Clase 22

Java con persistencia en Base de Datos

Cómo crear una aplicación Java para gestión de base de datos (2da parte)

Clases entidad

Para que desde el lenguaje de programación Java se pueda tratar **cada registro o fila de las tablas como un objeto Java**, se debe crear una clase asociada a cada tabla de la base de datos. A ese tipo de clases se las conoce como **Clases Entidad (*Entity Classes*)**. También se puede acceder a una base de datos desde Java sin la utilización de objetos relacionados con las tablas, pero para este tipo de aplicación que se quiere crear resultará más cómodo el uso de objetos.

Para acceder a las bases de datos sin usar directamente clases asociadas a las tablas se usaría la librería **JDBC (Java Database Connectivity)**, includa en los paquetes *java.sql*, y para este caso que vamos a tratar (usando clases entidad) se usa la librería **JPA (Java Persistence API)** que se encuentra en los paquetes *javax.persistence*.

Las **clases entidad** que se acaban de comentar, se podrían crear manualmente, pero ***NetBeans*** ofrece una herramienta para **crearlas automáticamente** una vez que se haya establecido la conexión con la base de datos en dicho entorno de desarrollo, como se ha hecho en los pasos anteriores.

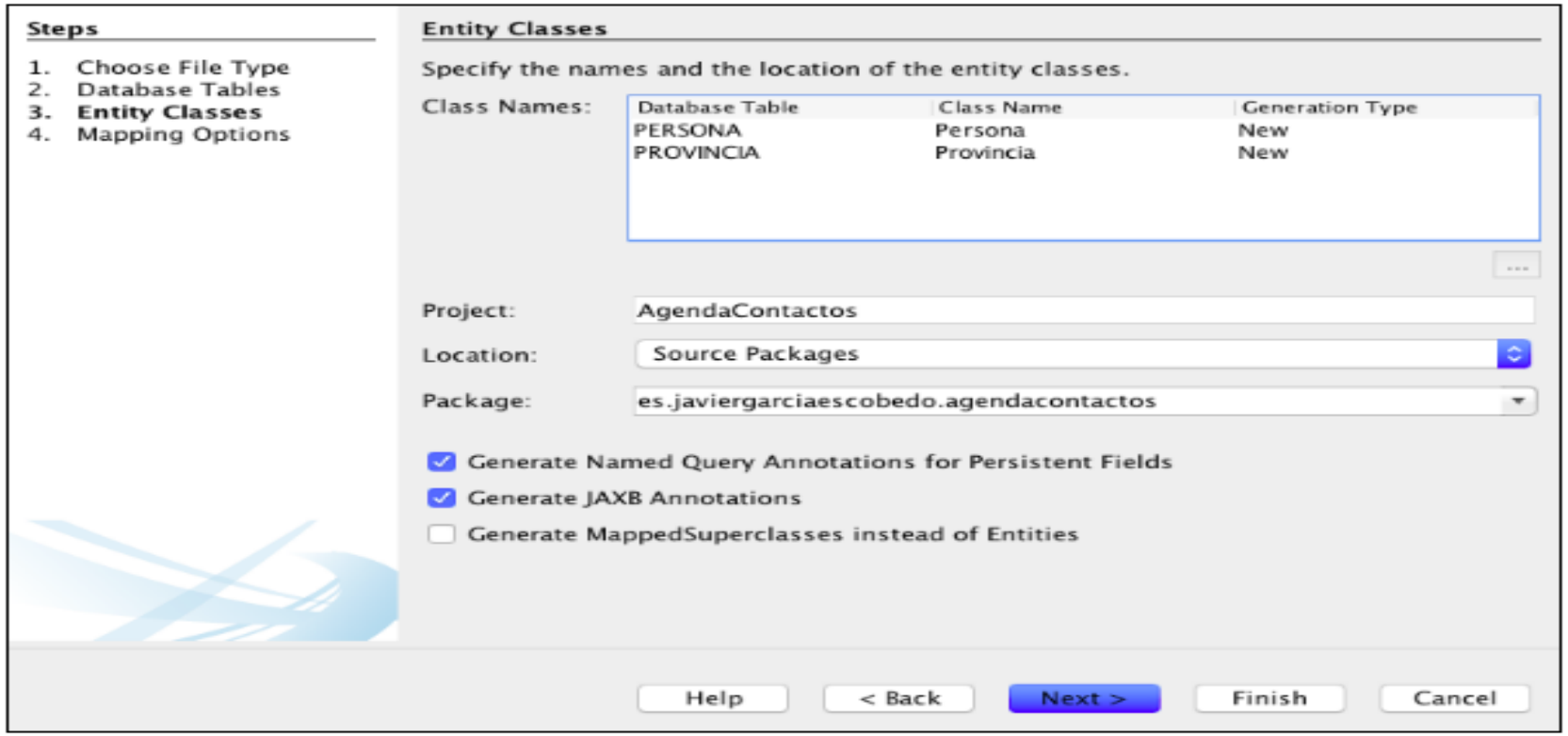
Para ello, usa el menú contextual en el proyecto, o sobre el paquete de fuentes donde se deseen crear las clases entidad, y usa la opción ***New > Entity Classes from Database***. Tras hacerlo, aparecerá una ventana de diáologo donde se deberá indicar la conexión con la base de datos que se quiere utilizar, y en la parte central aparecerán las tablas que contiene.

