	UNIVERSIDAD DON BOSCO **FACULTA DE INGENIERIA** **ESCUELA DE COMPUTACION**
Ciclo II	**Desarrollo de aplicaciones con Web Frameworks** **Guía de Laboratorio No. 7** **Java Persistente API**

I. OBJETIVOS.

- Que el alumno comprenda los beneficios de las aplicaciones construidas con Java Persistence API (JPA).

II. INTRODUCCION TEORICA

La mayoría de la información de las aplicaciones empresariales es almacenada en bases de datos relacionales. La persistencia de datos en Java, y en general en los sistemas de información, ha sido uno de los grandes temas a resolver en el mundo de la programación.

Al utilizar únicamente JDBC se tiene el problema de crear demasiado código para poder ejecutar una simple consulta. Por lo tanto, para simplificar el proceso de interacción con una base de datos (select, insert, update, delete), se ha utilizado desde hace ya varios años el concepto de frameworks ORM (Object Relational Mapping), tales como JPA e Hibernate.

Java Persistence API, mejor conocido como JPA, es el estándar de persistencia en Java. JPA implementa conceptos de frameworks ORM (Object Relational Mapping).

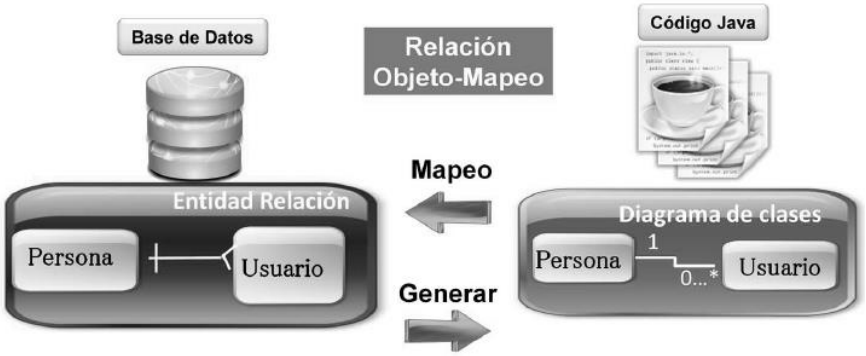


Figura 1. Object Relational Mapping en Java

Como se puede observar en la figura 1, un framework ORM nos permite "mapear" una clase Java con una tabla de Base de Datos. Por ejemplo, la clase Persona, al crear un objeto en memoria, podemos almacenarlo directamente en la tabla de Persona, simplemente ejecutando una línea de código: em.persist(persona). Esto ha simplificado enormemente la cantidad de código a escribir en la capa de datos de una aplicación empresarial.

Componentes de la persistencia en Java

JEE define cuatro módulos principales para el control de la persistencia en Java:

1. El API de persistencia de Java, que está basado en objetos persistentes denominados **entidades JPA**. Estos objetos están relacionados con los elementos de una base de datos de la siguiente manera:

   a. Una tabla de BD se asocia a la clase Entity de JPA.
   b. Un registro de una tabla de BD se identifica como una instancia de una entidad JPA.
   c. Una columna de una tabla de BD es un atributo de una instancia de entidad

   Además, se definen anotaciones que sirven para establecer mapeos entre tablas y entidades JPA.

2. Un lenguaje propio para realizar consultas a bases de datos denominado **JPQL** (Java Persistence Query Language).

3. **El API de criterios de persistencia.** Vinculados al interfaz EntityManager, que incluye las operaciones de persistencia que pueden realizarse sobre las entidades JPA:
   a. Define su ciclo de vida.
   b. Gestiona la comunicación con la base de datos.

4. Mapeo de metadatos de un modelo relacional de base de datos a un modelo de objetos (ORM-Object/ Relational Mapping) mediante anotaciones o ficheros XML.

La idea de JPA es trabajar con objetos Java y no con código SQL, de tal manera que podamos enfocarnos en el código Java. JPA permite abstraer la comunicación con las bases de datos y crea un estándar para ejecutar consultas y manipular la información de una base de datos.

ENTIDADES Y PERSISTENCIAS CON JPA

Los objetos Entity de JPA

Una clase de entidad es un POJO y puede configurarse por medio de anotaciones. Las clases que definen objetos tipo Entity, son marcadas por la anotación @Entity y su anatomía se caracteriza por las siguientes características:

- El POJO marcado por la anotación **@Entity** tiene estructura de JavaBean: un constructor vacío declarado como public o protected, propiedades declaradas como private/protected y métodos getX() y setX() para acceder a ellas.

- Todos los atributos de la entidad serán persistentes y se corresponderán con las columnas de la tabla que mapea. Solo los marcados por la anotación **@Transient** no lo serán.

- La clase no podrá ser declarada como final. Los atributos persistentes no podrán ser declarados como static o final.

A continuación, se presenta un ejemplo de una clase de entidad con anotaciones:

```java
@Entity
public class Persona implements Serializable {

    @Id
    @GeneratedValue
    private Long personaId;

    @Column(nullable = false)
    private String nombre;

    private String apePaterno;
    private String apeMaterno;
    private String email;
    private Integer telefono;

    //Constructores, getters, setters
}
```

Anotaciones para mapeo de atributos

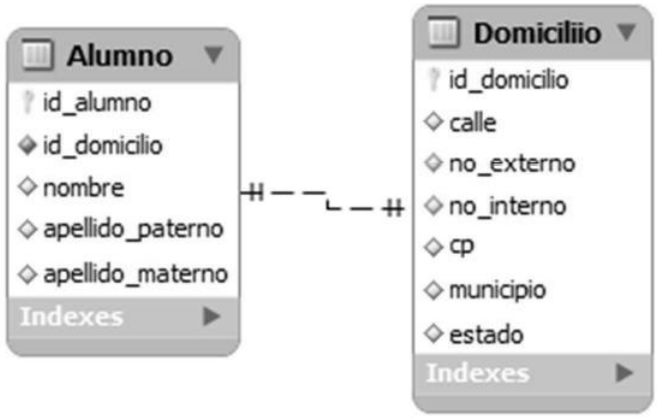
Se muestra a continuación una tabla que recoge las principales anotaciones que pueden utilizarse para configurar el mapeo de atributos entre tablas de bases de datos y propiedades del objeto Entity.

