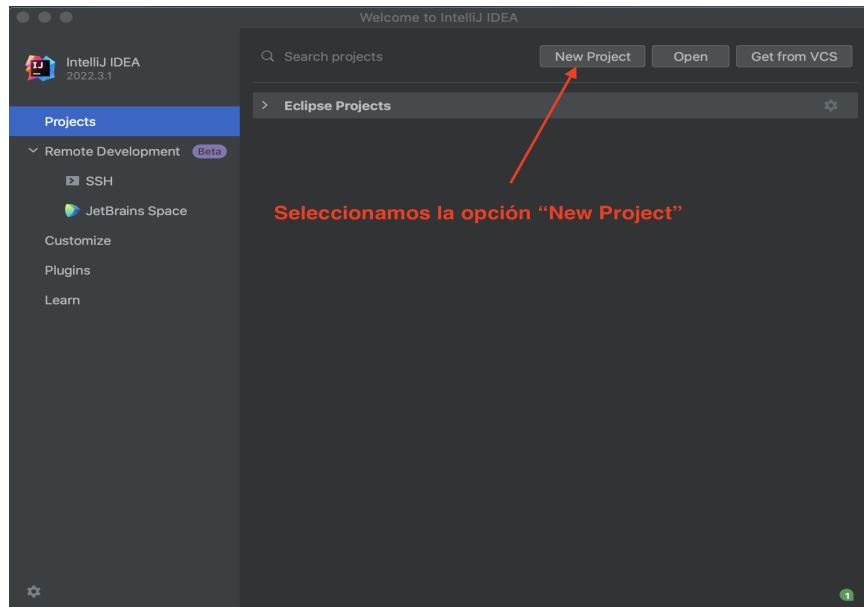


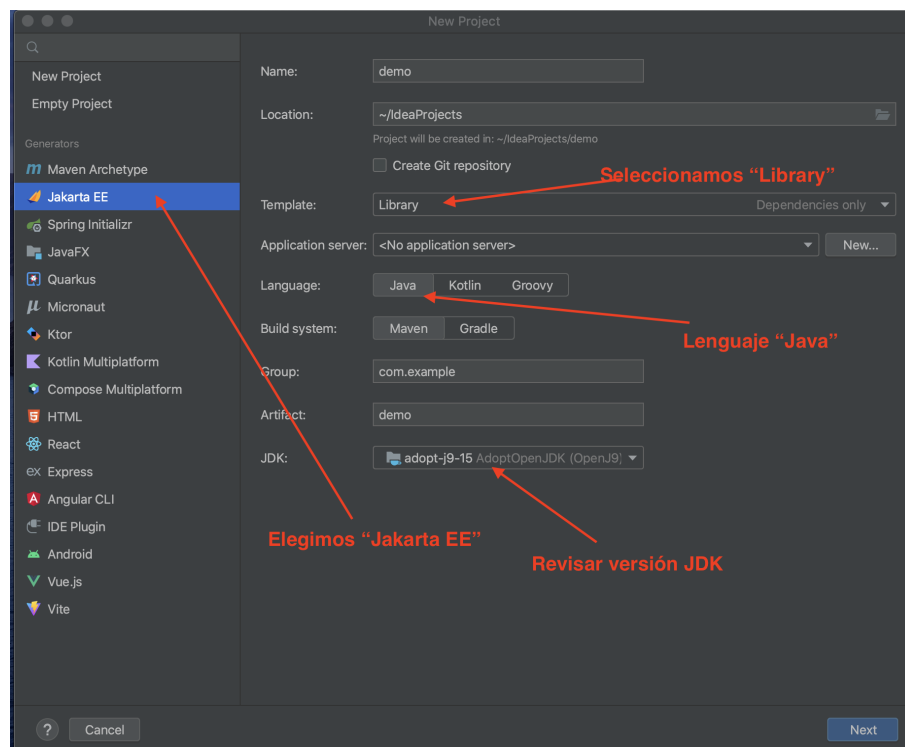
IntelliJ + Hibernate

1. Crear un proyecto usando el ORM **Hibernate** en el entorno de desarrollo **IntelliJ**

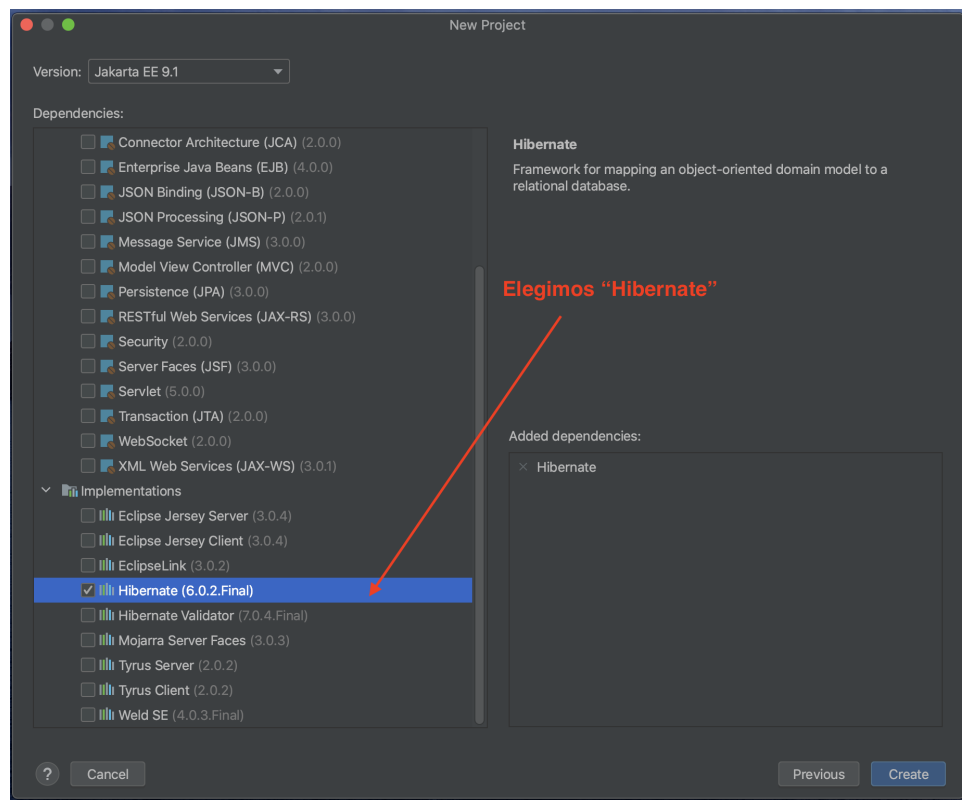
