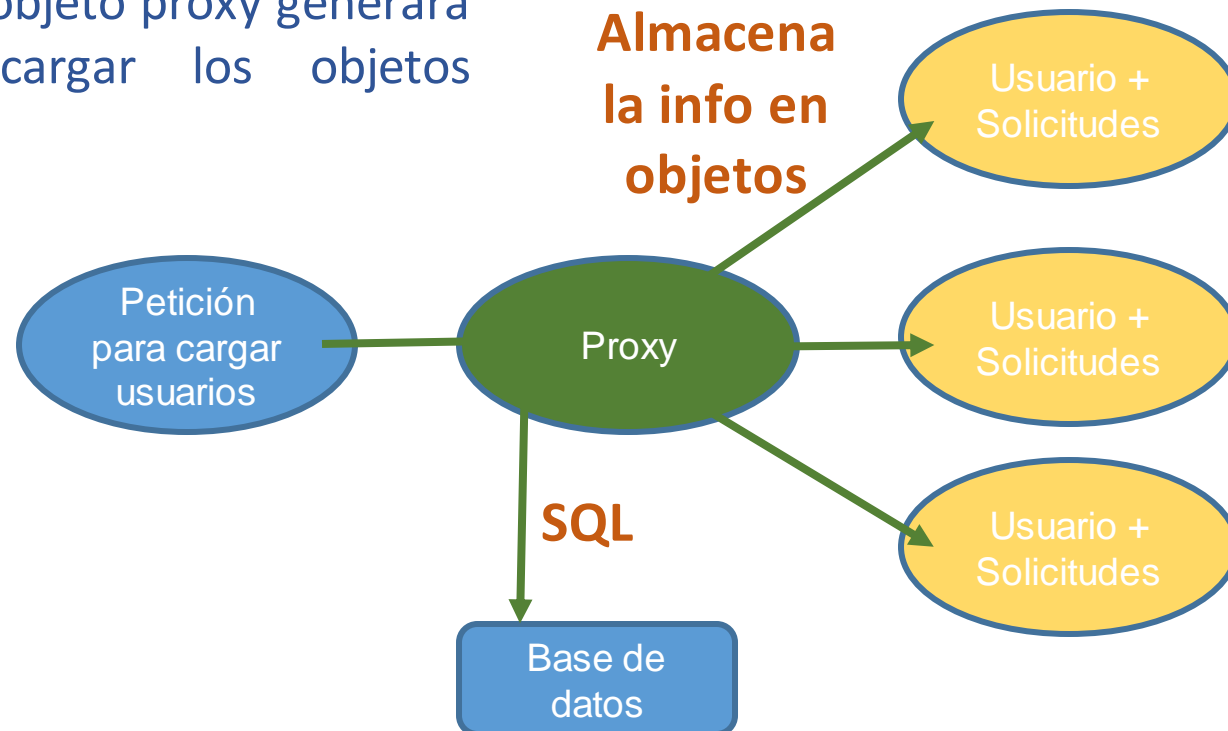
Desventajas:

- Largo tiempo de carga inicial
- Cargar demasiados datos innecesarios puede afectar el rendimiento

¿Lazy Loading es lo mejor?

