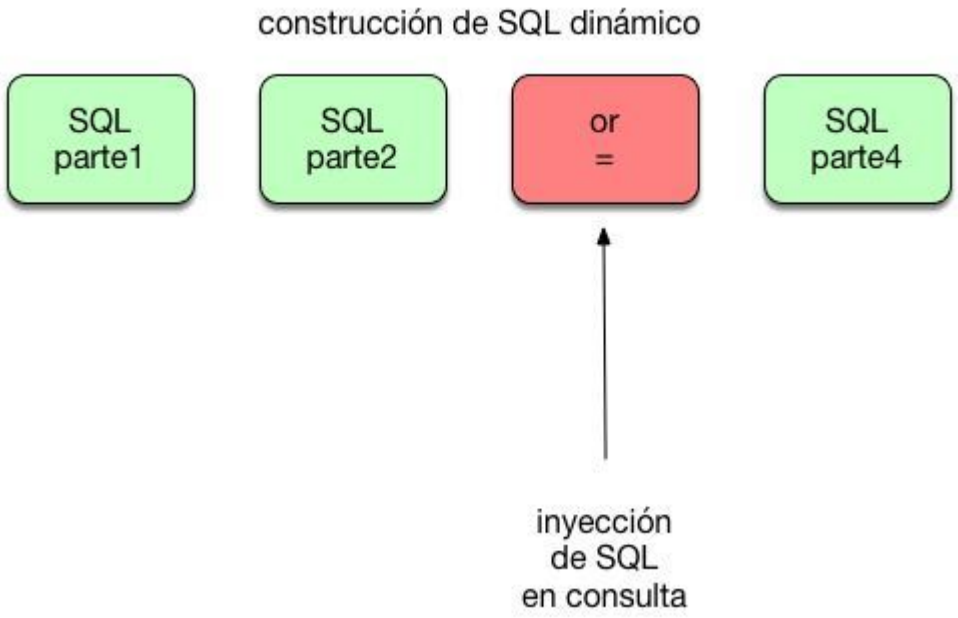
**CURSO JPA**
**GRATIS**
**APUNTATE!!**

El uso de JPA Criteria nos aporta muchas ventajas en cuanto a la construcción de SQL Dinámico utilizando JPA . Ahora bien su su sintaxis y su forma de trabajar son bastante diferentes a la forma clásica. Vamos a apoyarnos en el ejemplo anterior de JPA SQL Injection para construir el mismo ejemplo utilizando JPA Criteria . En el ejemplo anterior teníamos el siguiente código:

```
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) {
                seleccionarPersona("pedro", "perez");
        }
        private static void seleccionarPersona(String nombre, String
apellidos) {
                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadPersonas");
                EntityManager em = emf.createEntityManager();
                try {
                        String sql = "select p from Persona p ";
                        if (nombre != null || apellidos != null) {
```

```
                                sql += " where ";
                        }
                        if (nombre != null) {
                                sql +=  " nombre='" + nombre + "'";
                        }
                        if (apellidos != null) {
                                if(nombre!=null) {
                                        sql += " and  apellidos='" +
apellidos + "'";
                                }else{
                                        sql += "  apellidos='" +
apellidos + "'";
                                }
                        }
                        TypedQuery<Persona&amp;amp> consulta =
em.createQuery(sql, Persona.class);
                        List<Persona&amp;amp> lista =
consulta.getResultList();
                        lista.forEach((p) -&amp;amp> {
                                System.out.println(p.getNombre());
                        });
                } catch (Exception e) {
                        e.printStackTrace();
                } finally {
                        em.close();
                }
        }
}
```

El código nos generaba una SQL dinámica pero ya habíamos visto que si cambiamos los parámetros e intentabamos inyectar SQL , podríamos generar un agujero de seguridad.




