El concepto de JPA Remove y sus peculiaridades . Java Persistence API siempre tiene curiosidades incluso en el manejo de operaciones que nos pueden parecer esenciales o básicas . Vamos a ver un ejemplo del uso de JPA remove para eliminar entidades de una base de datos. Supongamos que disponemos de la clase Libro.

```
package com.arquitecturajava.jpabasico.dominio;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Libro {
	@Id
	private String isbn;
	private String titulo;
	private String autor;
	private Date fecha;
	private int precio;

	public String getIsbn() {
		return isbn;
	}

	public void setIsbn(String isbn) {
		this.isbn = isbn;
	}

	public String getTitulo() {
		return titulo;
```

```
}

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

public String getAutor() {
        return autor;
}

public void setAutor(String autor) {
        this.autor = autor;
}

public Date getFecha() {
        return fecha;
}

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

public int getPrecio() {
        return precio;
}

public void setPrecio(int precio) {
        this.precio = precio;
}

public Libro(String isbn, String titulo, String autor, Date
```

```
fecha, int precio) {
		super();
		this.isbn = isbn;
		this.titulo = titulo;
		this.autor = autor;
		this.fecha = fecha;
		this.precio = precio;
	}

	public Libro(String isbn) {
		super();
		this.isbn = isbn;
	}

	public Libro() {
		super();
	}

	@Override
	public String toString() {
		return "Libro [isbn=" + isbn + ", titulo=" + titulo +
", autor=" + autor + "]";
	}

}
```

Esta clase esta construida con los métodos set y get y las anotaciones de JPA que nos permiten persistirla en la base de datos . Evidentemente tendremos que configurar nuestro proyecto para que sea un proyecto Maven e incluya las dependencias necesarias.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.arquitecturajava</groupId>
        <artifactId>jpabasico</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <dependencies>

                <!--
https://mvnrepository.com/artifact/org.hibernate.javax.persistence/hib
ernate-jpa-2.1-api -->
                <dependency>
<groupId>org.hibernate.javax.persistence</groupId>
                        <artifactId>hibernate-jpa-2.1-api</artifactId>
                        <version>1.0.2.Final</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
                <dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-core</artifactId>
                        <version>5.4.11.Final</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>6.0.6</version>
                </dependency>
```

```
        </dependencies>
</project
```

Una vez tenemos todas las dependencias instaladas. Es momento de construir el fichero persistence.xml que quedará ubicado dentro de la carpeta /resources/META-INF

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
 <persistence-unit name="unidadLibros">
 <class>com.arquitecturajava.jpabasico.dominio.Libro</class>
 <properties>
 <property name= "hibernate.show_sql" value="true" />
 <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />
 <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
 <property name="javax.persistence.jdbc.user" value="root" />
 <property name="javax.persistence.jdbc.password" value="root" />
 <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost/jpa1" />
</properties>

 </persistence-unit>
</persistence>
```

JPA Remove y EntityManager

El fichero se encarga de configurar de forma rápida la conexión a la base de datos. Una vez

hecho esto es cuestión de construir un programa main que sea el encargado de realizar un JPA remove. La primera casuística es muy sencilla:

```
package com.arquitecturajava.jpabasico;

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

import com.arquitecturajava.jpabasico.dominio.Libro;

public class Principal3Borrar {

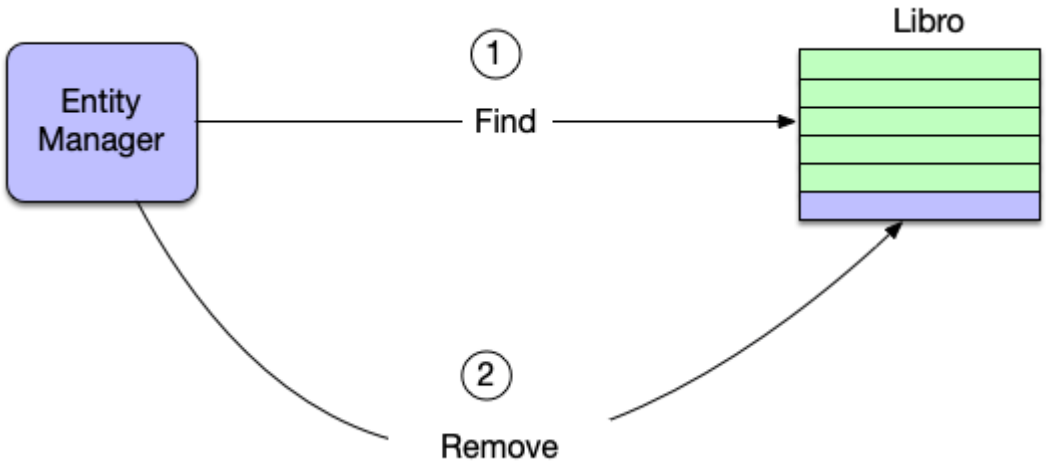
	public static void main(String[] args) {


		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("unidadLibros");
		EntityManager em = emf.createEntityManager();
		Libro miLibro = em.find(Libro.class, "1A");
		try {
			em.getTransaction().begin();
			em.remove(miLibro);
			em.getTransaction().commit();
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			em.close();

		}
```

```
        }
}
```

