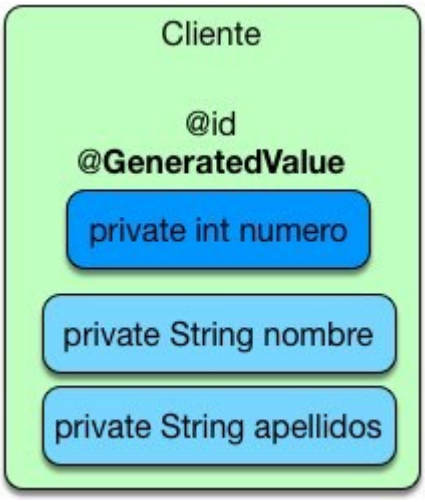
Usar la anotacion @GeneratedValue con JPA es algo bastante habitual ya que existen muchas tablas cuyas claves primarias preferimos que sean autoincrementales. Así pues solemos configurar las entidades de JPA para que usen estas anotaciones.



Vamos a ver un ejemplo y profundizar un poco en su funcionamiento. Para ello vamos a definir las dependencias del proyecto vía Maven:

```
        <dependency>
<groupId>org.hibernate.javax.persistence</groupId>
                        <artifactId>hibernate-jpa-2.1-api</artifactId>
                        <version>1.0.0.Final</version>
                </dependency>
                <dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-core</artifactId>
                        <version>5.2.11.Final</version>
                </dependency>
```

```
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.1.44</version>
		</dependency>

		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>4.12</version>
			<scope>test</scope>
		</dependency>
```

Hecho esto configuramos el fichero de persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
version="2.0">
	<persistence-unit name="curso">
		<properties>
			<property name="hibernate.show_sql"
value="true" />
			<property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL57Dialect" />
			<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="mysql" />
                        <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost/arquitectura" />

                        <property name="javax.persistence.schema-
generation.database.action" value="drop-and-create" />
                </properties>
        </persistence-unit>
</persistence>
```

JPA @GeneratedValue

El siguiente paso es generar una Entidad que use la anotación , en nuestro caso la clase Cliente.

```
package com.arquitecturajava.jpa1;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Cliente {

        @Id
```

```
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private long numero;
	private String nombre;
	private String apellidos;
	public long getNumero() {
		return numero;
	}
	public String getNombre() {
		return nombre;
	}
       public void setNumero(long numero) {
		this.numero = numero;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public Cliente(String nombre, String apellidos) {
		super();
		this.nombre = nombre;
		this.apellidos = apellidos;
	}
}
```

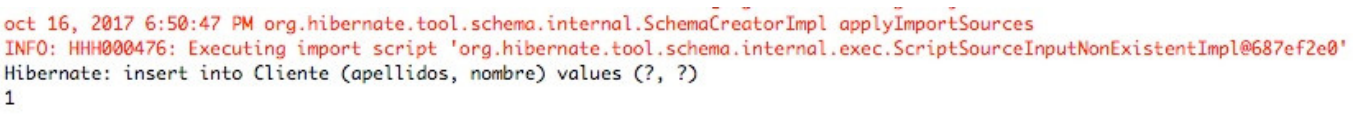
Acabamos de utilizar la anotación @GeneratedValue para obligar a que el campo sea autoincremental . Hemos eliminado el parámetro del constructor . Es momento de crear un

programa principal que inserte un cliente en la base de datos.

```
package com.arquitecturajava.main;

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

import com.arquitecturajava.jpa1.Cliente;

public class Principal {

	public static void main(String[] args) {

		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("curso");
		EntityManager em = emf.createEntityManager();
		Cliente c=new Cliente ("pepe","gomez");
		em.getTransaction().begin();
		em.persist(c);
		em.getTransaction().commit();
		System.out.println(c.getNumero());

	}

}
```

Esto lanzará una consulta de inserción sobre la base de datos que nos insertará el Cliente

con la clave primaria autogenerada. Si revisamos la consola nos sorprenderá ver lo siguiente:



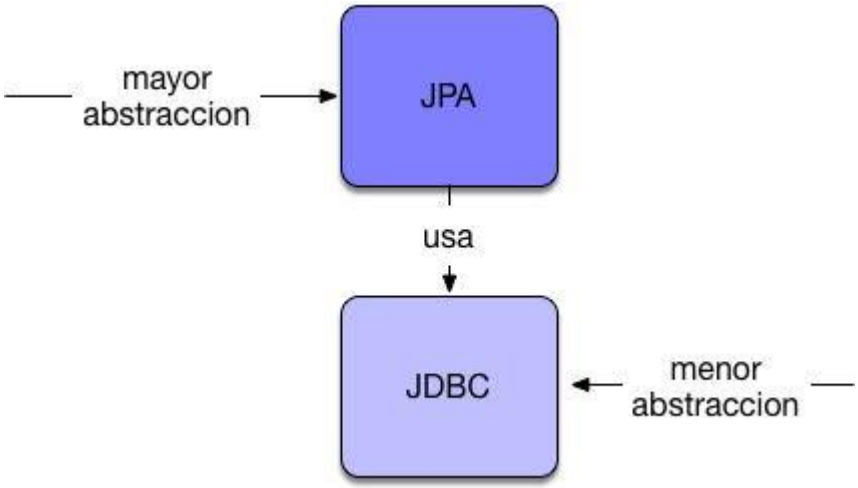
El código es correcto se inserta el cliente y la primary key es generada automaticamente por lo tanto no se insertan 3 columnas sino 2. Ahora bien en cuanto leo el numero del cliente el programa Java me devuelve “1”. El dato es correcto pero únicamente hemos realizado la consulta de inserción . ¿Como es posible que ya tengamos el campo relleno sin realizar una select?.

JPA y JDBC

Estos es debido a que el API de JDBC a nivel de driver permite una vez realizada la inserción con un insert acceder directamente a la primary key generada sin tener que realizar consultas adicionales usando el método getGeneratedKeys() del statement.

```
statement.getGeneratedKeys();
```

Recordemos que JPA es una capa de abstracción sobre JDBC por lo tanto el API se apoya en las ventajas que aporta el Driver.



Esto nos hace realmente sencilla la gestión de primary keys autoincrementales.

Otros artículos relacionados

1. JPA Lazy fetching proxies y rendimiento
2. JPA Criteria API , un enfoque diferente
3. JPA SQL Injection y sus problemas
4. JPA Maven