Paso 1. Seleccionamos la opción de “Nuevo proyecto”.



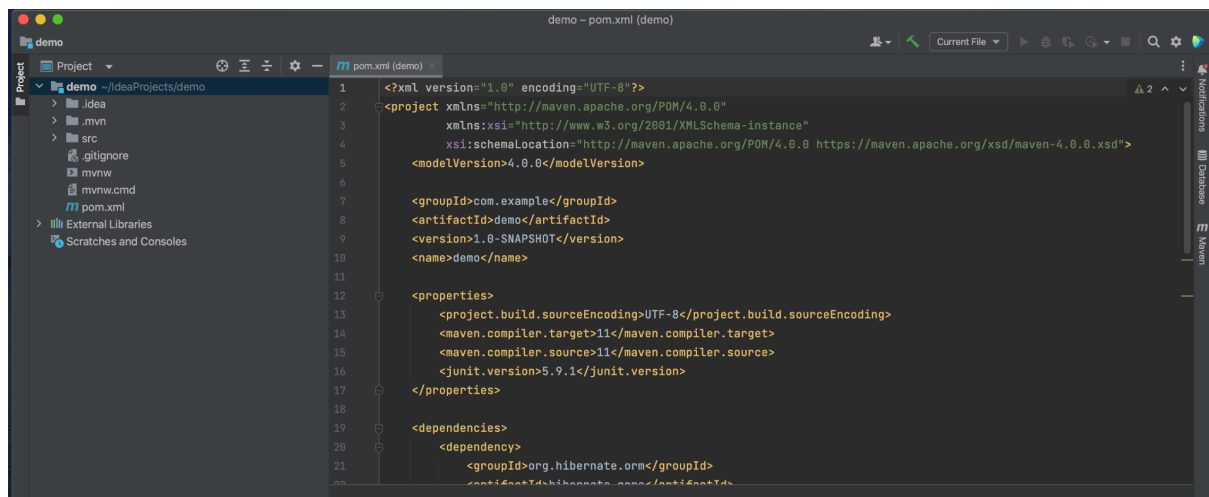
Paso 2. Marcamos las diferentes opciones del proyecto.



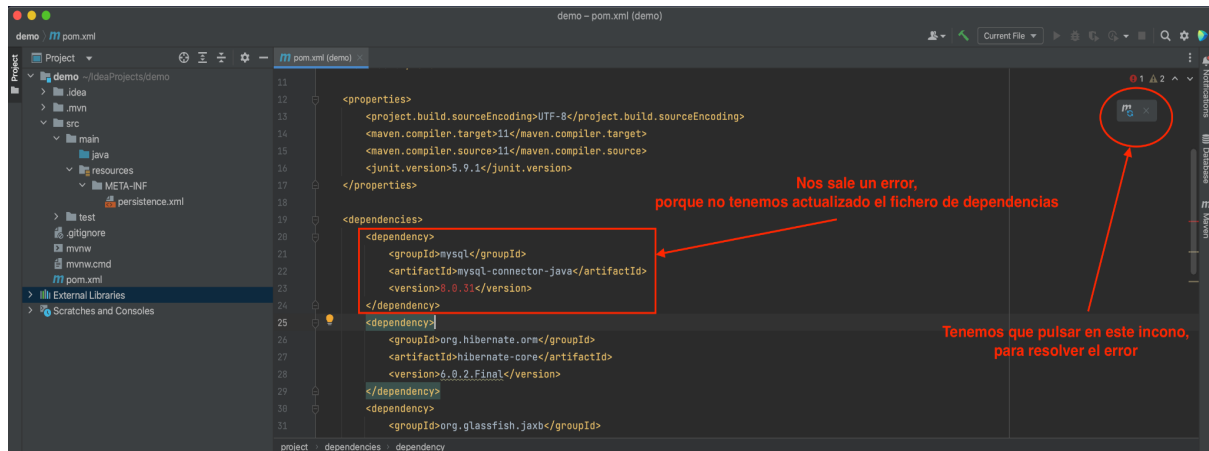
Paso 3. Añadimos la dependencia "Hibernate".



