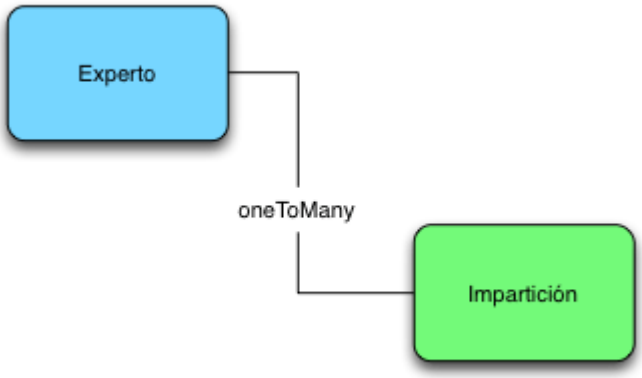
Mucha gente desconoce el concepto de JPA Entity Graph , y como nos pueden ayudar a mejorar el rendimiento de las consultas de JPA que creamos. Para entender como funcionan hay que recordar algunas cosas de JPA. En primer lugar que todas las consultas que realizamos oneToMany son lazy feching , es decir los datos se cargan según los vamos solicitando generándose en muchas ocasiones las indeseadas n+1 queries.

JPA OneToMany

Para entender mejor el concepto ,vamos a construir un ejemplo usando dos clases: Experto e Impartición. Ambas clases están relacionadas a través de una relación @oneToMany. Un Experto es capaz de realizar varias imparticiones.



Vamos a mostrar el contenido de ambas clases:

```
package com.arquitecturajava;


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

```
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.NamedAttributeNode;
import javax.persistence.NamedEntityGraph;
import javax.persistence.OneToMany;

@Entity
public class Experto {
	@Id
	private String nombre;
	@OneToMany(mappedBy="experto")
	private List<Imparticion> imparticiones= new
ArrayList<Imparticion>();
	public List<Imparticion> getImparticiones() {
		return imparticiones;
	}

	public void setImparticiones(List<Imparticion> imparticiones)
{
		this.imparticiones = imparticiones;
	}

	public String getNombre() {
		return nombre;
	}

	public Experto(String nombre) {
		super();
		this.nombre = nombre;
```

```
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public void addImparticion(Imparticion i) {
		imparticiones.add(i);
	}

	public Experto() {
		super();
	}
}
```

```
package com.arquitecturajava;


import java.util.Date;

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

@Entity
```

```
public class Imparticion {
	@Id
	private int id;
	private Date fecha;
	private String titulo;
	@ManyToOne
	@JoinColumn(name="nombre_experto")
	private Experto experto;

	public Imparticion() {
		super();
	}

	public Imparticion(int id, Date fecha, String titulo, Experto
experto) {
		super();
		this.id = id;
		this.fecha = fecha;
		this.titulo = titulo;
		this.experto = experto;
	}

	public Experto getExperto() {
		return experto;
	}

	public void setExperto(Experto experto) {
		this.experto = experto;
	}

	public int getId() {
```

```
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public Date getFecha() {
		return fecha;
	}

	public void setFecha(Date fecha) {
		this.fecha = fecha;
	}

	public String getTitulo() {
		return titulo;
	}

	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}
}
```

Vamos a crear un programa main y solicitamos una lista de los Expertos:

```
package com.arquitecturajava;

import java.util.List;

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

public class Principal {

        public static void main(String[] args) {

                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadCharla");
                EntityManager em = emf.createEntityManager();
                TypedQuery<Experto> consulta = em.createQuery("select
e from Experto e", Experto.class);

                List<Experto> lista = consulta.getResultList();

                for (Experto e : lista) {

                        System.out.println(e.getNombre());
                        for (Imparticion i : e.getImparticiones()) {

                                System.out.println(i.getTitulo());
                        }

                }
                em.close();
```

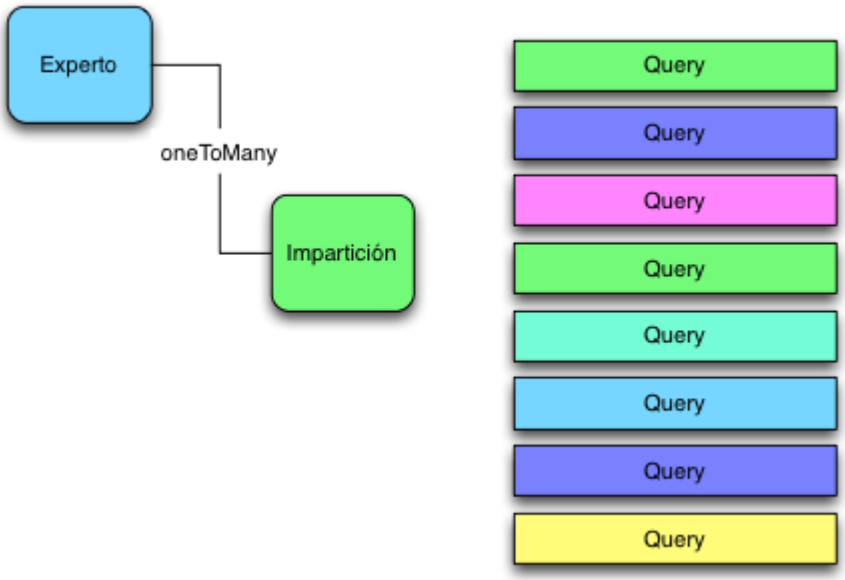
```
        }

