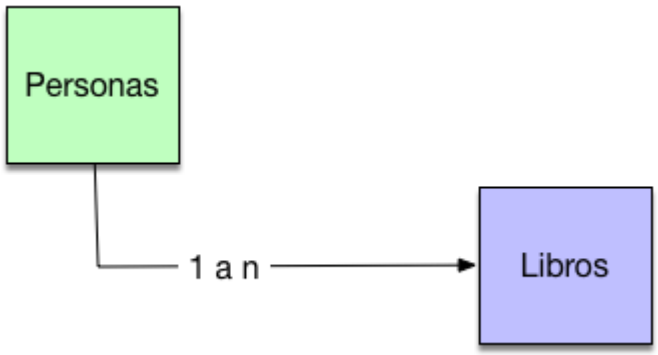
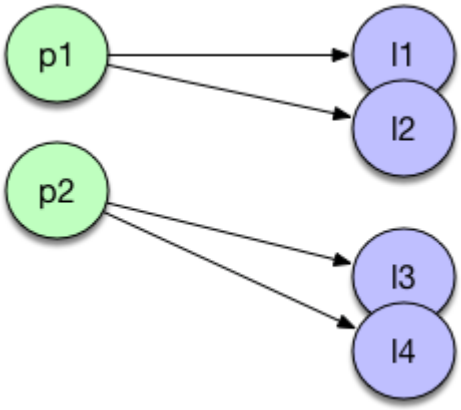
El concepto de JPA Lazy Fetching siempre cuesta un poco entenderlo y es uno de los conceptos fundamentales de Java Persistence API .¿Cómo funciona exactamente? Vamos a explicarlo a detalle. Lo primero que tenemos que entender es que cuando trabajamos con JPA , definimos un modelo de dominio que relaciona las diferentes clases entre ellas.



Cuando JPA ejecuta una consulta se apoya en el modelo de dominio para generar un grafo de objetos en memoria.



El concepto de JPA Lazy Fetching esta asociado a como se carga este grafo de objetos en memoria. Hay situaciones en el cual lo cargamos completo pero hay otras situaciones en las

cuales su carga es parcial. Vamos a a partir del siguiente modelo de clases:

```
package com.arquitecturajava;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
public class Libro {
        @Id
        private String titulo;
        private int paginas;
         @ManyToOne(optional=false)
     @JoinColumn(name="persona_nombre")
        private Persona persona;
        public String getTitulo() {
                return titulo;
        }
        public void setTitulo(String titulo) {
                this.titulo = titulo;
        }
        public int getPaginas() {
                return paginas;
        }
        public void setPaginas(int paginas) {
                this.paginas = paginas;
        }
```

```
	public Persona getPersona() {
		return persona;
	}
	public void setPersona(Persona persona) {
		this.persona = persona;
	}
	public Libro(String titulo, int paginas, Persona persona) {
		super();
		this.titulo = titulo;
		this.paginas = paginas;
		this.persona = persona;
	}
	public Libro() {
		super();
	}
}
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class Persona {
	@Id
	private String nombre;
	private String apellidos;
```

```
private int edad;
@OneToMany(mappedBy="persona",cascade = CascadeType.PERSIST)
private List<Libro> libros= new ArrayList<Libro>();
public List<Libro> getLibros() {
        return libros;
}
public void setLibros(List<Libro> libros) {
        this.libros = libros;
}
public boolean addLibro(Libro e) {
        return libros.add(e);
}
public String getNombre() {
        return nombre;
}
public void setNombre(String nombre) {
        this.nombre = nombre;
}
public String getApellidos() {
        return apellidos;
}
public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
}
public int getEdad() {
        return edad;
}
public void setEdad(int edad) {
        this.edad = edad;
}
public Persona(String nombre, String apellidos, int edad) {
```

```
                super();
                this.nombre = nombre;
                this.apellidos = apellidos;
                this.edad = edad;
        }
        public Persona() {
                super();
        }
}
```

Podemos lanzar una consulta que solo cargue las Personas

```
package com.arquitecturajava;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Principal4 {

        public static void main(String[] args) {
                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadPersonas");
                EntityManager em = emf.createEntityManager();
                List<Persona> lista=em.createQuery("select p from
Persona p", Persona.class).getResultList();
                for (Persona p:lista) {
```

```
                        System.out.println(p.getNombre());
                        System.out.println(p.getApellidos());
                }
        }
}
```