Eager

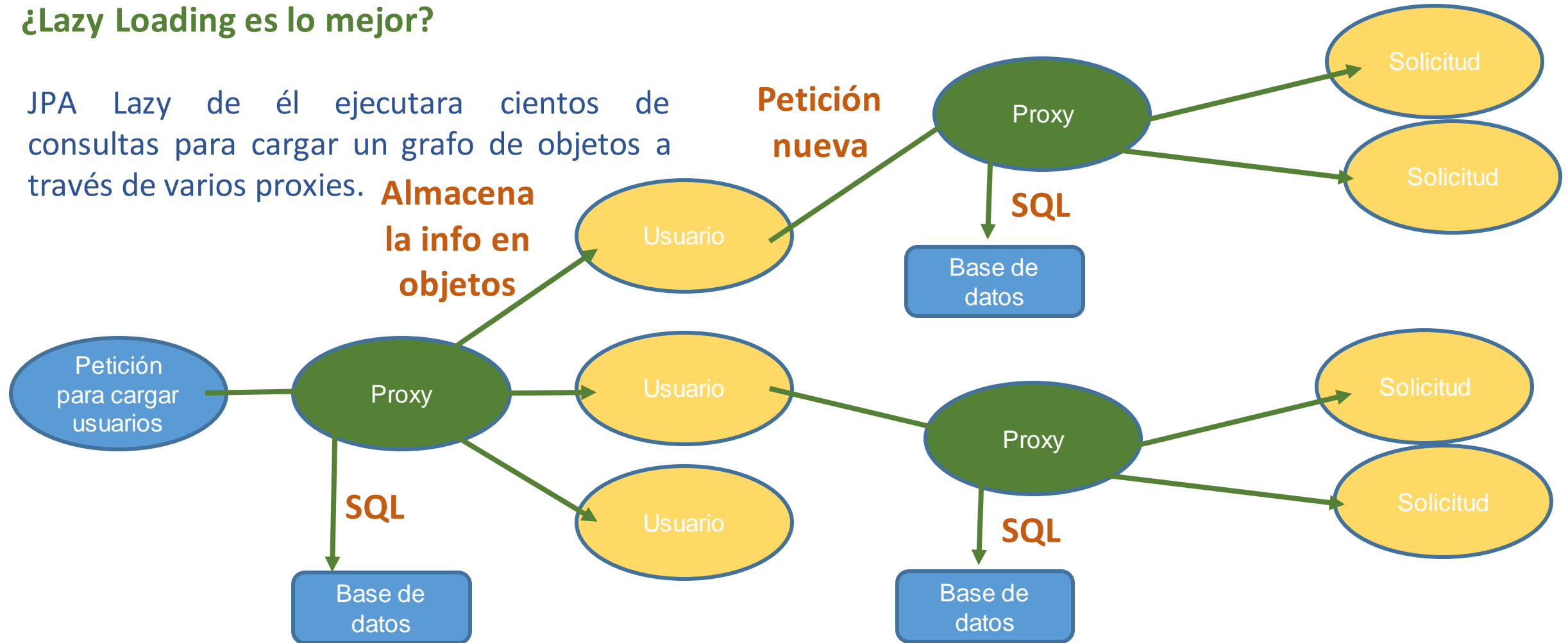
JPA Eager mediante un objeto proxy generará una consulta para cargar los objetos solicitados



¿Lazy Loading es lo mejor?

JPA Lazy de él ejecutara cientos de consultas para cargar un grafo de objetos a través de varios proxies.

**Almacena
la info en
objetos**



Las relaciones entre entidades a menudo dependen de la existencia de otra entidad, por ejemplo, la relación *Persona - Dirección* . Sin la *Persona* , la entidad *Dirección* no tiene ningún significado propio.

Cuando eliminamos la entidad *Persona* , nuestra entidad *Dirección* también debería eliminarse

La cascada es la forma de lograrlo. Cuando realizamos alguna acción en la entidad objetivo, la misma acción se aplicará a la entidad asociada.

Tipo de cascada JPA

Todas las operaciones en cascada específicas de JPA están representadas por la enumeración *javax.persistence.CascadeType* que contiene entradas:

- *ALL*
- *PERSIST*
- *MERGE*
- *REMOVE*
- *REFRESH*
- *DETACH*

CascadeType.PERSIST

El tipo de cascada PERSIST propaga la operación de persistencia de una entidad principal a una secundaria. Cuando guardamos la entidad de persona, la entidad de dirección también se guardará.

```
@Entity
@Table(name = "Usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(nullable = false, length = 45)
    private String username;
    @Column(unique=true, name="nombre", nullable = false, length = 45)
    private String nombreUsuario;
    @Column( unique=true, nullable = false, length = 100)
    private String email;
    private String password;
    private Integer estatus;
    @Column( nullable = true)
    private Date fechaRegistro;

    @OneToMany(fetch = FetchType.LAZY, mappedBy="objUsuario")
    private List<Solicitud> solicitudes;

    @OneToOne(cascade={CascadeType.PERSIST}, CascadeType.REMOVE}, mappedBy = "objUsuario")
    private Direccion objDireccion;
```

CascadeType.PERSIST

El tipo de cascada PERSIST propaga la operación de persistencia de una entidad principal a una secundaria. Cuando guardamos la entidad de persona, la entidad de dirección también se guardará.