ANOTACIÓN	DESCRIPCIÓN
`@Column`	Especifica una columna de la tabla a mapear sobre un atributo de la clase `Entity`.
`@Transient`	Identifica a un atributo no persistente que no se mapeará en la base de datos.
`@Id`	Marca el atributo como identificador (llave primaria). Según JPA, es obligatorio que exista un atributo de este tipo en cada clase.
`@JoinColumn`	Indica un atributo que actúa como llave foránea en otra entidad. Algunos de sus atributos son los siguientes: `name:` Identifica a la columna que guarda la clave foránea. Es el único campo obligatorio. `referenced:` nombre de la tabla relacionada que contiene la columna.

Relaciones entre entidades

Al igual que sucede entre las tablas de una base de datos relacional, es posible establecer relaciones entre diferentes objetos tipo Entity. Estas relaciones implican cardinalidad, es decir el número de referencias que tiene un objeto en relación con otro. Serán especificadas a través de las anotaciones:

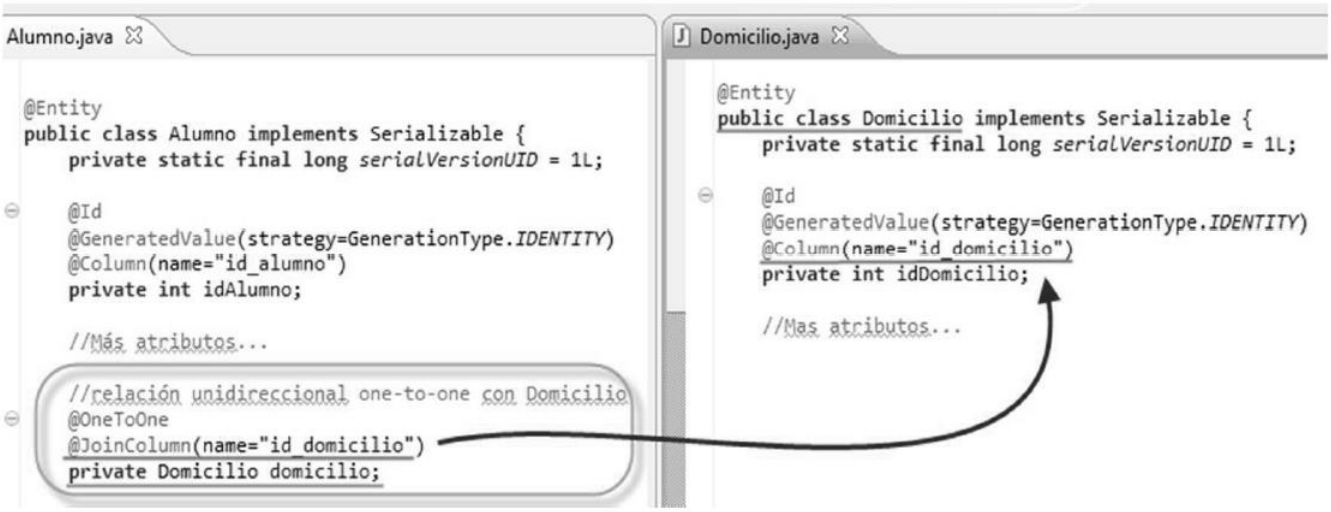
- **@OneToOne:** cada lado de la relación tiene como máximo una referencia al otro objeto.



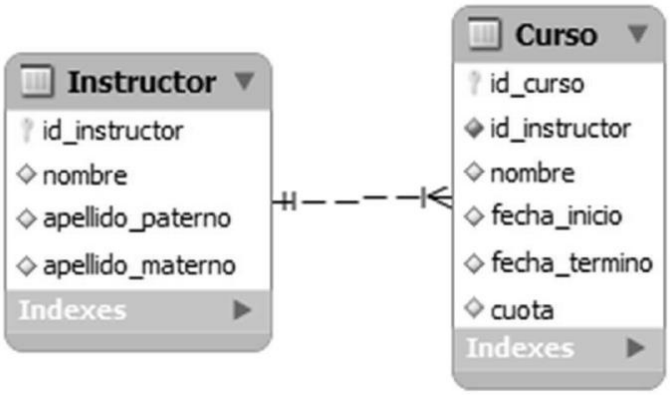
En la figura anterior podemos observar una relación de 1 a 1, en la cual una entidad Alumno tiene una relación con sólo un domicilio, y viceversa, esto es, la cardinalidad entre las entidades es de 1 a 1.

Podemos observar que la clase de Alumno es la que guarda la referencia de un objeto Domicilio, para mantener una navegabilidad unidireccional y que a partir de un objeto Alumno podamos recuperar el objeto Domicilio asociado.

Es importante destacar que el manejo de relaciones es por medio de objetos, y no atributos aislados, esto nos permitirá ejecutar queries con JPQL que recuperen objetos completos.