En este caso se usa el EntityManagerFactory para obtener la conexión a la base de datos y el EntityManager como gestor de entidades para buscar la entidad en la base de datos y abrir una transacción que se encargue de borrarla.



Si ejecutamos nuestro código este borrará un elemento de la base de datos. Hasta aquí todo correcto. Ahora bien que pasará cuando ejecutemos el código sin usar previamente el EntityManager para buscar la entidad .

```
package com.arquitecturajava.jpabasico;

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

import com.arquitecturajava.jpabasico.dominio.Libro;

public class Principal3Borrar2 {

        public static void main(String[] args) {
```

```
                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("unidadLibros");
                EntityManager em = emf.createEntityManager();
                Libro miLibro = new Libro("1A");
                try {
                        em.getTransaction().begin();
                        em.remove(miLibro);
                        em.getTransaction().commit();
                } catch (Exception e) {
                        e.printStackTrace();
                } finally {
                        em.close();

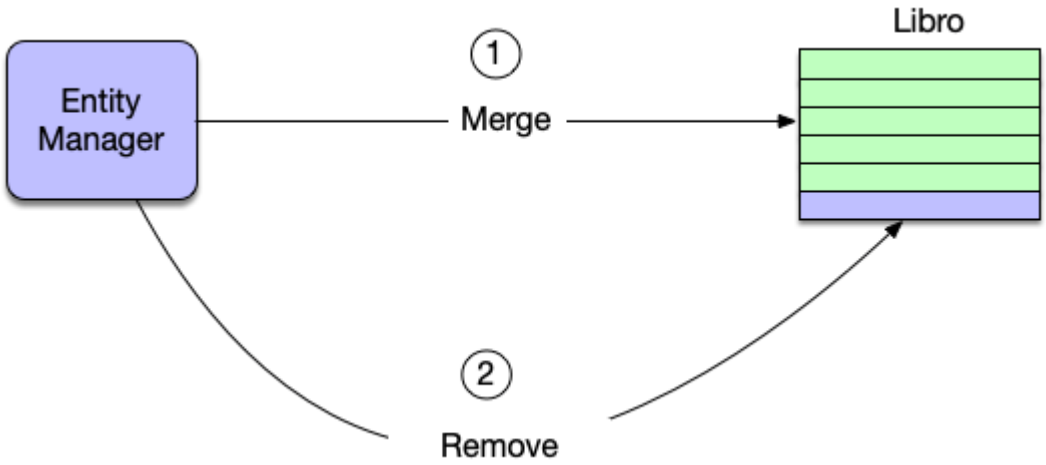
                }

        }
}
```

La realidad es que el código falla y se lanza una excepción.

JPA y Merge

Mucha gente cuando comienza con JPA no entiende muy bien que es lo que esta sucediendo. Sin embargo es totalmente normal ya que la entidad no esta siendo gestionada por el EntityManager previamente y una entidad que no este gestionada no puede ser borrada .



Para solventar este problema debemos usar el método Merge del API del EntityManager para previamente poner la entidad bajo el control de JPA

```
package com.arquitecturajava.jpabasico;

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

import com.arquitecturajava.jpabasico.dominio.Libro;

public class Principal5BorrarMerge {

	public static void main(String[] args) {

		Libro miLibro = new Libro("1A");

		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("unidadLibros");
		EntityManager em = emf.createEntityManager();
		try {
			em.getTransaction().begin();
```

```
                em.remove(em.merge(miLibro));
                em.getTransaction().commit();
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                em.close();

            }

    }
}
```

De esta forma si que JPA será capaz de eliminar la entidad de la base de datos .

Otros artículos relacionados

- Ejemplo JPA
- JPA EntityGraph
- Open Session in View
- Java Persistence API