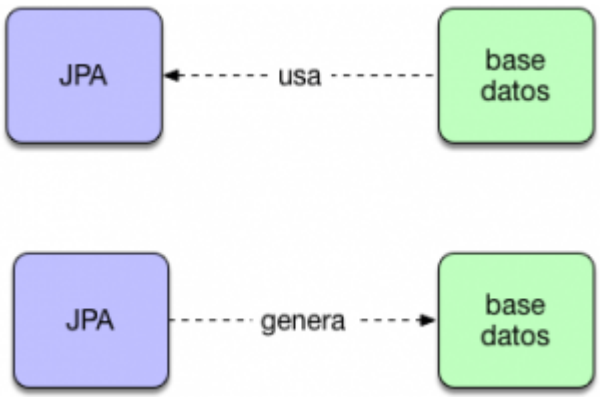
El uso de un JPA Database schema es bastante habitual . Cuando trabajamos con JPA nos podemos encontrar con dos opciones principales. La primera que el schema de base de datos ya exista y que nosotros únicamente tengamos que adaptarnos a él. En segundo lugar una situación en la que no existe schema de base de datos y utilizaremos nuestros objetos de negocio para generar dicho schema.



JPA DataBase Schema y objetos

Vamos a abordar un ejemplo que se encargue de construir un JPA Database Schema a partir de los objetos de negocio que diseñamos con nuestras clases Java . Para ello el primer paso será configurar las dependencias de maven que el proyecto necesita:

```
<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>5.2.10.Final</version>
</dependency>
 <dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <version>5.1.6</version>
 </dependency>
 </dependencies>
```

El siguiente paso es diseñar los objetos de negocio que vamos a utilizar en nuestro modelo:

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;

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

@Entity
public class Persona {
 @Id
 private String nombre;
 private String apellidos;
 private int edad;

 @OneToMany(mappedBy="persona")
 private List<Telefono> telefonos= new ArrayList<Telefono>();
```

```
public void addTelefono(Telefono telefono) {
telefonos.add(telefono);
}

public List<Telefono> getTelefonos() {
return telefonos;
}
public void setTelefonos(List<Telefono> telefonos) {
this.telefonos = telefonos;
}
public Persona() {
super();
}
public Persona(String nombre, String apellidos, int edad) {
super();
this.nombre = nombre;
this.apellidos = apellidos;
this.edad = edad;
}
public String getNombre() {
return nombre;
}
public void setNombre(String nombre) {
this.nombre = nombre;
}
public String getApellidos() {
return apellidos;
}
public void setApellidos(String apellidos) {
this.apellidos = apellidos;
}
```

```
 public int getEdad() {
 return edad;
 }
 public void setEdad(int edad) {
 this.edad = edad;
 }
}
```

```
package com.arquitecturajava;

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

@Entity
public class Telefono {
 @Id
 private String marca;
 private int nserie;
 @ManyToOne
 private Persona persona;

 public String getMarca() {
 return marca;
 }
 public void setMarca(String marca) {
 this.marca = marca;
 }
 public int getNserie() {
 return nserie;
```

```
 }
 public void setNserie(int nserie) {
 this.nserie = nserie;
 }


}
```

Una vez que tenemos el modelo de objeto nos queda configurar el persistence.xml para que soporte la generación de schemas.

```
<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="arquitecturajava2">

 <properties>
 <property name="hibernate.show_sql" value="true" />
 <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5Dialect" />
 <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:3306/java2" />
```

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


 </properties>
 </persistence-unit>

</persistence>
```

En este caso lo hemos configurado para que borre (drop) y vuelve a crear todas las tablas. Para ello hemos tenido que añadir la propiedad javax.persistence.schema-generation.database.action. Una vez hecho esto es suficiente con crear un programa main muy sencillo que invoque el método generate schema:

```
package com.arquitecturajava;


import javax.persistence.Persistence;

public class Principal {
        public static void main(String[] args) {
                Persistence.generateSchema("arquitecturajava2",null);
        }
}
```

Una vez ejecutado el programa veremos como se lanzan las diferentes consultas SQL

```
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
may 31, 2017 11:56:49 AM org.hibernate.engine.jdbc.env.internal.LobCreatorBuilderImpl useContextualLobCreation
INFO: HHH000423: Disabling contextual LOB creation as JDBC driver reported JDBC version [3] less than 4
Hibernate: drop table if exists Persona
may 31, 2017 11:56:49 AM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolate
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentIniti
Hibernate: drop table if exists Telefono
Hibernate: create table Persona (nombre varchar(255) not null, apellidos varchar(255), edad integer not null, primary key (nor
may 31, 2017 11:56:49 AM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolate
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentIniti
Hibernate: create table Telefono (marca varchar(255) not null, nserie integer not null, telefono_marca varchar(255), primary
Hibernate: alter table Telefono add constraint FKdr2o1gcjybl1rw0na28ev8qfi foreign key (telefono_marca) references Telefono (
may 31, 2017 11:56:49 AM org.hibernate.tool.schema.internal.SchemaCreatorImpl applyImportSources
INFO: HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.ScriptSourceInputNonExistentImpl@7e990ed7'
```

Como resultado se generarán las diferentes tablas en la base de datos:

```
mysql> show tables;
+----------------+
| Tables_in_java2 |
+----------------+
| Persona        |
| Telefono       |
+----------------+
2 rows in set (0,00 sec)
```

Otros artículos relacionados:

1. JPA Proxy y su funcionamiento
2. Un ejemplo de JPA embedded objects
3. Utilizando JPA NamedQueries
4. Oracle Persistence