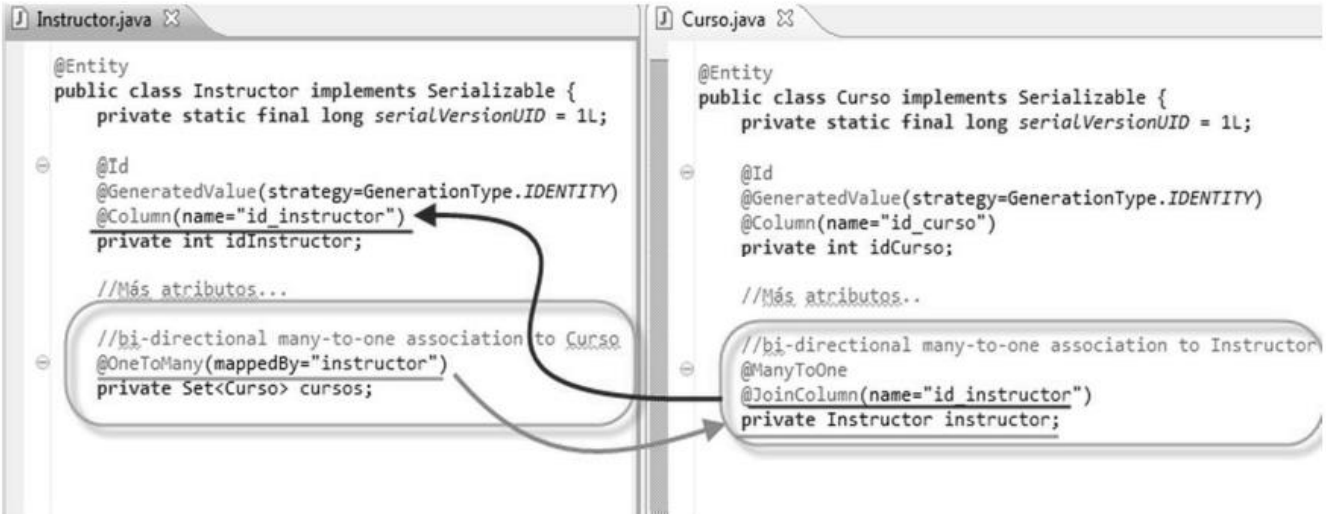


- **@OneToMany y @ManyToOne:** implican que una instancia de un objeto Entity puede estar relacionado con varias instancias de otro (y viceversa).



Cuando un objeto de Entidad está asociado con una colección de otros objetos de Entidad, es más común representar esta relación como una relación de Uno a Muchos. Por ejemplo, en la figura anterior podemos observar la relación de un Instructor el cual puede tener asignados varios Cursos (relación de 1 a muchos).

Si queremos saber desde la clase **Curso** qué instructor tiene asociado, deberemos agregar el mapeo bidireccional (ManyToOne) hacia Instructor. Y en la clase Instructor, se especifica una relación uno a muchos (OneToMany) hacia una colección de objetos de tipo Curso, el cual puede ser una estructura de datos Set o un List.



Métodos para gestionar el ciclo de vida de las entidades

- **public void persist (Object entity):** este método graba una entidad en la base de datos y la marca como gestionada. Bajo este método se configuran las sentencias INSERT que van a modificar los datos en la BD, pero esto no significa que sean ejecutados automáticamente. Lo normal es que no se ejecuten hasta que terminen de configurarse todas las sentencias que forman parte de una transacción.

- **public <T> T merge (T entity):** mezcla una entidad con el contexto de persistencia del EntityManager, devuelve el resultado de la mezcla y actualiza el estado de la entidad en la base de datos.

- **public void remove(Object entity):** elimina una entidad de la base de datos.

- **public <T> T find(Class<T> entityClass, Object primarykey):** localiza una instancia de una entidad a través de su clave primaria.

- **public void flush():** sincroniza el estado de las entidades en el contexto de persistencia del interfaz EntityManager con la base de datos.

- **public void refresh(Object entity):** refresca la entidad desde la base de datos.

- **public Query createQuery(String jpqlString):** crea una consulta query dinámica utilizando JPQL.

- **public Query createNativeQuery(String jqlString):** crea una query dinámica utilizando SQL nativo.

- **public Query createNamedQuery(String name):** crea una instancia predefinida para establecer consultas.

JAVA PERSISTENCE QUERY LANGUAGE (JPQL)

JPQL es un lenguaje de consulta similar a SQL. **La principal diferencia es que SQL funciona con esquemas, tablas y procedimientos almacenados del DBMS pero JPQL los manipula con objetos de Java y sus respectivos atributos.**

El origen de este lenguaje está en la necesidad de estandarizar un lenguaje de consultas aplicable al esquema de abstracción de datos y persistencia definido a través de objetos tipo Entity. Al ser un lenguaje de expresiones, éstas pueden compilarse para convertirse en lenguaje de consultas SQL o a cualquier otro.

Tipos de Queries:

- **Dynamic queries:** Consultas que reciben parámetros en tiempo de ejecución.
- **Named queries:** Consultas ya creadas previamente, y que se pueden ejecutar solo utilizando el nombre.
- **Native queries:** Consultas con SQL nativa, ya que hay casos de uso que lo requieren.

Sintaxis de JPQL

Al igual que SQL, JPQL basa su capacidad de acción en cuatro tipos de sentencias: SELECT, INSERT, UPDATE, DELETE. Las operaciones de inserción se pueden resolver de forma casi completa con el método persist() del Entity Manager. A continuación, se va a exponer la sintaxis del lenguaje en el uso de cada una de las sentencias restantes.

SENTENCIA SELECT

SELECT puede contener a su vez algunos tipos de modificadores o clausulas:

- **SELECT**, para especificar el tipo de objetos o valores que deben ser seleccionados.
- **FROM**, para especificar el dominio de datos al que se refiere la consulta.
- **WHERE**, modificador opcional, que puede ser utilizado para restringir los resultados devueltos por la consulta.
- **GROUP BY**, modificador opcional, que permite agrupar los resultados.
- **HAVING**, modificador opcional, que permite incluir condiciones de filtrado.
- **ORDER BY**, modificador opcional, que permite ordenar los resultados devueltos por la consulta.

A continuación, se incluyen algunos ejemplos sencillos de consultas:

- `SELECT m FROM MatriculasEntity m`

  Esta consulta devuelve todas las matrículas, identificadas de forma general por la variable m.

- `SELECT DISTINCT a FROM AlumnosEntity a`

  El modificador DISTINCT sirve para eliminar de la respuesta valores duplicados, en el hipotético caso de que existan.

- `SELECT DISTINCT a FROM AlumnosEntity WHERE a.nombre = :nombre AND a.apellido1 :=apellido1`

  En este caso se devolverán todos los elementos no duplicados de tipo entidad AlumnosEntity cuyo nombre y primer apellido coincida con el especificado como parámetro.