}
```

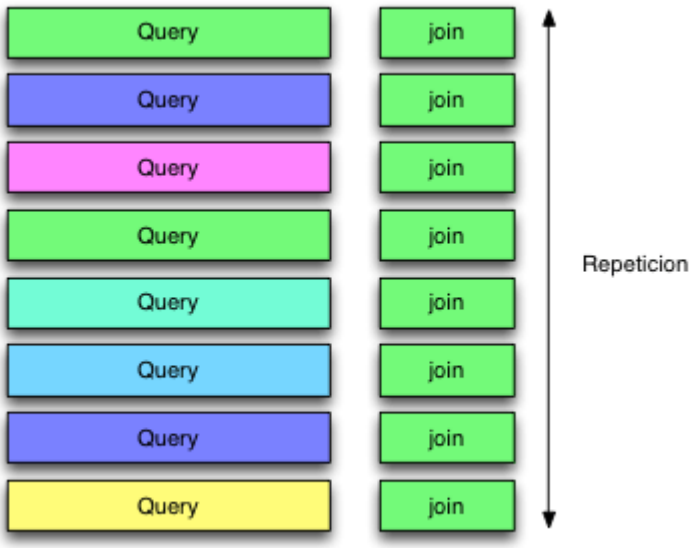
Una vez solicitada la lista accedemos al conjunto de imparticiones que cada Experto tiene. El resultado lo podemos ver por la consola.

```
Principal (52) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_71.jdk/Contents/Home/bin/java (9
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select experto0_.nombre as nombre1_0_ from Experto experto0_
pedro
Hibernate: select imparticio0_.nombre_experto as nombre_e4_1_0_, imparticio0_.id
 as id1_1_0_, imparticio0_.id as id1_1_1_, imparticio0_.nombre_experto as nombre
_e4_1_1_, imparticio0_.fecha as fecha2_1_1_, imparticio0_.titulo as titulo3_1_1_
 from Imparticion imparticio0_ where imparticio0_.nombre_experto=?
excel
gema
Hibernate: select imparticio0_.nombre_experto as nombre_e4_1_0_, imparticio0_.id
 as id1_1_0_, imparticio0_.id as id1_1_1_, imparticio0_.nombre_experto as nombre
_e4_1_1_, imparticio0_.fecha as fecha2_1_1_, imparticio0_.titulo as titulo3_1_1_
 from Imparticion imparticio0_ where imparticio0_.nombre_experto=?
word
wordAvanzado
```

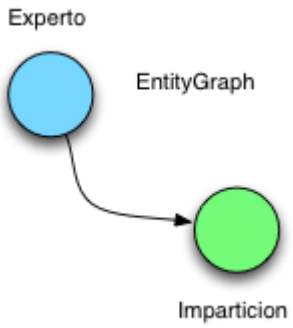
Lamentablemente nos encontramos ante una situación en la que se producen n+1 Queries. En este caso tres consultas una para obtener los expertos y otras dos para cada grupo de imparticiones. Hubiera sido mejor utilizar un Join. Esto se soluciona habitualmente usando un fetchJoin que nos permite realizar un Join con JPA. Sin embargo nos queda un problema. ¿Cuantas consultas diferentes haremos sobre estas dos tablas?



La respuesta es que muchas. En la mayoría de estas consultas necesitaremos que se realice el mismo join.



Para evitar estas situaciones se han creado los JPA Entity Graph que nos permiten definir un grafo de componentes y atributos que cargar en una query.



Usando JPA Entity Graph

Para ello necesitaremos añadir la anotación de NamedEntityGraph en la clase Experto.

```
</pre>
@Entity
@NamedEntityGraph(

name = "ExpertoConImparticiones", attributeNodes = {
@NamedAttributeNode("imparticiones"),
})
public class Experto {
<pre>
```

En segundo lugar debemos modificar el programa principal para que cuando la consulta se realice se apoye en un EntityGraph.

```
 
```

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

public class Principal2 {

public static void main(String[] args) {

EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadCharla");
EntityManager em = emf.createEntityManager();
TypedQuery<Experto> consulta = em.createQuery("select distinct e from
Experto e", Experto.class);

consulta.setHint("javax.persistence.loadgraph",
em.getEntityGraph("ExpertoConImparticiones"));

List<Experto> lista = consulta.getResultList();

for (Experto e : lista) {

System.out.println(e.getNombre());
for (Imparticion i : e.getImparticiones()) {

System.out.println(i.getTitulo());
}

}
em.close();
```

```
}

}
```

Una vez hecho esto la consulta se ejecutará como un join.

```
sep 09, 2016 8:41:32 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
sep 09, 2016 8:41:33 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiat
or initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select distinct experto0_.nombre as nombre1_0_0_, imparticio1_.id as
id1_1_1_, imparticio1_.nombre_experto as nombre_e4_1_1_, imparticio1_.fecha as f
echa2_1_1_, imparticio1_.titulo as titulo3_1_1_, imparticio1_.nombre_experto as
nombre_e4_1_0__, imparticio1_.id as id1_1_0__ from Experto experto0_ left outer
join Imparticion imparticio1_ on experto0_.nombre=imparticio1_.nombre_experto
pedro
excel
gema
word
wordAvanzado
```

Así podremos reutilizar los JPA Entity Graph según nuestras necesidades.

Otros artículos relacionados : ORM.XML , Introducción a JPA , JPA NamedQueries