Y ya tendríamos creado nuestro proyecto.

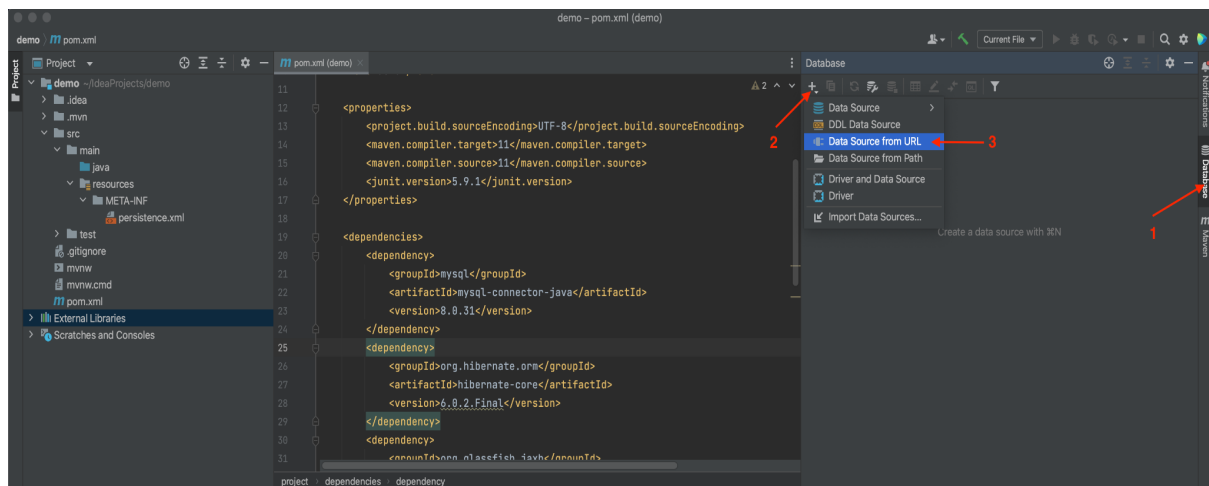


2. Añadimos la dependencia del conector a MySQL

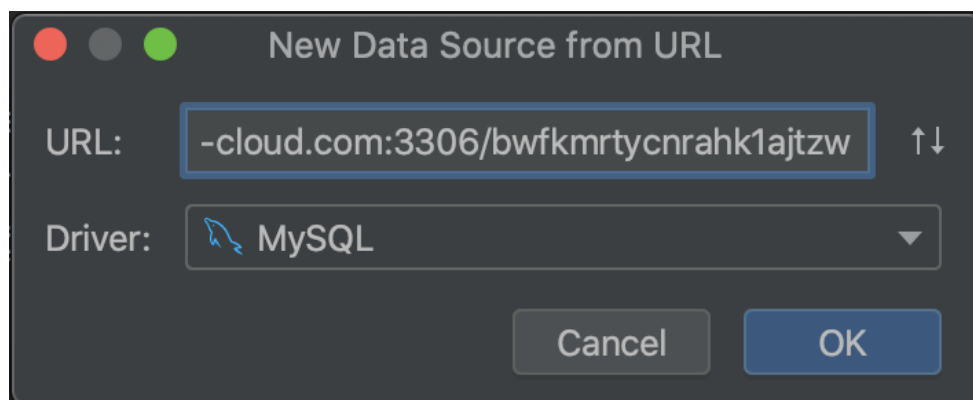


3. Creamos la conexión a la BBDD

