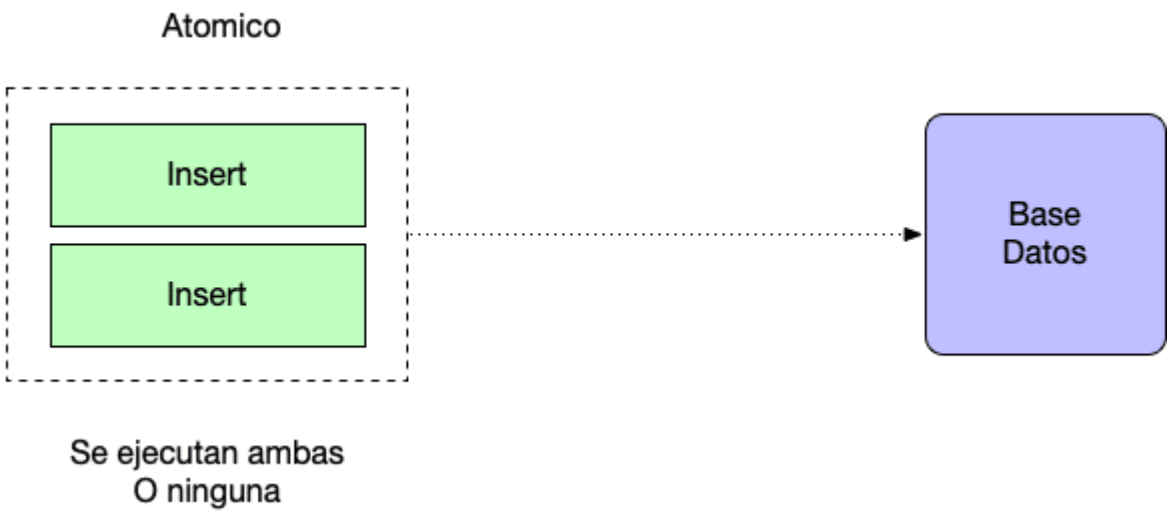
El concepto de JPA Transaction es uno de los conceptos claves cuando uno comienza a trabajar con JPA ( Java Persistence API) . Sin embargo una transacción es un concepto que pertenece más a las bases de datos que a JPA propiamente dicho .

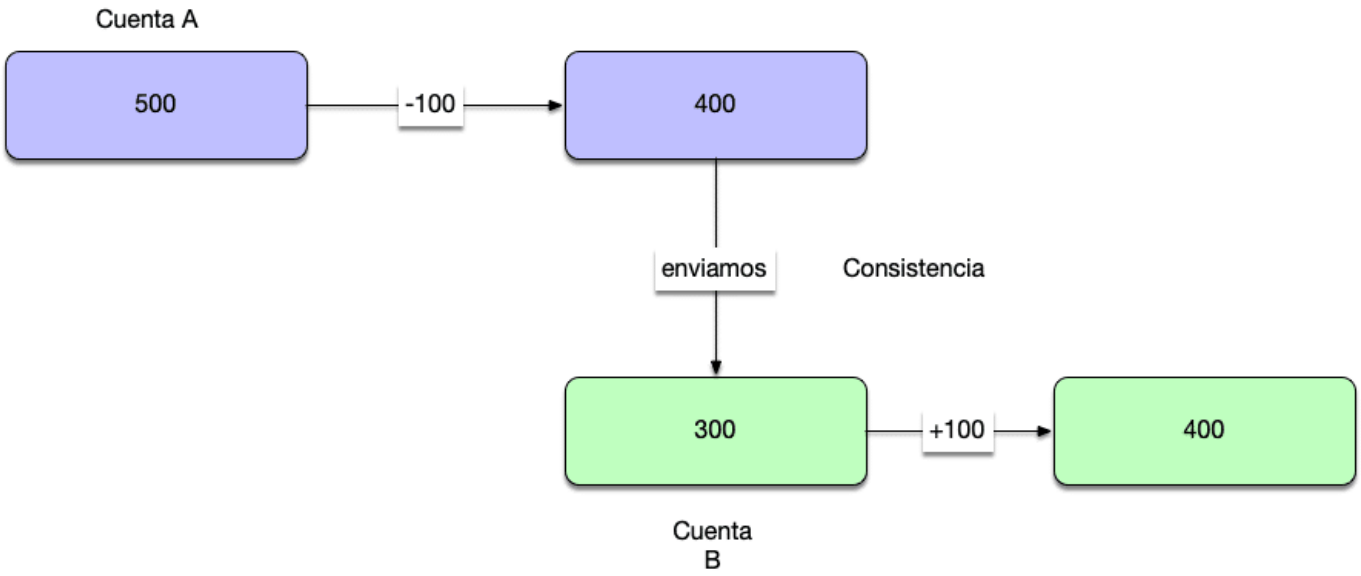
# ¿Que es exactamente una transacción?.

Una transacción hace referencia a una operación que se realiza contra una base de datos y que cumple con las propiedades ACID.

