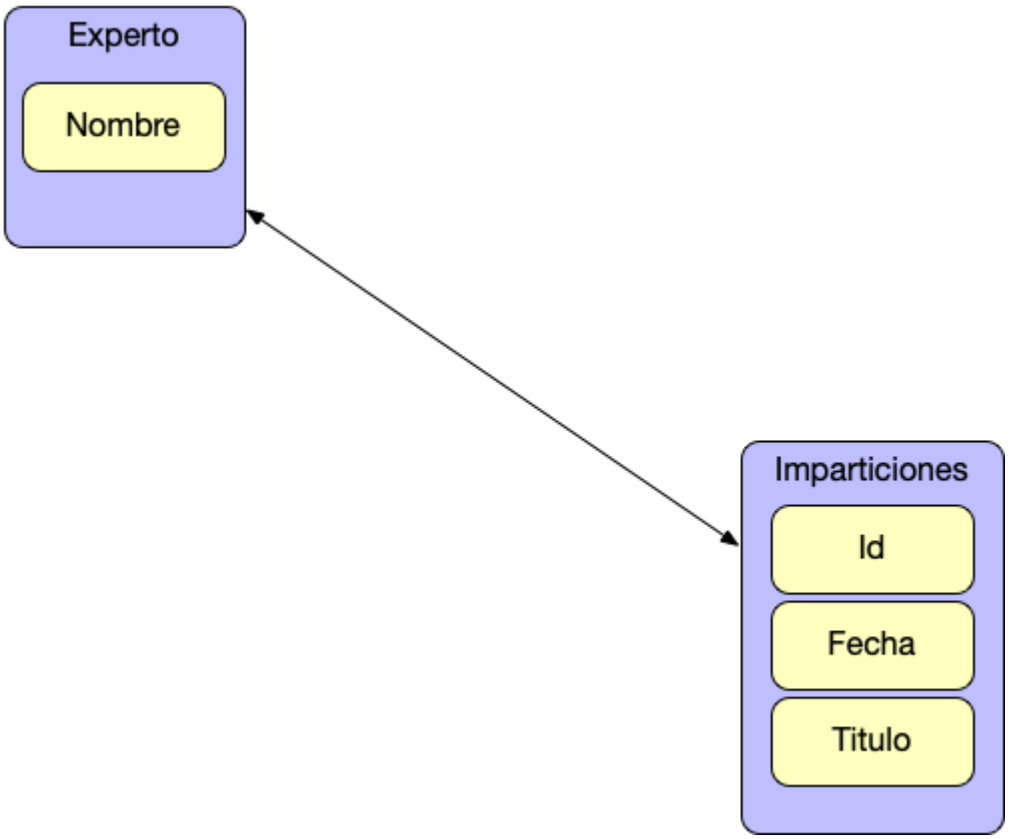
JPA Join Fetch es una de las opciones de las que dispone el estándar de JPA a la hora de reducir el número de consultas que se generan contra la base de datos. Algo que es bastante habitual y que degrada el rendimiento sino tenemos cuidado. Vamos a ver un ejemplo ,para ello partiremos de dos clases Java (Experto e Impartición) que están relacionadas (un Experto imparte varias charlas).



El código Java de las clases utilizando anotaciones JPA será :

```
package com.arquitectajava;

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 Experto {
    @Id
    private String nombre;

    public Experto() {
        super();
    }
    @OneToMany(mappedBy = "experto", cascade = CascadeType.PERSIST)
    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);
    }

}
```

```
package com.arquitectajava;

import java.util.Date;

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

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

    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 construir el programa principal que nos selecciona los Expertos para luego recorrer sus imparticiones.

```
package com.arquitectajava;

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());
            List < Imparticion > imparticiones = e.getImparticiones();
            for (Imparticion i: imparticiones) {
                System.out.println(i.getTitulo());
            }

        }
        em.close();
    }

}
```

Si ejecutamos el programa podremos ver el resultado en la consola:

```
Hibernate: select experto0_.nombre as nombre1_0_
juan
Hibernate: select imparticio0_.nombreExperto as
.net
java
miguel
Hibernate: select imparticio0_.nombreExperto as
php
c#
```

Los datos se imprimen , pero estamos realizando 3 consultas para obtenerlos .En primer lugar se seleccionan todos los Expertos y por cada Experto se realiza una consulta para saber sus imparticiones. Este es el problema clásico de las n+1 Queries. Cuando existan 100 imparticiones , se lanzarán 101 consultas lo cual es un verdadero problema. Vamos a solventar este problema utilizando JPA Join Fetch.

JPA Join Fetch

Para ello nos bastará con modificar la consulta de JPA que estamos utilizando y obligarla a que incluya las imparticiones añadiendo la clausula join fetch dentro de la consulta . Esto obligará a que Hibernate realice un Join en la base de datos. Ojo que hay que añadir una clausula distinct.

```
package com.arquitectajava;

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
distinct e from Experto e join fetch e.imparticiones", Experto.class);

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

        for (Experto e: lista) {

            System.out.println(e.getNombre());
```

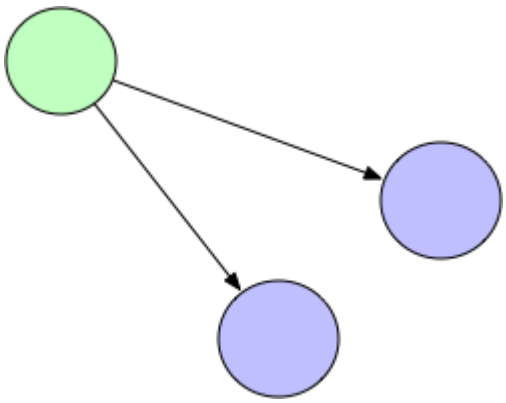
```
            List < Imparticion > imparticiones = e.getImparticiones();
            for (Imparticion i: imparticiones) {
                System.out.println(i.getTitulo());
            }
        }
        em.close();
    }

}
```

Solución con JPA

Ahora la consulta es “select distinct e from Experto e join fetch e.imparticiones” obligando a JPA a incluir las imparticiones a través de un JOIN.

```
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select distinct experto0_.nombre as nombre1_0_0_, impar
juan
.net
java
miguel
php
c#
```

Ahora una sola consulta es suficiente ya que nos hemos apoyado en JPA Join Fetch y cargará un único grafo de objetos.



Recordemos que aunque los frameworks de persistencia aportan mucho , hay que saber

utilizarlos. Un mal uso de estos frameworks puede generar los peores resultados.

Otros artículos relacionados :

- Java Override y encapsulación
- JavaScript ES6 fetch API
- Fetch API Promesas Ajax y conceptos claves
- JPA Proxy y su funcionamiento
- JPA OneToOne
- JPA OneToMany