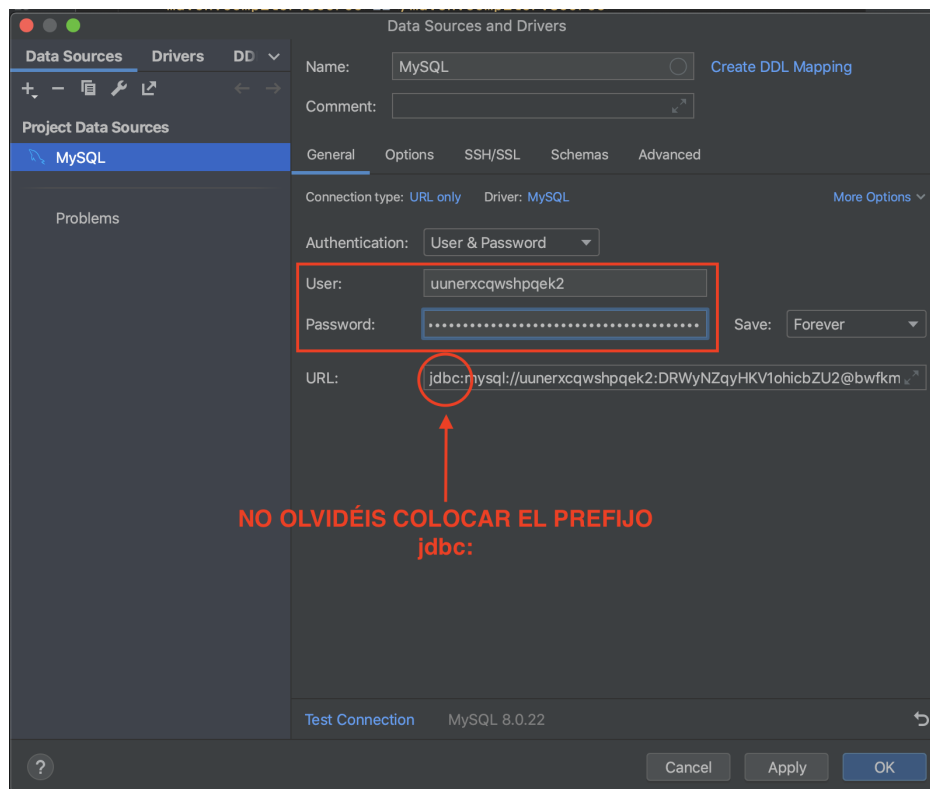
Paso 1. Escogemos la opción "Data Source from URL"



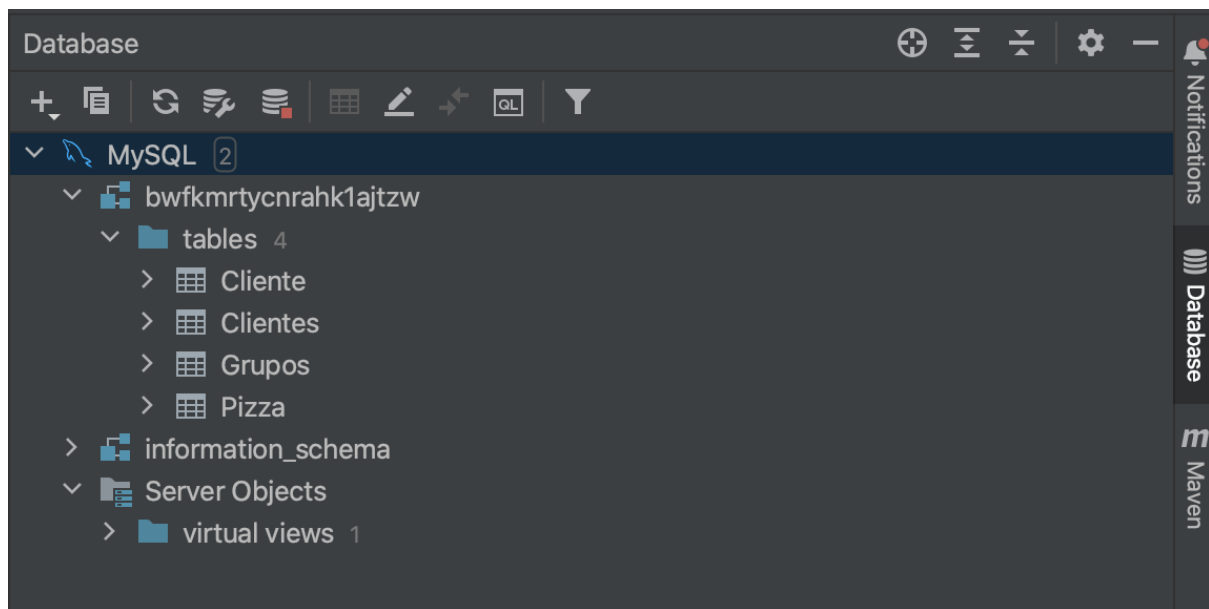
Paso 2. Insertamos la URL de conexión a nuestra BBDD



Paso 3. Introducimos usuario y contraseña.