- `SELECT a FROM AlumnosEntity a WHERE a.nombre LIKE 'Mari%'`

  En este caso se devolverán todos los elementos no duplicados de tipo entidad AlumnosEntity cuyo nombre comience por "Mari". De esta forma la partícula LIKE sirve para introducir patrones de texto.

- `SELECT COUNT(p) FROM Pelicula p`

  COUNT() es una función agregada de JPQL, cuya misión es devolver el número de ocurrencias tras realizar una consulta. Por tanto, en el ejemplo anterior, el valor devuelto por la función agregada es el resultado de la sentencia al completo. Otras funciones agregadas son AVG para obtener la media aritmética, MAX para obtener el valor máximo, MIN para obtener el valor mínimo, y SUM para obtener la suma de todos los valores.

- `SELECT p FROM Pelicula p WHERE p.duracion < 120 AND NOT (p.genero = 'Terror')`

  Gracias a la sentencia anterior se obtendrían todas las instancias de Pelicula con una duración menor a 120 minutos que no (NOT) son del género Terror.

- `SELECT p FROM Pelicula p WHERE p.duracion BETWEEN 90 AND 150`

  La sentencia anterior obtiene todas las instancias de Pelicula con una duración entre (BETWEEN) 90 y (AND) 150 minutos. BETWEEN puede ser convertido en NOT BETWEEN, en cuyo caso se obtendrían todas las películas que una duración que no (NOT) se encuentren dentro del margen (BETWEEN) 90-150 minutos.

  `SELECT p FROM Pelicula p WHERE p.titulo LIKE 'El%'`

  La sentencia anterior obtiene todas las instancias de Pelicula cuyo título sea como (LIKE) *El%* (el símbolo de porcentaje es un comodín que indica que en su lugar puede haber entre cero y más caracteres). Resultados devueltos por esta consulta incluirían películas con un título como *El Renacido*, *El Pianista*, o si existe, *El*. El otro comodín aceptado por LIKE es el carácter "guion bajo" (_), el cual representa un único carácter indefinido (ni cero caracteres ni más de uno; uno y solo uno).

Existen otras cláusulas que pueden ser utilizadas en las consultas como son:

- **NULL:** condición de nulidad en consultas, por ejemplo, para devolver relaciones que no existen entre dos entidades dadas.
- **EMPTY:** es una expresión condicional para indicar que un elemento está vacío.
- **BETWEEN-AND:** sirve para imponer condiciones en las consultas en un rango de valores.
- **Operadores de comparación <, >, =:** sirven para establecer comparaciones con el modificador WHERE.

Además, es posible utilizar expresiones funcionales como las siguientes:

- **CONCAT(String, String):** devuelve un String que es la concatenación de los dos pasados como parámetro.
- **LENGTH(String):** devuelve el entero que indica la longitud del String pasado como parámetro.

- **LOCATE(String, String [, start]):** devuelve un entero con la posición de un determinado String dentro de otro String. Si no se localiza, se devuelve un 0.
- **SUBSTRING(String, start, length):** devuelve un String que es un subconjunto del pasado como parámetro comenzando en la posición marcada por start y de longitud length.
- **TRIM([[LEADING|TRAILING|BOTH] char) FROM] (String):** esta función elimina un determinado carácter desde el comienzo o final de un String. Si no ese especifica ningún carácter, se eliminan espacios en blanco.
- **LOWER(String):** convierte un String al equivalente en minúsculas.
- **UPPER(String):** convierte un String al equivalente en mayúsculas.

Y otras como:

- **Expresiones aritméticas:** `ABS(number)`, `MOD(int, int)`, `SQRT(double)` y `SIZE(Collection)`.
- **Expresiones horarias:** `CURRENT_DATE`, `CURRENT_TIME` y `CURRENT_TIMESTAMP`.

SENTENCIA UPDATE

Permite actualizar valores almacenados por la entidad. Puede incluir de forma opcional el modificador WHERE. Se ilustra con un ejemplo:

```
UPDATE CursosEntity c SET c.nota = 8.5 WHERE c.nombre = 'JavaEE' OR c.nombre = 'JavaSE'
```

SENTENCIA DELETE

Permite borrar un registro completo correspondiente a una entidad. Al igual que UPDATE, puede incluir de forma opcional el modificador WHERE. Algunos ejemplos a continuación:

- `DELETE FROM CursosEntity c WHERE c.nota = 9`
- `DELETE FROM CursosEntity c WHERE c.nombre IS EMPTY`

Uso del JPQL

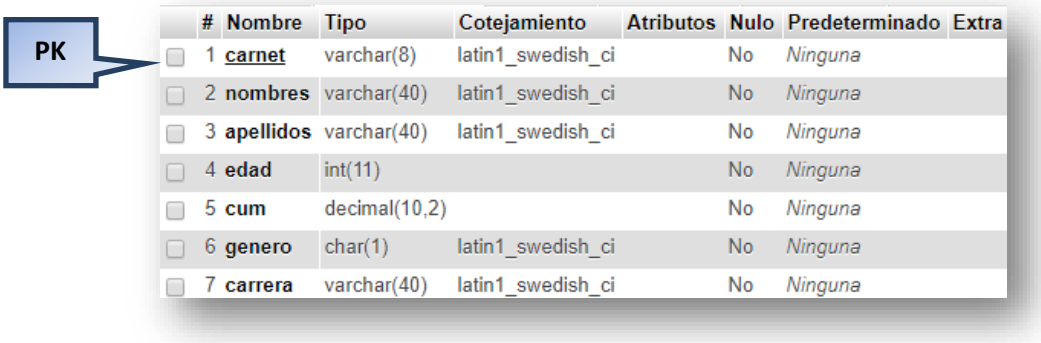
Las consultas expresadas en JPQL serán ejecutadas a través del interfaz EntityManager y los métodos createX como los siguientes:

- `Query createQuery(String qlString)`: este método se utiliza para crear consultas dinámicas que toman valor en ejecución.
- `Query createNamedQuery(String name)`: este método se utiliza para crear consultas estáticas que están codificadas previamente o predefinidas como metadatos a través del objeto `javax.persistence.NamedQuery` que las marca con la anotación `@NamedQuery`.