Esta consulta únicamente selecciona las personas , sin sus libros

Tenemos una única consulta que devuelve los datos de las personas:

```
INFO: HHH10001003: Autocommit mode: false
jul 20, 2017 11:18:59 AM org.hibernate.engine.jdbc.connections.intern
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Thu Jul 20 11:18:59 CEST 2017 WARN: Establishing SSL connection witho
jul 20, 2017 11:19:00 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
jul 20, 2017 11:19:00 AM org.hibernate.hql.internal.QueryTranslatorFa
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select persona0_.nombre as nombre1_1_, persona0_.apellidos
pepe
perez
juan
perez
```

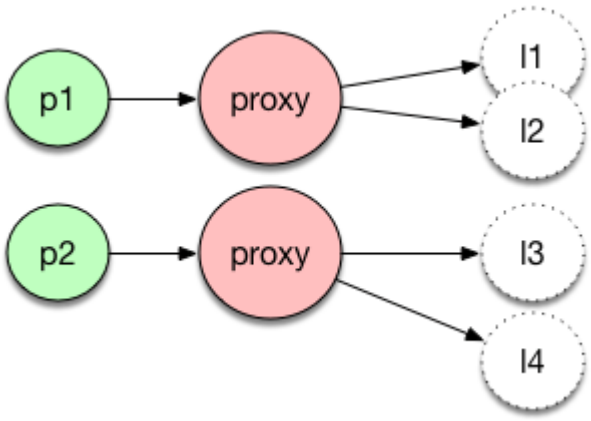
¿Qué sucede si nosotros le pedimos a nuestro código que acceda a la relación de libros y muestre los libros por la consola?.

```
package com.arquitecturajava;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Principal5 {

	public static void main(String[] args) {
		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadPersonas");
		EntityManager em = emf.createEntityManager();
		List<Persona> lista=em.createQuery("select p from
Persona p", Persona.class).getResultList();
		for (Persona p:lista) {
			System.out.println(p.getNombre());
			System.out.println(p.getApellidos());
			for(Libro l :p.getLibros()) {
				System.out.println(l.getTitulo());
				System.out.println(l.getPaginas());
			}
		}
	}
}
```

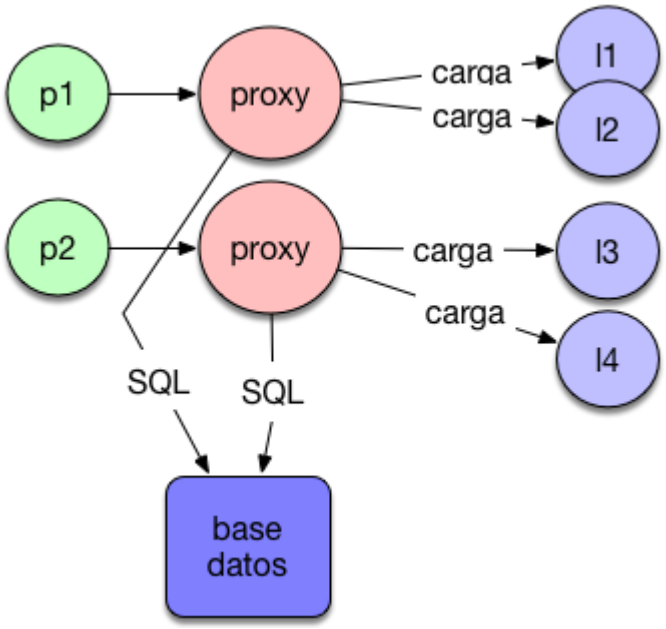
El framework nos devuelve los datos sorprendentemente , sin problemas .

```
Problems  Javadoc  Declaration  Console
Principal5 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (20 j
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select persona0_.nombre as nombre1_1_, persona0_.apellidos as apellido2_1
pepe
perez
Hibernate: select libros0_.persona_nombre as persona_3_0_0_, libros0_.titulo as titu
java
200
juan
perez
Hibernate: select libros0_.persona_nombre as persona_3_0_0_, libros0_.titulo as titu
java2
200
```