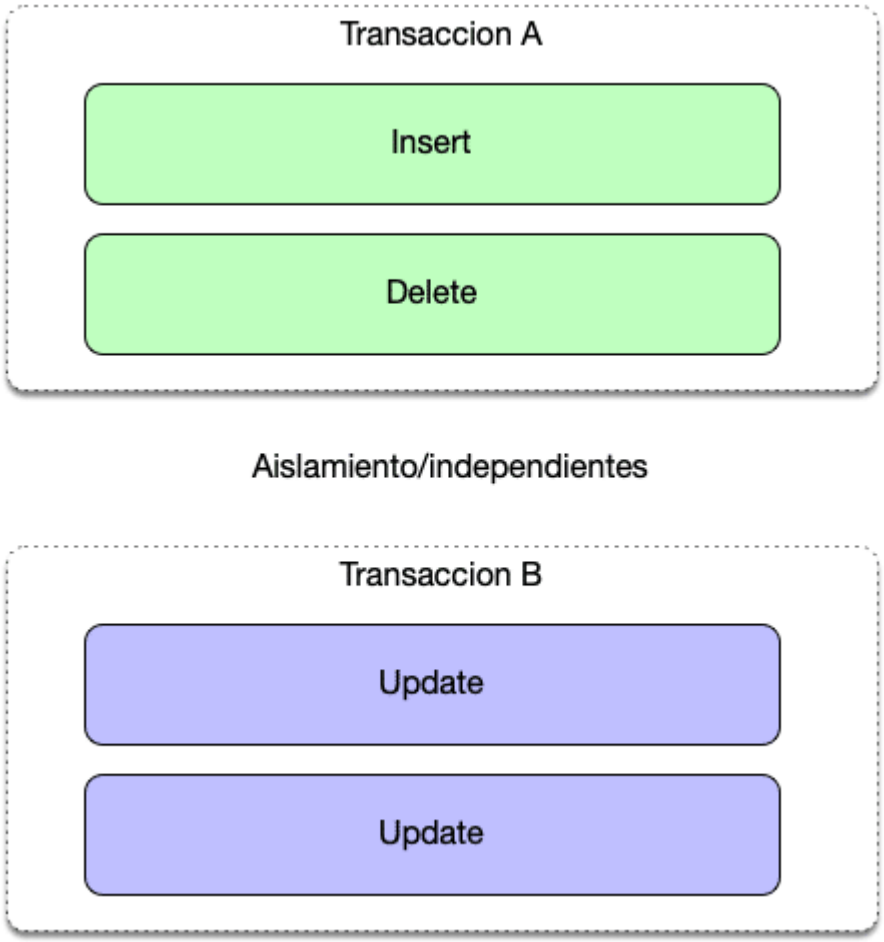
Atomic : Habla de cuando una operación es indivisible o se realiza de forma completa o no se realiza ninguna de sus partes . Esto es muy habitual en base de datos ya que si por ejemplo queremos realizar dos inserciones si las ejecutamos dentro de una transacción o se realizarán ambas o no se realizará ninguna. Cualquier fallo hará que la transacción se revierta y no se aplique ninguna modificación.




Consistente: Es decir si nosotros por ejemplo realizamos dos modificaciones en la base de datos una que me elimina 100 euros de mi saldo y otra que inserta 100 euros en otra cuenta . El concepto de consistencia hace referencia a que los números deben cuadrar si resto 100 euros de mi cuenta en la otra deben de entrar 100 no pueden entrar 70.




Isolated: Aislada , la operación que nosotros realizamos no puede verse afectada por otras operaciones que se estén ejecutando en ese momento




Durable: La información queda grabada en la base de datos para el futuro y persiste

# JPA Transaction

Vamos a manejar estos conceptos dentro de JPA . Para ello nos vamos a crear una clase que podamos persistir en la base de datos

```
package com.arquitecturajava.dominio;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
```

```
@Entity
@Table(name = "libros")
public class Libro {

  @Id
  private String isbn;
  private String titulo;
  private String autor;

  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 Libro(String isbn, String titulo, String autor) {
    super();
    this.isbn = isbn;
    this.titulo = titulo;
```

```
  this.autor = autor;
}
public Libro() {
  super();
}
public Libro(String isbn) {
  super();
  this.isbn = isbn;
}
@Override
public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + ((isbn == null) ? 0 : isbn.hashCode());
  return result;
}
@Override
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
    return false;
  Libro other = (Libro) obj;
  if (isbn == null) {
    if (other.isbn != null)
      return false;
  } else if (!isbn.equals(other.isbn))
    return false;
  return true;
```

```
  }
}
```

Una vez construida esta clase deberemos generar el Persistence.xml

# Persistence.xml

```
<?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="biblioteca" transaction-
type="RESOURCE_LOCAL">
        <class>com.arquitecturajava.dominio.Libro</class>
        <properties>
                <property name= "hibernate.show_sql" value="true" />
                <!-- cuando nos conectamos desde JPA hay  q elegir el
tipo de motor -->
      <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.url"
value="jdbc:mysql://localhost:8889/biblioteca" />
            <property name="javax.persistence.jdbc.user" value="root"
/>
            <property name="javax.persistence.jdbc.password"
value="root" />
        </properties>
    </persistence-unit>
```

```
</persistence>
```