# JPA Criteria API

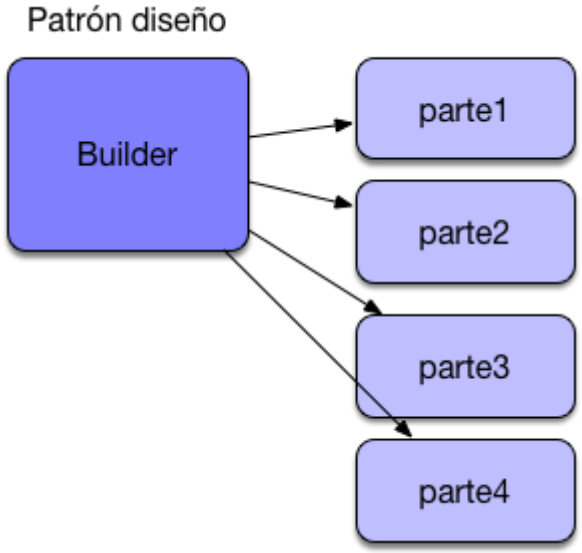
Para solventar este problema que se genera a partir de la necesidad de construir SQL Dinámico podemos recurrir al API de Criteria y diseñar la consulta con ella.

```
ppackage com.arquitecturajava;
pimport java.util.List;
pimport javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;
ppublic class Principal2 {
public static void main(String[] args) {
        seleccionarPersonas("pedro","perez");
}
```

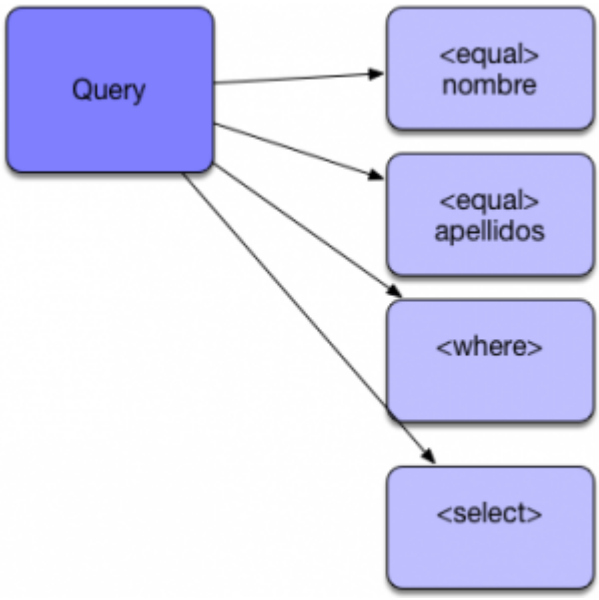
```
private static void seleccionarPersonas(String nome, String apellidos)
{
                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("UnidadPersonas");
                EntityManager em = emf.createEntityManager();
                try {
                CriteriaBuilder cb = em.getCriteriaBuilder();
                CriteriaQuery<Persona&amp;amp> consulta =
cb.createQuery(Persona.class);
                Root<Persona&amp;amp> personas =
consulta.from(Persona.class);
                        Predicate p1 = null,p2 = null;
                if(nome!=null) {
                         p1= cb.equal(personas.get("nome"),"pedro");
                        }
                if(apellidos!=null) {
                         p2=
cb.equal(personas.get("apellidos"),"perez");
                        }
                Predicate nomeApellidos=cb.or(p1,p2);
                consulta.select(personas).where(nomeApellidos);
                List<Persona&amp;amp> lista =
em.createQuery(consulta).getResultList();
                        lista.forEach((p) -&amp;amp> {
                        System.out.println(p.getNome());
                });
                } catch (Exception e) {
                e.printStackTrace();
                } finally {
                        em.close();
        }
```

```
        }
}
```