Clases entidad

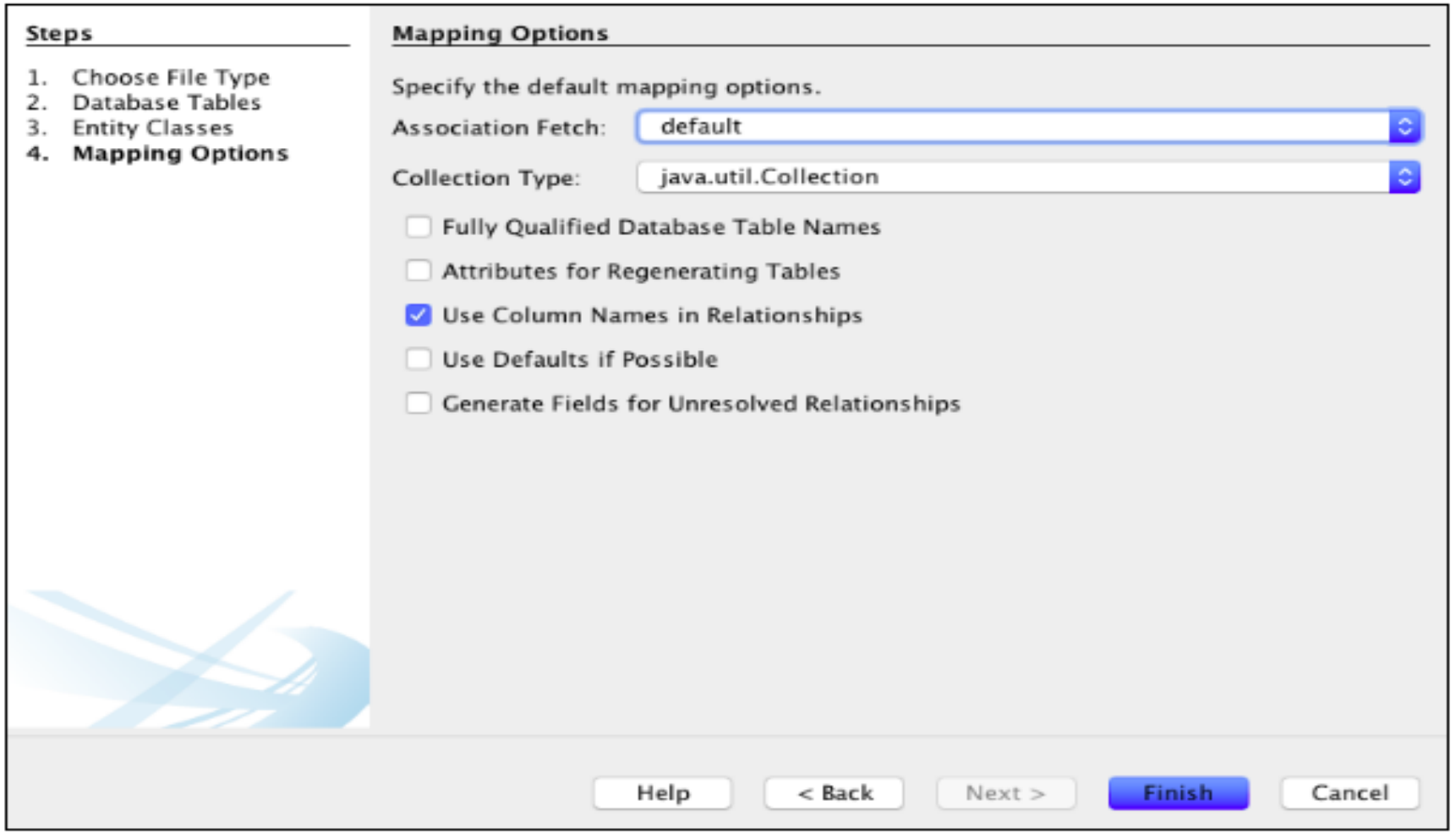
Clases entidad

Las **tablas** que se deseen utilizar para crear sus clases entidad correspondientes se deben **añadir a la lista de la derecha**. En este caso se añadirán todas, por lo que puedes usar el **botón *Add All***. Comprueba que aparecen las tablas en la parte derecha y usa el **botón *Next*** para continuar.

En la siguientes pantallas se pueden concretar algunos detalles sobre las clases entidad que se van a crear, pero puedes dejar las opciones por defecto que aparecen y continuar hasta finalizar este proceso.



Clases entidad



Clases entidad

Como resultado, se debe haber creado en los paquetes de fuentes **una clase Java por cada tabla** de la base de datos, así como el archivo ***persistence.xml*** dentro del paquete *META-INF*, que contiene información sobre la conexión con la base de datos como la ruta donde se encontrará, el usuario y contraseña de acceso, etc.

Si observas en **contenido** de ambas clases entidad que se han creado, podrás comprobar que son muy similares. Se muestra a continuación el contenido de la clase ***Provincia.java*** al ser más corta que *Persona.java*. A continuación se comentará brevemente los aspectos más importantes para que conozcas su contenido.

Clases entidad

```
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;

@Entity
@Table(name = "PROVINCIA")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Provincia.findAll", query = "SELECT p FROM Provincia p")
    , @NamedQuery(name = "Provincia.findById", query = "SELECT p FROM Provincia p WHERE p.id = :id")