Antes

```
private void almacenarUsuario()
{
    Usuario user = new Usuario();
    user.setNombre("Andrea Sanchez");
    user.setEmail("Andrea@unicauca.edu.co");
    user.setFechaRegistro(new Date());
    user.setUsername("Andrea");
    user.setPassword("12345");
    user.setEstatus(1);

    Usuario objUsuarioAlmacenado=this.servicioBDUsuarios.save(user);

    System.out.println("Id generado para el usuario: " + objUsuarioAlmacenado.getId());

    Direccion objDireccion= new Direccion();
    objDireccion.setCalle("calle 8 no 23 A 34");
    objDireccion.setCiudad("palmira");
    objDireccion.setPais("Colombia");

    objDireccion.setObjUsuario(objUsuarioAlmacenado);

    Direccion objDireccionAlmacenada=this.servicioBDDirecciones.save(objDireccion);
    System.out.println("id almacenado: " + objDireccionAlmacenada.getId());
}
```

Despues

```
private void almacenarUsuario() {
    Usuario user = new Usuario();
    user.setNombre(nombre:"Andrea Sanchez");
    user.setEmail(email:"Andrea2@unicauca.edu.co");
    user.setFechaRegistro(new Date());
    user.setUsername(username:"Andrea2");
    user.setPassword(password:"12345");
    user.setEstatus(estatus:1);
    user.setSalario(salario:5000);

    Direccion objDireccion = new Direccion();
    objDireccion.setCalle(calle:"calle 8 no 23 A 34");
    objDireccion.setCiudad(ciudad:"palmira");
    objDireccion.setPais(pais:"Colombia");

    user.setObjDireccion(objDireccion);
    objDireccion.setObjUsuario(user);

    this.servicioBDUsuarios.save(user);
}
```

CascadeType.PERSIST

```
Hibernate: insert into Usuarios (email, estatus, fechaRegistro, nombre, password, salario, username) values (?, ?, ?, ?, ?, ?, ?)
2023-10-18 13:52:51.771 DEBUG 10616 --- [main] org.hibernate.SQL : insert into Direcciones (calle, ciudad, idUsuario, pais) values (?, ?, ?, ?)
Hibernate: insert into Direcciones (calle, ciudad, idUsuario, pais) values (?, ?, ?, ?)
2023-10-18 13:52:51.902 INFO 10616 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2023-10-18 13:52:51.935 INFO 10616 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2023-10-18 13:52:52.229 INFO 10616 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

Tabla usuarios

















<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	juan@unicauca.edu.co	1	2022-11-02 00:00:00.000000	juan perez	12345	50000	juanp
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	catalina@unicauca.edu.co	1	2022-11-06 00:00:00.000000	catalina Robledo	cata123	40000	cata12
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	3	Andrea2@unicauca.edu.co	1	2024-04-08 16:52:07.849000	Andrea Sanchez	12345	5000	Andrea2

Tabla direcciones

	 Editar	 Copiar	 Borrar		id	calle	ciudad	pais	idUsuario
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1		calle 8 no 23 A 34	palmira	Colombia	3

CascadeType.REMOVE

Propaga la operación de eliminación de la entidad principal a la secundaria.

```
@Entity
@Table(name = "Usuarios")
public class Usuario {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(nullable = false, length = 45)
    private String username;
    @Column(unique=true, name="nombre", nullable = false, length = 45)
    private String nombreUsuario;
    @Column( unique=true, nullable = false, length = 100)
    private String email;
    private String password;
    private Integer estatus;
    @Column( nullable = true)
    private Date fechaRegistro;

    @OneToMany(fetch = FetchType.LAZY, mappedBy="objUsuario")
    private List<Solicitud> solicitudes;

    @OneToOne(cascade={CascadeType.REMOVE, CascadeType.PERSIST}, mappedBy = "objUsuario")
    private Direccion objDireccion;
```

```
private void eliminarUsuario()
{
    Optional<Usuario> optional = this.servicioBDUsuarios.findById(36);
    Usuario user=optional.get();
    this.servicioBDUsuarios.delete(user);
}
```

CascadeType.REMOVE

```
2023-10-18 13:58:03.624 DEBUG 11560 --- [main] org.hibernate.SQL : delete from Usuarios where id
=?
Hibernate: delete from Usuarios where id=?
2023-10-18 13:58:03.697 INFO 11560 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFacto
ry for persistence unit 'default'
2023-10-18 13:58:03.704 INFO 11560 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initia
ted...
2023-10-18 13:58:03.739 INFO 11560 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown comple
ted.
```


<https://www.baeldung.com/jpa-cascade-types>

<https://www.baeldung.com/hibernate-lazy-eager-loading>

<https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/transaction.html>

<https://www.apascualco.com/spring-boot/spring-transactional/>

Muchas gracias
Preguntas