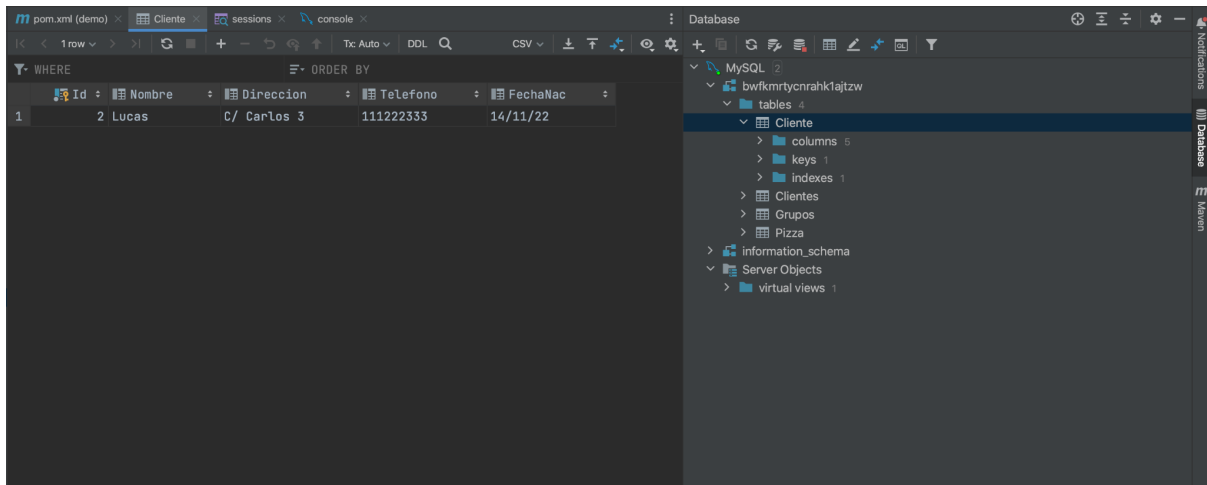


Y ya podemos ver en la ventana **“Database”** toda la información sobre la BBDD a la que nos hemos conectado.



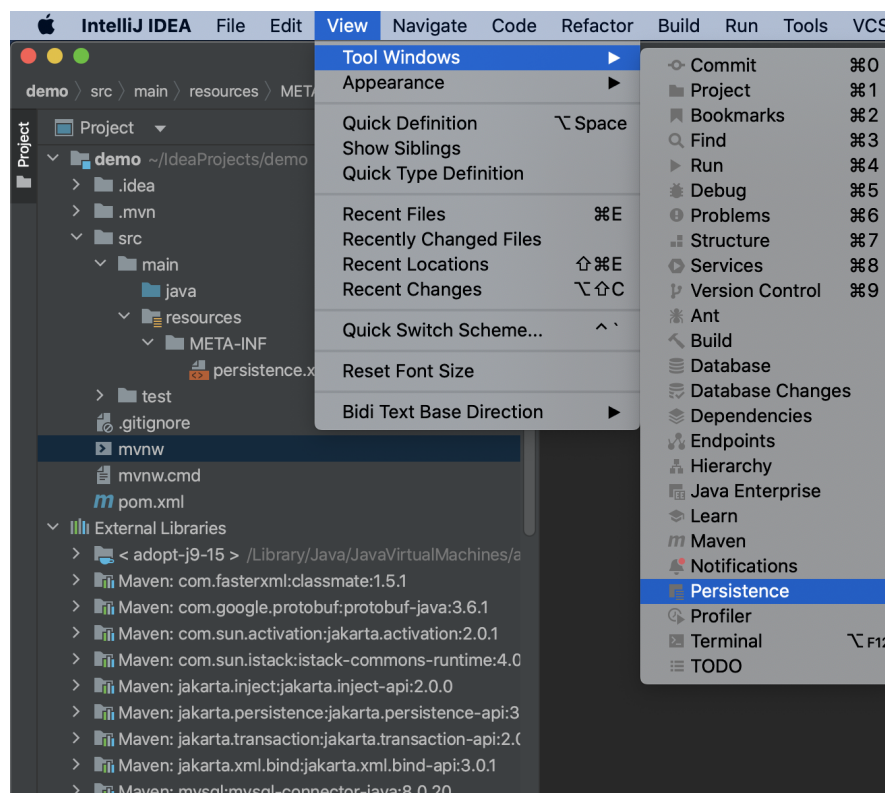
En este caso, vamos a trabajar únicamente con la tabla **“Clientes”**.

Si pinchamos dos veces sobre el icono junto al nombre “**Ciente**”, podremos ver el contenido de la misma.