    , @NamedQuery(name = "Provincia.findByCodigo", query = "SELECT p FROM Provincia p WHERE p.codigo = :codigo")
    , @NamedQuery(name = "Provincia.findByNombre", query = "SELECT p FROM Provincia p WHERE p.nombre = :nombre")})
public class Provincia implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "ID")
    private Integer id;
    @Column(name = "CODIGO")
```

Clases entidad


```
    private String codigo;
    @Basic(optional = false)
    @Column(name = "NOMBRE")
    private String nombre;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "provincia")
    private Collection<Persona> personaCollection;

    public Provincia() {
    }

    public Provincia(Integer id) {
        this.id = id;
    }

    public Provincia(Integer id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }
```

Clases entidad


```
public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

@XmlTransient
public Collection<Persona> getPersonaCollection() {
    return personaCollection;
}

public void setPersonaCollection(Collection<Persona> personaCollection) {
    this.personaCollection = personaCollection;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}
```

Clases entidad


```
    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Provincia)) {
            return false;
        }
        Provincia other = (Provincia) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "es.javiergarciaescobedo.agendacontactos.Provincia[ id=" + id + " ]";
    }

}
```

Propiedades de la clase

Los elementos más importantes de una clase entidad podrían ser las propiedades de la clase, que como puedes ver en su código fuente, están **relacionadas directamente con cada una de las columna** de la tabla.