Es momento de construir el programa Main y definir un EntityTransaction a la hora de persistir los datos en la base de datos:

```
package com.arquitecturajava.main;

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

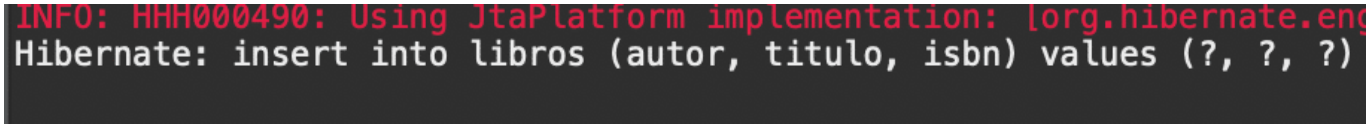
import com.arquitecturajava.dominio.Libro;

public class Principal {

  public static void main(String[] args) {
    // unidad de persistencia
    EntityManagerFactory emf=
Persistence.createEntityManagerFactory("biblioteca");
    EntityManager em= emf.createEntityManager();
    Libro libro= new Libro("1A","java","pedro");
    EntityTransaction t= em.getTransaction();
    try {
      t.begin();
      em.persist(libro);
      t.commit();
    } catch (Exception e) {
      // TODO Auto-generated catch block
      t.rollback();
    }
  }
```

```
}
```

Una vez tenemos todo construido ejecutamos el código y nos insertará un registro en la base de datos :




Acabamos de insertar una nueva fila en la base de datos

# JPA Transaction y rollbacks

Es momento de realizar la misma operación pero de una forma más transaccional por ejemplo podemos solicitar que nuestro código inserte dos libros.

```
package com.arquitecturajava.main;

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

import com.arquitecturajava.dominio.Libro;

public class Principal {

  public static void main(String[] args) {
    // unidad de persistencia
    EntityManagerFactory emf=
Persistence.createEntityManagerFactory("biblioteca");
    EntityManager em= emf.createEntityManager();
```

```
    Libro libro2= new Libro ("3A","JPA","Gema");
    Libro libro3= new Libro ("4A","Java","David");
    EntityTransaction t= em.getTransaction();
    try {
      t.begin();
      em.persist(libro2);
      em.persist(libro3);
      t.commit();
    } catch (Exception e) {
      // TODO Auto-generated catch block
      t.rollback();
    }
  }

}
```

Todo sigue funcionando correctamente . La gestión transaccional la podemos ver con mayor claridad cuando insertamos otros libros pero alguno de ellos previamente existe.

```
package com.arquitecturajava.main;

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

import com.arquitecturajava.dominio.Libro;

public class Principal {

  public static void main(String[] args) {
    // unidad de persistencia
```

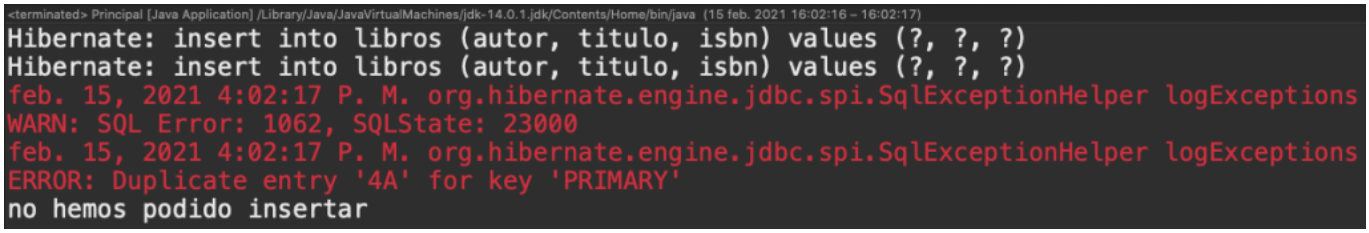
```
    EntityManagerFactory emf=
Persistence.createEntityManagerFactory("biblioteca");
    EntityManager em= emf.createEntityManager();
    Libro libro2= new Libro ("6B","Spring","Gema");
    Libro libro3= new Libro ("4A","Java","David");
    EntityTransaction t= em.getTransaction();
    try {
      t.begin();
      em.persist(libro2);
      em.persist(libro3);
      t.commit();
    } catch (Exception e) {
      // TODO Auto-generated catch block
      t.rollback();
      System.out.println("no hemos podido insertar");
    }
  }

}
```

En este caso el Libro 6B no existe en la base de datos y deberíamos ser capaces de insertarlo . Pero estamos dentro de una transacción por lo tanto si el insert de la siguiente linea 4A no se puede insertar porque ya existe , nos encontraremos que la operación es atómica y ninguna de las dos registros se persiste.




Acabamos de hacer una operación de rollback.




Acabamos de ejecutar una operación transaccional sobre la base de datos . En la cual si algo falla todo los cambios se desechan o se vuelven para atrás ya que es una operación atómica e indivisible.

## Otros artículos relacionados

- JPA @OneToOne y relaciones 1 a 1
- JPA Object States y como usarlos
- Java Optional Repository y JPA
- Curso JPA
- Spring Boot JPA y su configuración