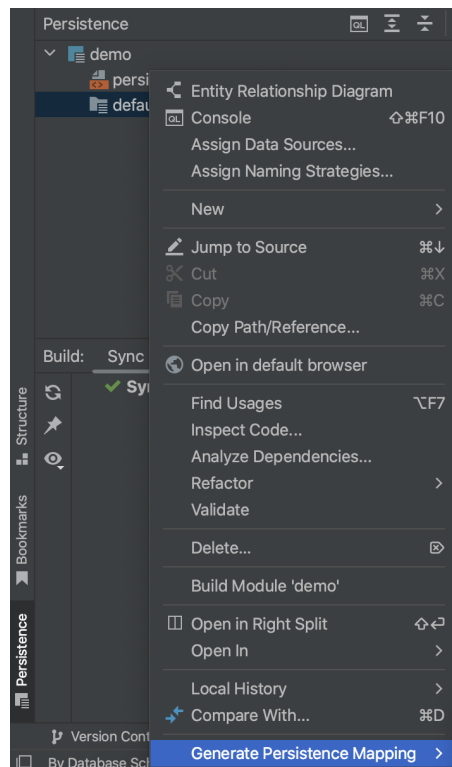


4. Generamos las entidades usando el esquema de la BBDD

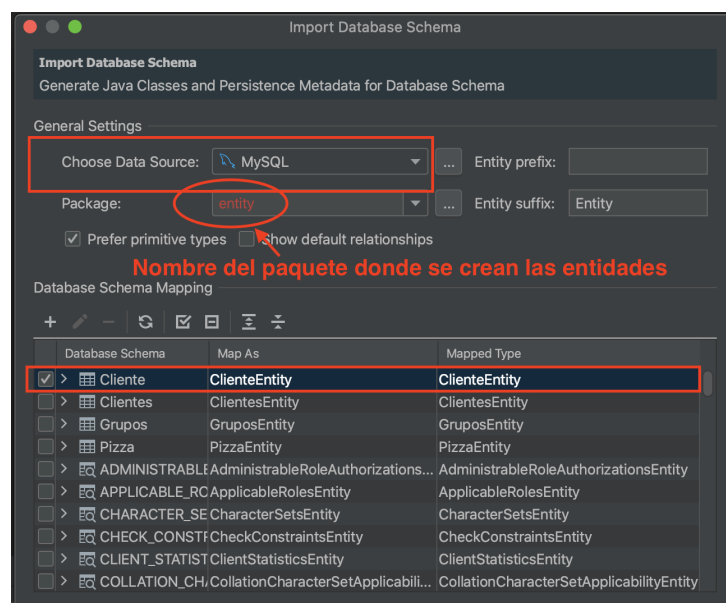
Paso 1. Seguimos la siguiente ruta para que se nos muestre la ventana de “Persistencia”.



Paso 2. Pinchamos con el botón derecho sobre la carpeta “default” y elegimos la ruta que se ve en la imagen.

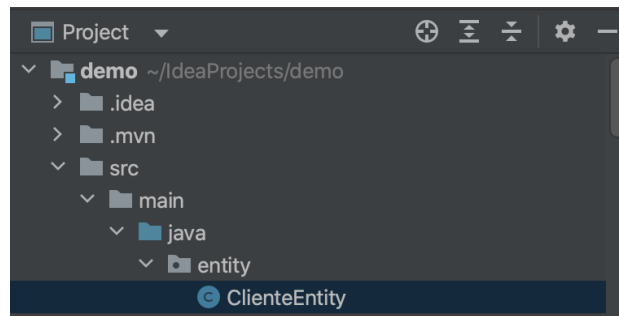


Paso 3. En la ventana emergente que nos aparece, seleccionamos como fuente “MySQL” y marcamos únicamente la tabla “Cliente”. Cuidado porque el package entity debe estar creado previamente en /src/main/java



Al pulsar **OK** se nos genera la entidad. En la imagen no se ve porque se ha recortado para que quepa en esta hoja.

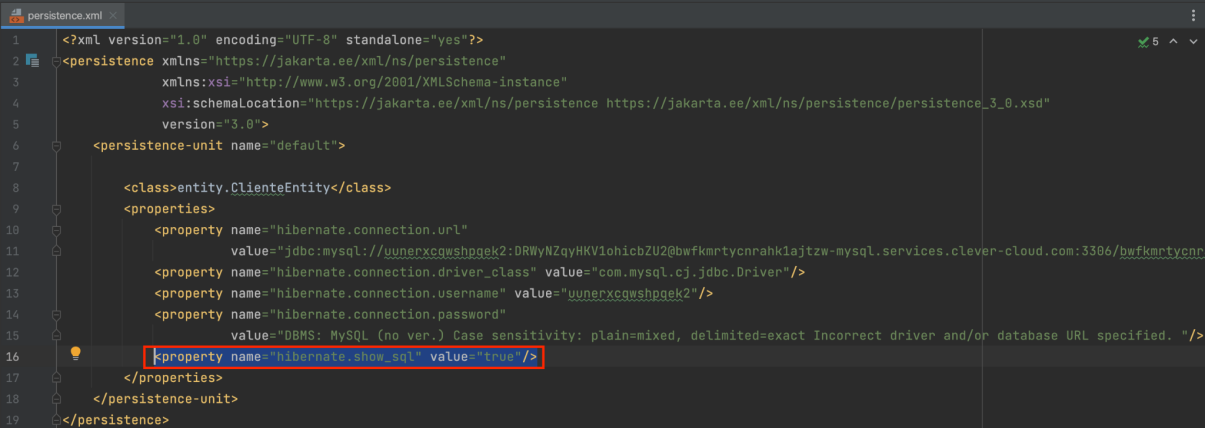
Y aquí podemos ver cómo ya se ha generado la clase **ClienteEntity**.



```
1 package entity;
2
3 import jakarta.persistence.*;
4
5 @Entity
6 @Table(name = "Cliente", schema = "bwfkmrtycnrahk1ajtzw", catalog = "")
7 public class ClienteEntity {
8     @GeneratedValue(strategy = GenerationType.IDENTITY)
9     @Id
10    @Column(name = "Id")
11    private float id;
12
13    @Basic
14    @Column(name = "Nombre")
15    private String nombre;
16
17    @Basic
18    @Column(name = "Direccion")
19    private String direccion;
20
21    @Basic
22    @Column(name = "Telefono")
23    private String telefono;
24
25    @Basic
26    @Column(name = "FechaNac")
27    private String fechaNac;
28
29    public float getId() { return id; }
```

5. Modificación del fichero **persistence.xml**

Añadimos la siguiente línea al fichero.