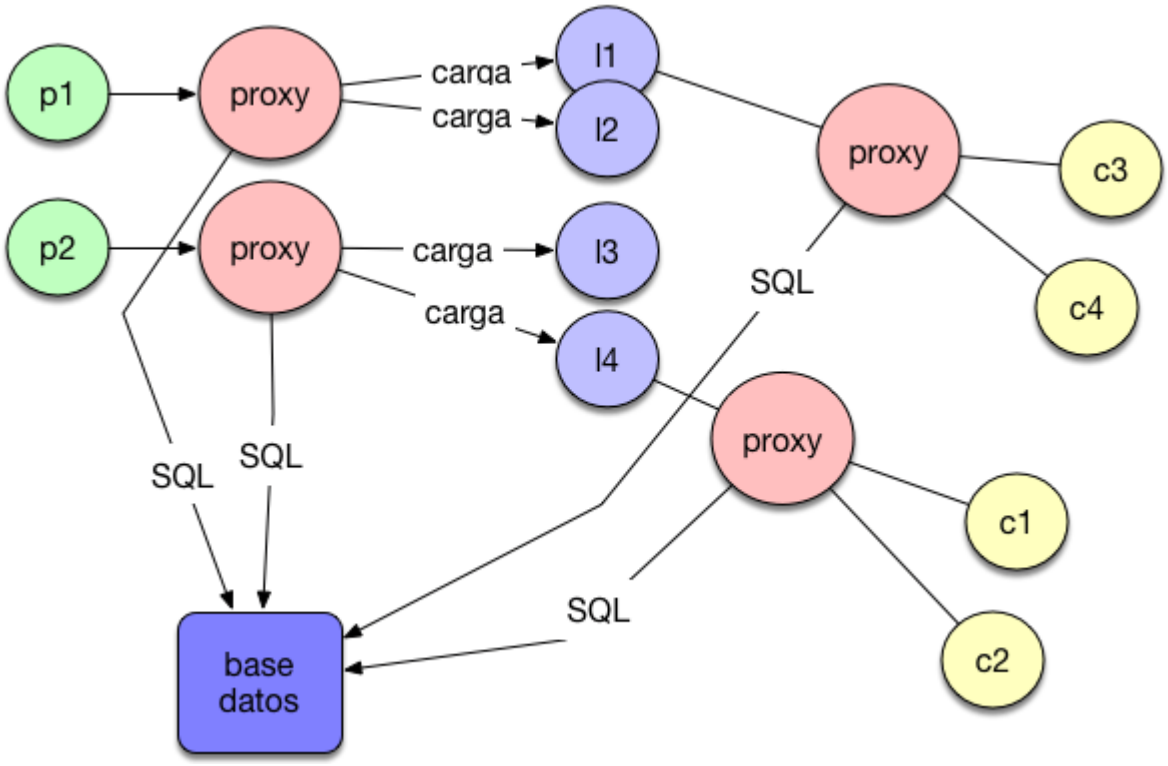
¿Qué ha pasado?. No hemos seleccionado los libros previamente, así que no los tenemos en memoria ¿ Cómo es que los datos aparecen?. Acabamos de utilizar las capacidades de JPA Lazy Fetching. JPA al construir las relaciones ha ubicado proxies o intermedarios en cada una de ellas.



De tal forma que cuando accedamos a la relación estos proxies se activan acceden a la base y se traen los nuevos objetos



De esta manera optimizamos el acceso a los datos. Muchas veces la gente piensa que esta es la mejor forma de proceder . Pero lamentablemente no es siempre así ya que el uso de JPA Lazy Fetching si abusamos de él ejecutara cientos de consultas para cargar un grafo de objetos complejos a través de los proxies.



Es lo que comúnmente se denomina el problema de las N+1 Queries o el antipatron de OpenSession in View . Siempre que sea posible es mejor usar un enfoque tipo fetch join en las cuales el desarrollador indica que datos quiere de inicio a nivel de JPA QL

Otros artículos relacionados:

1. JPA Criteria API , un enfoque diferente
2. JPA SQL Injection y sus problemas
3. Un ejemplo de JPA Entity Graph
4. JPA WikiPedia