El interfaz proporciona además otro método: `Query createNativeQuery(String sqlString)` que permite ejecutar directamente sentencias SQL nativas.

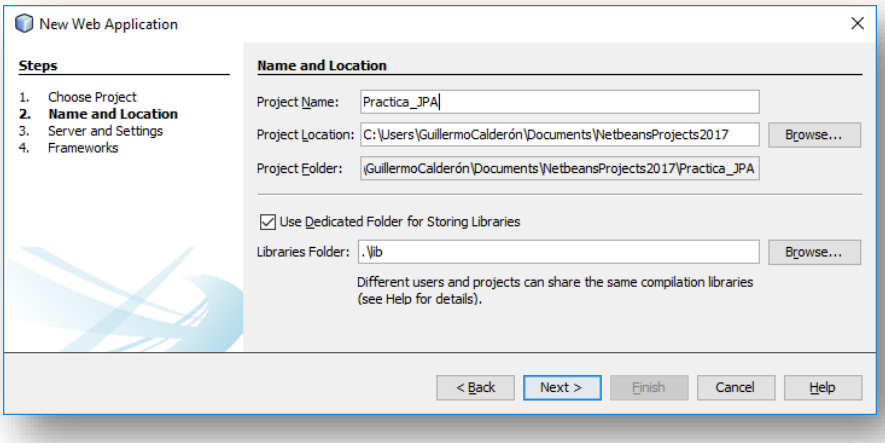
III. PROCEDIMIENTO

1. Crear una base de datos MySQL con nombre "**registro_estudiantes**". Dentro de esta bd deberá crear una tabla llamada "**estudiantes**" con la siguiente estructura:

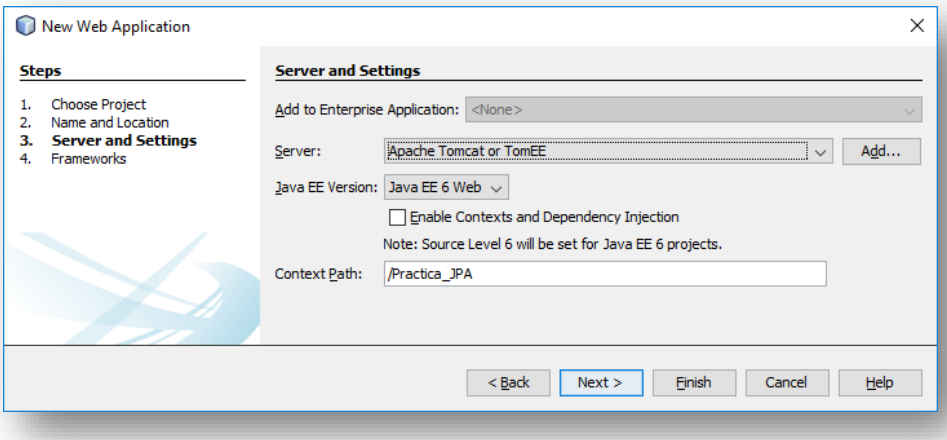


#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
1	carnet	varchar(8)	latin1_swedish_ci		No	*Ninguna*	
2	nombres	varchar(40)	latin1_swedish_ci		No	*Ninguna*	
3	apellidos	varchar(40)	latin1_swedish_ci		No	*Ninguna*	
4	edad	int(11)			No	*Ninguna*	
5	cum	decimal(10,2)			No	*Ninguna*	
6	genero	char(1)	latin1_swedish_ci		No	*Ninguna*	
7	carrera	varchar(40)	latin1_swedish_ci		No	*Ninguna*	

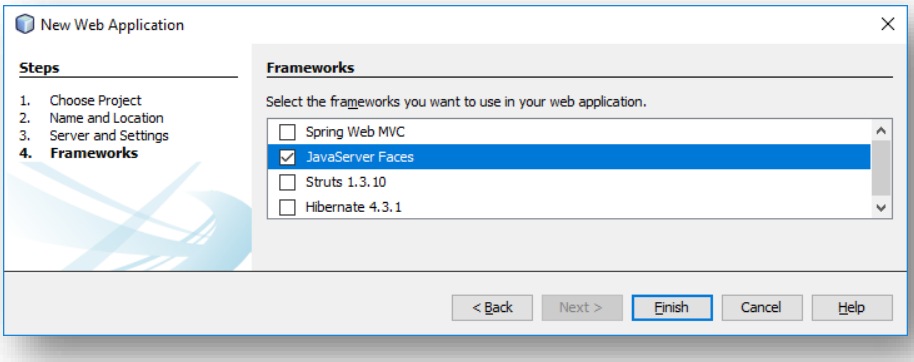
2. Abrir netbeans para crear un nuevo proyecto web (Web Application) con nombre "Practica_JPA".



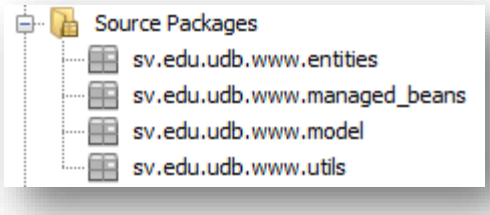
Asegúrese de seleccionar Apache Tomcat como servidor web y la versión 6 del JEE.



En la siguiente pantalla Netbeans nos preguntará si vamos a utilizar alguno de los frameworks que tiene disponibles. Deberá seleccionar el framework "**Java Server Faces**".