A screenshot of a code editor showing the contents of a file named 'persistence.xml'. The XML structure is as follows: Line 1: <?xml version="1.0" encoding="UTF-8" standalone="yes"?> Line 2: <persistence xmlns="https://jakarta.ee/xml/ns/persistence" Line 3: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Line 4: xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd" Line 5: version="3.0"> Line 6: <persistence-unit name="default"> Line 7: Line 8: <class>entity.ClienteEntity</class> Line 9: <properties> Line 10: <property name="hibernate.connection.url" Line 11: value="jdbc:mysql://uunerxcgwshpgek2:DRWYNZqyHKV1ohicbZU2@bwfkmrtycnrahk1ajtzw-mysql.services.clever-cloud.com:3306/bwfkmrtycnn Line 12: <property name="hibernate.connection.driver_class" value="com.mysql.cj.jdbc.Driver"/> Line 13: <property name="hibernate.connection.username" value="uunerxcgwshpgek2"/> Line 14: <property name="hibernate.connection.password" Line 15: value="DBMS: MySQL (no ver.) Case sensitivity: plain=mixed, delimited=exact Incorrect driver and/or database URL specified. "/> Line 16: <property name="hibernate.show_sql" value="true"/> Line 17: </properties> Line 18: </persistence-unit> Line 19: </persistence> The line 16 is highlighted with a red box. A lightbulb icon is visible to the left of line 16, indicating a warning or suggestion. The right side of the editor shows a search bar with '5' and some zoom controls.

Hibernate tiene una función incorporada para habilitar el registro de todas las declaraciones SQL generadas en la consola. *Esta función es buena para la resolución de problemas básicos y para ver qué está haciendo **Hibernate** detrás.*

6. Transacciones

En la siguiente imagen podemos ver cómo se crean las transacciones para poder operar con la BBDD.

```
no usages
public static void main(String[] args)
{
    //Creamos un objeto EntityManager para poder operar con la BBDD
    EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory( persistenceUnitName: "default");
    EntityManager entityManager = entityManagerFactory.createEntityManager();
    EntityTransaction transaction = entityManager.getTransaction();

    try
    {
        transaction.begin(); ← Inicio de la transacción

        //Aquí trabajamos con las entidades
        ClienteEntity c = new ClienteEntity();

        c.setId(1);
        c.setNombre("Juan");
        c.setDireccion("C/ Carlos III, 3");
        c.setTelefono("968121212");
        c.setFechaNac("12/01/2000");

        //Añadimos el objeto a la BBDD
        entityManager.persist(c); ← Equivale a un INSERT

        transaction.commit(); ← Fin de la transacción

    }
    finally {
        if (transaction.isActive())
        {
            transaction.rollback();
        }
        entityManager.close();
        entityManagerFactory.close();
    }
}
```

Actualización de los campos

Si se produce cualquier excepción, se deshace el trabajo hecho sin confirmar y se cierran los objetos que gestionan la transacción

7. HQL y NamedQueries

Hibernate utiliza un lenguaje de consulta potente (**HQL**) que se parece a SQL. Sin embargo, comparado con SQL, **HQL** está completamente orientado a objetos y comprende nociones como herencia, polimorfismo y asociación. Las consultas se escriben en HQL e Hibernate se encarga de convertirlas al SQL usado por la base de datos con la que estemos trabajando y ejecutarla para realizar la operación indicada.

HQL es que es case-insensitive, o sea que sus sentencias pueden escribirse en mayúsculas y minúsculas. Por lo tanto "SeLeCt", "seleCT", "select", y "SELECT" se entienden como la misma cosa.

Lo único con lo que debemos tener cuidado es con los nombres de las clases que estamos recuperando y con sus propiedades, ahí sí se distinguen mayúsculas y minúsculas. O sea, en este caso *"pruebas.Hibernate.Usuario"* NO ES LO MISMO que *"PrueBAs.Hibernate.UsuArio"*.

7.1 Cómo crear una NamedQuery

Paso 1. En la clase "ClienteEntity" escribimos la siguiente línea:

```
package entity;

import jakarta.persistence.*;

8 usages
@Entity
@NamedQuery(name = "AllClients", query= "SELECT c FROM ClienteEntity c")
@Table(name = "Cliente", schema = "bwfkmrtycnrahk1ajtzw", catalog = "")
public class ClienteEntity {
```

Paso 2. En la clase Main escribimos el siguiente código.

```
public class Main
{
    no usages
    public static void main(String[] args)
    {
        //Creamos un objeto EntityManager para poder operar con la BBDD
        EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory( persistenceUnitName: "default");
        EntityManager entityManager = entityManagerFactory.createEntityManager();
        EntityTransaction transaction = entityManager.getTransaction();

        try
        {
            transaction.begin();

            TypedQuery<ClienteEntity> allClients = entityManager.createNamedQuery( s: "AllClients", ClienteEntity.class);

            for(ClienteEntity c: allClients.getResultList())
            {
                System.out.println(c);
            }

            transaction.commit();
        }
        finally {
            if (transaction.isActive())
            {
                transaction.rollback();
            }
            entityManager.close();
            entityManagerFactory.close();
        }
    }
}
```