Para la tabla *Provincia* se han generado automáticamente las siquientes propiedades:

```
    private Integer id;
    private String codigo;
    private String nombre;
```

Y para la tabla *Persona*, estas otras propiedades, que como puedes ver tienen **tipos de datos relacionados con los que se han asignado a la tabla** de la base de datos. Por ejemplo, a la columna *NUM_HIJOS*, que se indicó en el script SQL que sería de tipo *SMALLINT*, se le ha asignado el tipo *Short* de Java a la propiedad *numHijos*. Además, puedes ver que los **guiones bajos** utilizados en los nombres de las columnas de las tablas, se usan para distinguir la separación de palabras para los identificadores en Java, empezando en mayúsculas.

```
    private Integer id;
    private String nombre;
    private String apellidos;
    private String telefono;
    private String email;
    private Date fechaNacimiento;
    private Short numHijos;
    private Character estadoCivil;
    private BigDecimal salario;
    private Boolean jubilado;
    private String foto;
    private Provincia provincia;
```

Anotaciones JPA

Sobre cada propiedad de las clases entidad puedes encontrar una serie de líneas de código que **comienzan por @**. Esas líneas son **anotaciones de JPA** (Java Persistence API) que utilizará para conoce la relación entre dichas propiedades y las columnas de la tabla relacionada.

Por ejemplo, justo antes de la propiedad nombre se encuentra la anotación:

```
@Column(name = "NOMBRE")
private String nombre;
```

De esa manera, JPA sabrá que la propiedad *nombre* está relacionada con la columna *NOMBRE*.

Algunas propiedades también tienen añadida la anotación:

```
@Basic(optional = false)
```

De esta manera se sabrá que en la tabla de la base de datos, esa columna es de carácter **obligatorio** (se ha indicado NOT NULL en la creación de la tabla), y no se podrá dejar vacío ese dato.

Y a las **claves primarias autonuméricas** que se van a utilizar se les ha añadido:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

Claves foráneas

En las propiedades que se utilizan como claves foráneas también se añaden anotaciones. Por ello, en la propiedad provincia de la clase *Persona* se ha añadido automáticamente las anotaciones:

```
@JoinColumn(name = "PROVINCIA", referencedColumnName = "ID")
@ManyToOne(optional = false)
private Provincia provincia;
```

Es **importante** que observes que la **propiedad *provincia*** de la clase *Persona* no es de tipo numérico, sino que ahora se trata como un **objeto de la clase *Provincia***. Recuerda que en el script SQL se declaró la columna como INTEGER, ya que es una clave foránea relacionada con la columna ID (de tipo INTEGER) de la tabla PROVINCIA:

```
PROVINCIA INTEGER NOT NULL
```

En cambio en la clase Java no aparece como *Integer* sino:

```
private Provincia provincia;
```

Por tanto, la propiedad Java asociada debería ser de tipo *Integer*, pero al ser una clave foránea se convierte directamente al objeto correspondiente. De esta manera será **muy sencillo acceder a cualquier información contenida en el objeto Provincia** (por ejemplo el nombre de la Provincia) al que está asociado un determinado objeto Persona, y no sólo el ID correspondiente a la Provincia.

Por otro lado, en la clase *Provincia*, se ha añadido una propiedad extra (***personaCollection***):

```
@OneToMany(cascade = CascadeType.ALL, mappedBy = "provincia")
private Collection<Persona> personaCollection;
```

Se trata de una **propiedad de tipo *Collection***, que es uno de los tipos de lista que ofrece Java. En dicha lista podremos encontrar, en este caso, todas las personas (objetos *Persona*) que tienen una determinada Provincia. De esta manera, a partir de un determinado objeto *Provincia*, podremos conocer todos los objetos *Persona* relacionados.