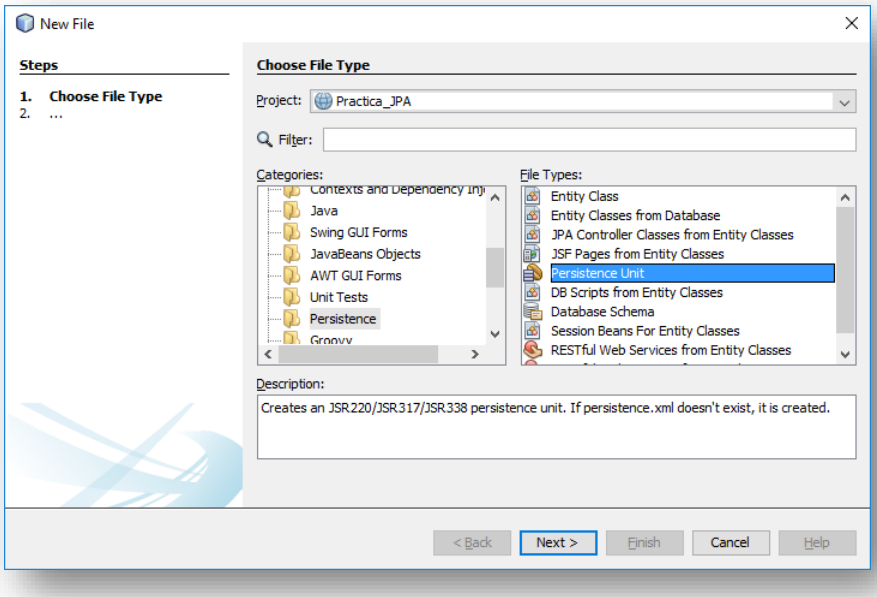


3. Crear los siguientes paquetes dentro de su proyecto:

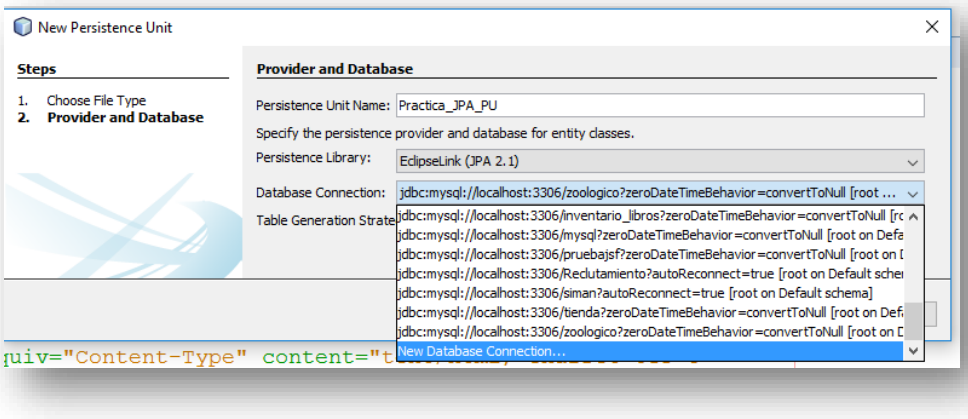


- sv.edu.udb.www.entities
- sv.edu.udb.www.managed_beans
- sv.edu.udb.www.model
- sv.edu.udb.www.utils

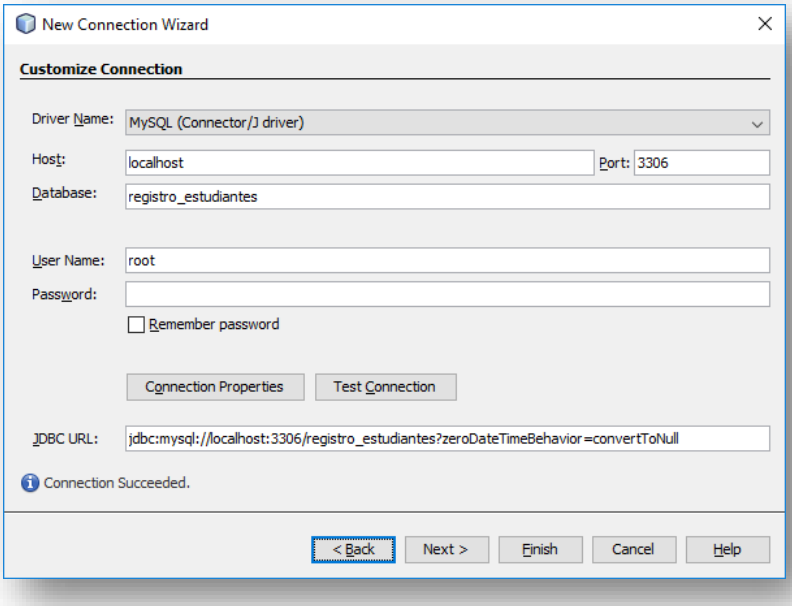
4. A continuación, crearemos la unidad de persistencia del proyecto. En esta unidad de persistencia definiremos los parámetros de configuración de la conexión con la base de datos (nombre de la bd, usuario, contraseña, etc.). Para realizar esta tarea debemos hacer click sobre el proyecto y seleccionar la opción "New>Other>Persistence> Persistence unit"



El nombre de la unidad de persistencia deberá ser "**Practica_JPA_PU"** y debe seleccionarse "EclipseLink" (la implementación oficial de JPA) como librería de persistencia. Además, deberá agregarse una nueva conexión hacia la bd "**registro_estudiantes**".



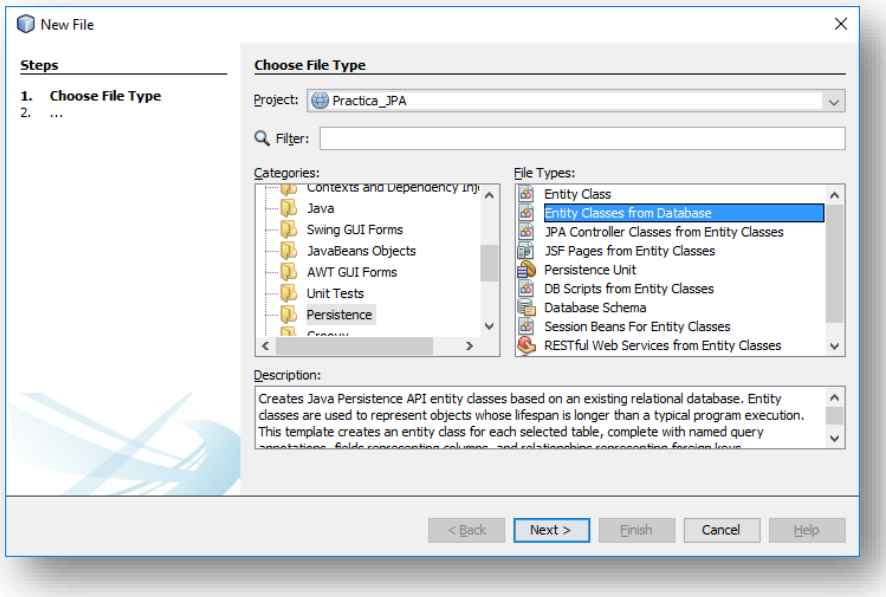
En la siguiente pantalla debe seleccionarse el driver de MySQL (Connector/J Driver). Finalmente deben definirse los parámetros de configuración de la conexión con la BD y probar dicha conexión.