Creamos la query

Este método nos devuelve todos los objetos obtenidos en la query

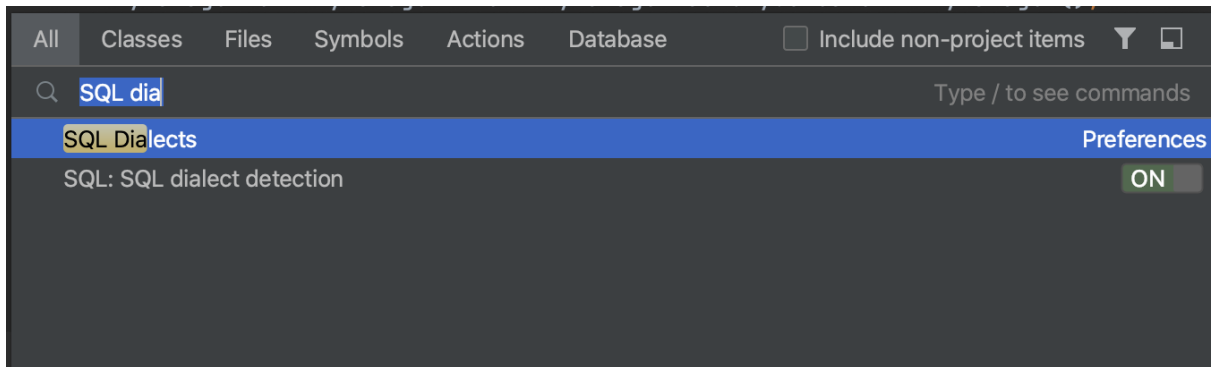
La salida del programa es la siguiente:

```
Hibernate: select c1_0.Id,c1_0.Direccion,c1_0.FechaNac,c1_0.Nombre,c1_0.Telefono from Cliente c1_0
ClienteEntity{id=1.0, nombre='Juan', direccion='C/ Carlos III, 3', telefono='968121212', fechaNac='12/01/2000'}
ClienteEntity{id=2.0, nombre='Lucas', direccion='C/ Carlos 3', telefono='111222333', fechaNac='14/11/22'}
ene. 02, 2023 9:33:35 P. M. org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PoolState stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://uunerxcqwhspqek2:DRWYNZqyHKV1ohicb2U2BwfkMrtycnrahk1ajtzw-mysql.services.clever-cloud.com:3306/bwfkMrtycnrahk1ajtzw]
```

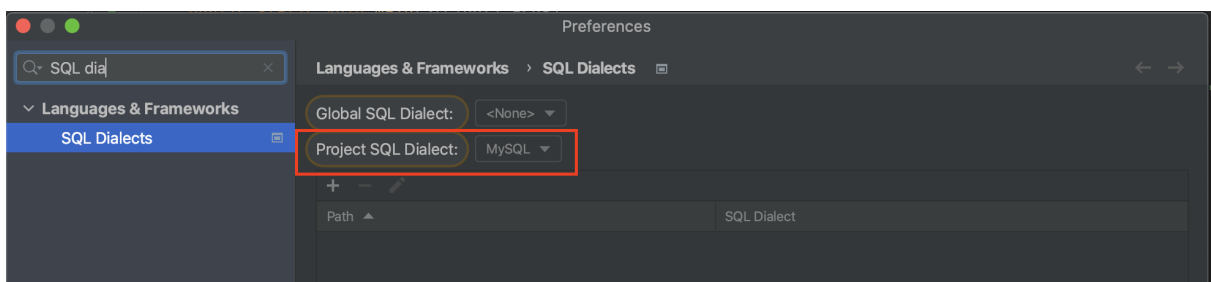
8. SQL NativeQueries

Paso 1. Establecemos MySQL como lenguaje de consulta para nuestros proyectos.

*Al pulsar la tecla **shift** dos veces seguidas aparecerá el siguiente cuadro de diálogo.*



Paso 2. Escribimos SQL dialects en el cuadro de búsqueda. En la nueva ventana, elegimos MySQL y pulsamos OK.



Paso 3. Ahora vamos a escribir el siguiente código en el programa:

```
public static void main(String[] args)
{
    //Creamos un objeto EntityManager para poder operar con la BBDD
    EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory( persistenceUnitName: "default");
    EntityManager entityManager = entityManagerFactory.createEntityManager();
    EntityTransaction transaction = entityManager.getTransaction();

    try
    {
        transaction.begin();

        Query nativeQuery = entityManager.createNativeQuery( s: "SELECT COUNT(*) FROM Cliente");

        System.out.println("Número de clientes: " + nativeQuery.getSingleResult());

        transaction.commit();
    }
    finally {
        if (transaction.isActive())
        {
            transaction.rollback();
        }
        entityManager.close();
        entityManagerFactory.close();
    }
}
```

SQL

Obtiene el resultado cuando es único

La salida obtenida por el programa es la siguiente:

```
Hibernate: SELECT COUNT(*) FROM Cliente
```

```
Número de clientes: 2 Salida
```

```
hibernate.show_sql
```