Métodos get y set

Como suele ser habitual, las propiedades de la clase son de ambito privado (*private*), por lo que la información que contienen no puede ser consultada ni modificada directamente desde otra clase. Se han generado automáticamente los métodos *get* y *set* correspondientes a cada propiedad (*getNombre*, *setNombre*, etc), para poder obtener y modificar el valor de cada propiedad.

```
public String getNombre() {
    return nombre;
}
```

```
public void setNombre(String nombre) {
    this.nombre = nombre;
}
```

Métodos constructores

Vamos Buenos Aires

También se han generado varios métodos constructores para poder crear objetos de estas clases entidad. Por ejemplo, para la clase *Persona* se han creado los siguientes métodos constructores:

```
public Persona()
public Persona(Integer id)
public Persona(Integer id, String nombre, String apellidos) {
```

Es decir, se ha creado un método constructor **sin parámetros**, otro para indicar únicamente un ID (al ser la **clave principal**) y otro con los campos que se han marcado como **obligatorios** (NOT NULL) en la tabla.

Consultas predefinidas (NamedQuery)

En la parte superior del código fuente de las clases entidad puedes encontrar otro conjunto de anotaciones, dentro de la **sección *@NamedQueries***. Ahí se encuentran predefinidas una serie de **consultas a la tabla, a las que se le asigna un nombre**, y que puedes utilizar posteriormente en el código Java sin necesidad de escribir el código SQL, sino únicamente indicando el nombre de la consulta.

Por ejemplo, en la clase entidad *Provincia* hay consultas como:

```
@NamedQuery(name = "Provincia.findAll", query = "SELECT p FROM Provincia p")
```

Que permitirá **obtener todas las provincias** que se encuentren en la tabla *Provincia*, indicando el nombre de consulta ***Provincia.findAll***.

O esta otra:

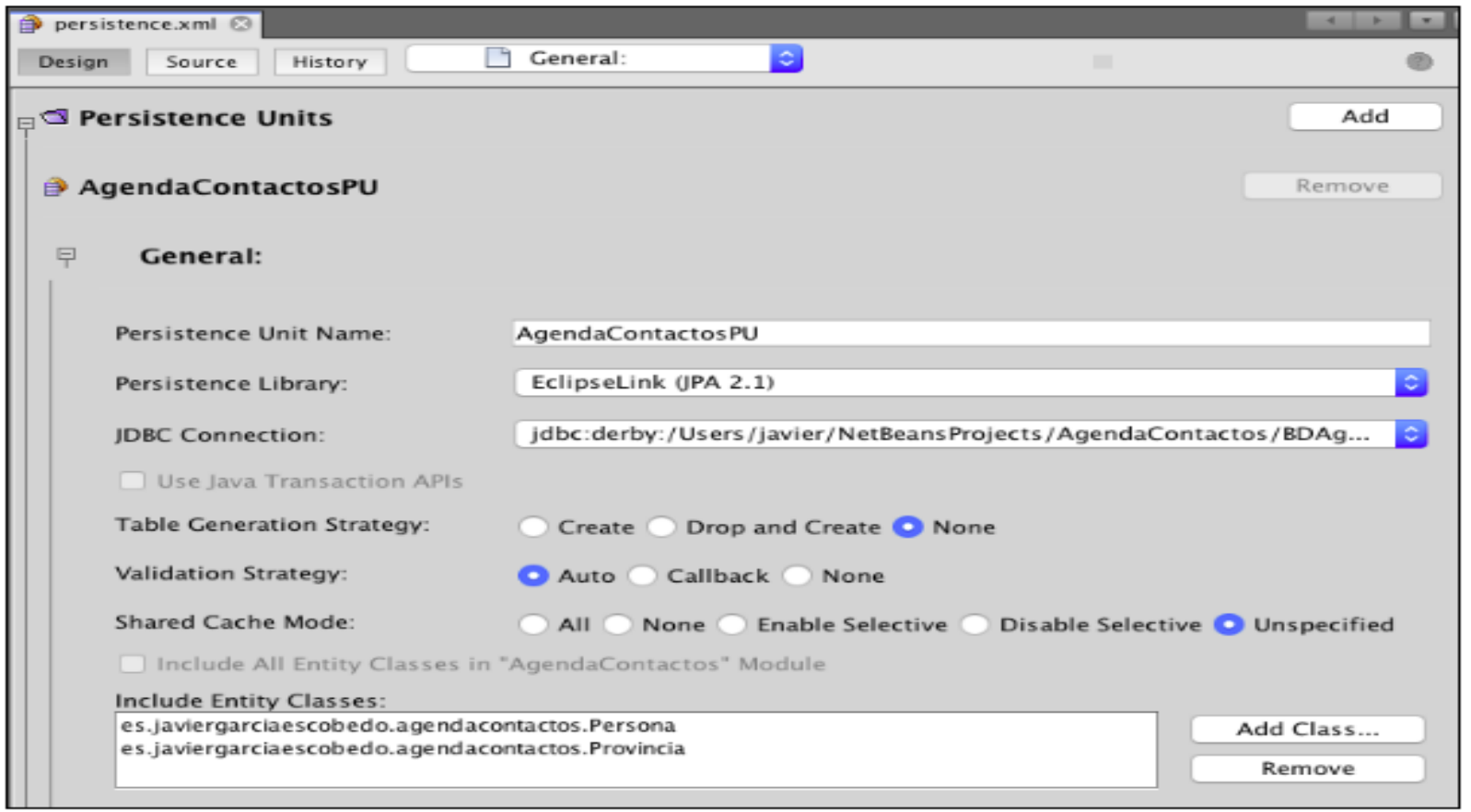
```
@NamedQuery(name = "Provincia.findByCodigo", query = "SELECT p FROM Provincia p WHERE p.codigo = :codigo")
```

Que permitirá obtener las provincias cuyo código corresponda con el que se indique en el parámetro *:codigo*.

Archivo persistence.xml de configuración de la Unidad de Persistencia

Anteriormente se comentó que además de las clases entidad, el asistente de generación de dichas clases crea además el **archivo *persistence.xml* dentro del paquete *META-INF***. En ese archivo se encuentra la configuración necesaria para realizar la conexión con la base de datos. Contiene, por ejemplo, el nombre de la base de datos, el usuario y contraseña de conexión, etc.

En la tecnología empleada por JPA se conoce como **Unidad de Persistencia (*PersistenceUnit*)** y el nombre que se le asigne debes tenerlo en cuenta, ya que se usará posteriormente en el código Java de la aplicación. En este caso se le ha asignado el nombre ***AgendaContactosPU***.



Archivo persistence.xml de configuración de la Unidad de Persistencia

Este archivo puede verse en modo visual, como se muestra en la imagen anterior, o en modo **código fuente XML**, usando la pestaña *Source*. En su contenido puedes ver, entre otras cosas, los nombres de las clases entidad que se utilizan, la ruta de conexión con la base de datos, así como el nombre de usuario y contraseña de acceso.

```
<?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-i
nstance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence
_2_1.xsd">
  <persistence-unit name="AgendaContactosPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>es.javiergarciaescobedo.agendacontactos.Provincia</class>
    <class>es.javiergarciaescobedo.agendacontactos.Persona</class>
    <shared-cache-mode>ENABLE_SELECTIVE</shared-cache-mode>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:derby:/<<<RUTA_AL_PROYECTO>>>/AgendaContactos/BDAgendaCont
actos;create=true"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>
      <property name="javax.persistence.jdbc.password" value="root"/>
    </properties>
  </persistence-unit>
</persistence>
```