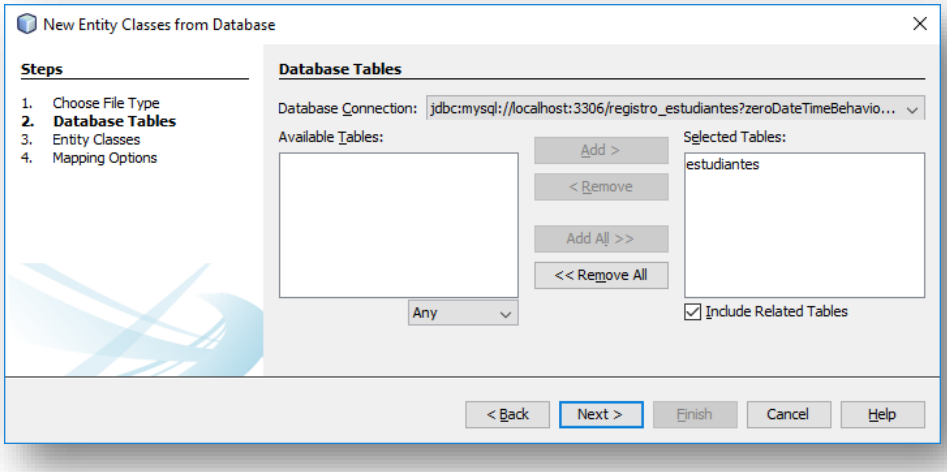
Si todo ha salido bien al configurar la unidad de persistencia, en el nodo de archivos de configuración del proyecto (configuration Files) debería haber un archivo denominado persistence.xml con el siguiente contenido:

```
<?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-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="Practica_JPA_PU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/registro_estudiantes?zeroDateTimeBehavior=co
nvertToNull"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="javax.persistence.schema-generation.database.action"
value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

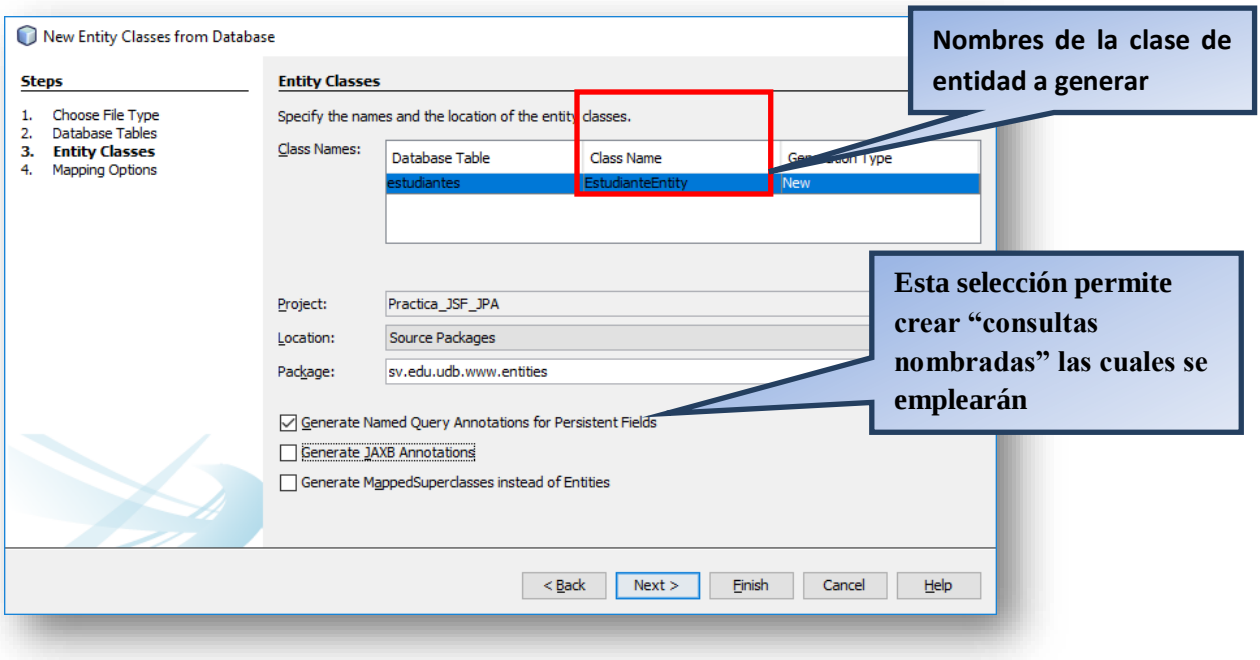
5. Ahora procederemos a generar la clase de entidad a partir de la tabla de la base de datos. Para ello, ubíquese sobre el paquete **sv.edu.udb.www.entities,** haga click derecho y seleccione la opción “new>Other>Persistence>Entity Class from Database”.



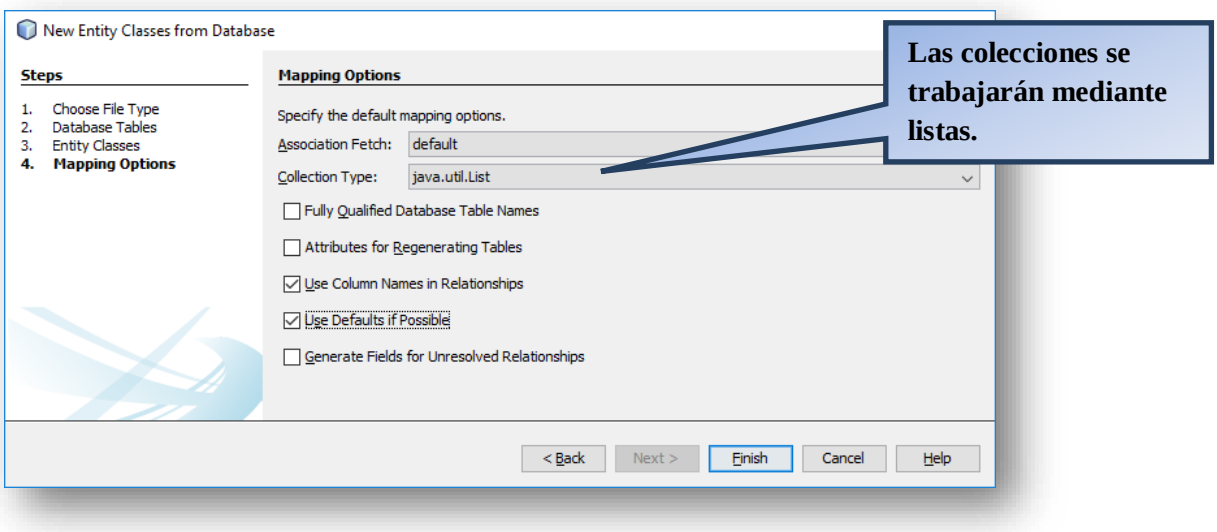
A continuación, seleccione la tabla estudiantes.



Luego proceda a cambiar el nombre de la clase de entidad de tal forma que se llame **EstudianteEntity**.



En la pantalla posterior debe seleccionar las siguientes opciones:



Al finalizar el asistente, Netbeans habrá generado la clase EstudianteEntity a partir de la tabla "estudiantes" de la base de datos. El código de esta clase de entidad se muestra a continuación:

EstudianteEntity.java