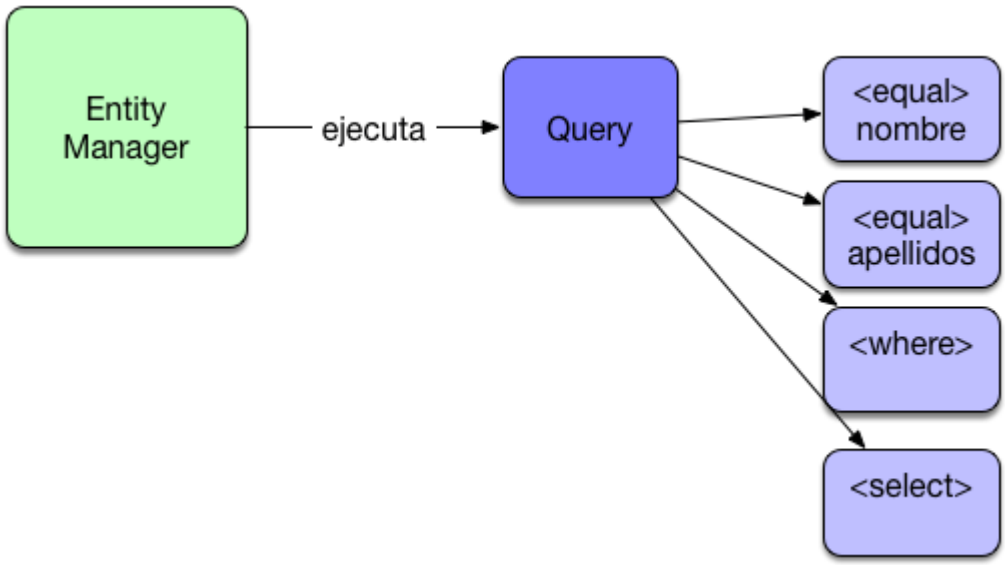
¿Como funciona exactamente el API de Criteria? . Este API se encarga de diseñar la consulta a través de un Builder, un patrón de diseño que construye un objeto paso a paso




A través de estos pasos se construye el objeto por complejo verificando que todo es correcto. Este es el enfoque JPA Criteria API , vamos construyendo poco a poco cada trazo.



Para después ejecutar esa consulta construida utilizando el EntityManager:




De tal forma que si invocamos la consulta y pasamos los siguientes parámetros

seleccionarPersonas(“pedro”,”perez”);

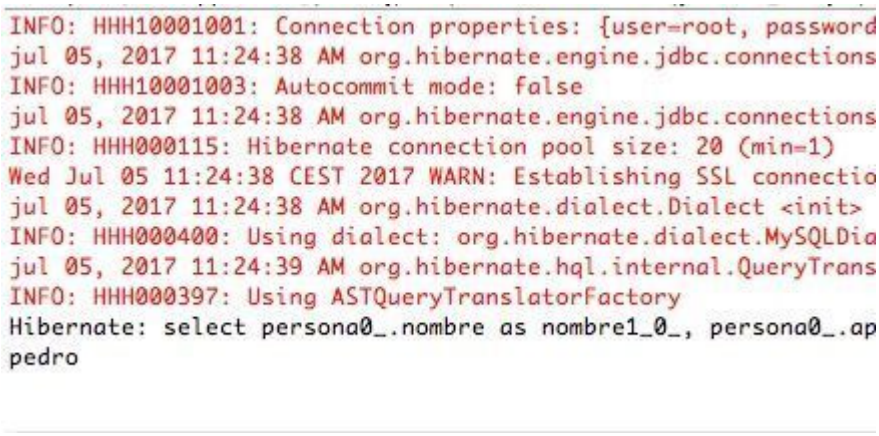
La consulta se ejecutara sin problemas :

```
INFO: HHH10001001: Connection properties: {user=root, password
jul 05, 2017 11:24:38 AM org.hibernate.engine.jdbc.connections
INFO: HHH10001003: Autocommit mode: false
jul 05, 2017 11:24:38 AM org.hibernate.engine.jdbc.connections
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Wed Jul 05 11:24:38 CEST 2017 WARN: Establishing SSL connectio
jul 05, 2017 11:24:38 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDia
jul 05, 2017 11:24:39 AM org.hibernate.hql.internal.QueryTrans
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select persona0_.nombre as nombre1_0_, persona0_.ap
pedro
```

Como el API de Criteria valida cada parámetro , tampoco tendremos problemas si nos inyectan código no válido.

seleccionarPersonas(“pedro”,“”='”);

El Api de Criteria se encargará de revisarlo y generar la consulta parametrizada correcta.




No sufriremos una inyección de SQL.

Otros artículos relacionados:

1. JPA Composite Key y business objects
2. JPA Single Table Inheritance
3. Utilizando JPA NamedQueries
4. Hibernate Criteria