```
 package sv.edu.udb.www.entities;

import java.io.Serializable;
import java.math.BigDecimal;
import javax.persistence.Basic;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@Table(name = "estudiantes")
@NamedQueries({
    @NamedQuery(name = "EstudianteEntity.findAll", query = "SELECT e FROM
EstudianteEntity e"),
    @NamedQuery(name = "EstudianteEntity.findByCarnet", query = "SELECT e FROM
EstudianteEntity e WHERE e.carnet = :carnet"),
    @NamedQuery(name = "EstudianteEntity.findByNombres", query = "SELECT e
FROM EstudianteEntity e WHERE e.nombres = :nombres"),
    @NamedQuery(name = "EstudianteEntity.findByApellidos", query = "SELECT e
FROM EstudianteEntity e WHERE e.apellidos = :apellidos"),
    @NamedQuery(name = "EstudianteEntity.findByEdad", query = "SELECT e FROM
EstudianteEntity e WHERE e.edad = :edad"),
    @NamedQuery(name = "EstudianteEntity.findByCum", query = "SELECT e FROM
EstudianteEntity e WHERE e.cum = :cum"),
    @NamedQuery(name = "EstudianteEntity.findByGenero", query = "SELECT e FROM
EstudianteEntity e WHERE e.genero = :genero"),
    @NamedQuery(name = "EstudianteEntity.findByCarrera", query = "SELECT e
FROM EstudianteEntity e WHERE e.carrera = :carrera")})
public class EstudianteEntity implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
```

Consultas nombradas creadas por JPA las cuales podemos utilizar posteriormente.

```
    private String carnet;
    @Basic(optional = false)
    private String nombres;
    @Basic(optional = false)
    private String apellidos;
    @Basic(optional = false)
    private int edad;
    @Basic(optional = false)
    private BigDecimal cum;
    @Basic(optional = false)
    private Character genero;
    @Basic(optional = false)
    private String carrera;

    public EstudianteEntity() {
    }

    public EstudianteEntity(String carnet) {
        this.carnet = carnet;
    }

    public EstudianteEntity(String carnet, String nombres, String apellidos,
int edad, BigDecimal cum, Character genero, String carrera) {
        this.carnet = carnet;
        this.nombres = nombres;
        this.apellidos = apellidos;
        this.edad = edad;
        this.cum = cum;
        this.genero = genero;
        this.carrera = carrera;
    }

    public String getCarnet() {
        return carnet;
    }

    public void setCarnet(String carnet) {
        this.carnet = carnet;
    }

    public String getNombres() {
        return nombres;
    }

    public void setNombres(String nombres) {
        this.nombres = nombres;
    }

    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;
    }

    public BigDecimal getCum() {
        return cum;
    }

    public void setCum(BigDecimal cum) {
        this.cum = cum;
    }

    public Character getGenero() {
        return genero;
    }

    public void setGenero(Character genero) {
        this.genero = genero;
    }

    public String getCarrera() {
        return carrera;
    }

    public void setCarrera(String carrera) {
        this.carrera = carrera;
    }

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

    @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 EstudianteEntity)) {
            return false;
        }
        EstudianteEntity other = (EstudianteEntity) object;
        if ((this.carnet == null && other.carnet != null) || (this.carnet !=
null && !this.carnet.equals(other.carnet))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "sv.edu.udb.www.entities.EstudianteEntity[ carnet=" + carnet +
" ]";
    }
}
```

6. Proceda a crear una clase llamada "**JpaUtil**" dentro del paquete **sv.edu.udb.www.utils**. Ubique el siguiente código fuente dentro de la clase.

```
package sv.edu.udb.www.utils;

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

public class JpaUtil {

    private static final EntityManagerFactory emFactory;

    static {
        emFactory = Persistence.createEntityManagerFactory("Practica_JPA_PU");
    }
    public static EntityManager getEntityManager() {
        return emFactory.createEntityManager();
    }
}
```

Nombre de la unidad de persistencia

7. Crear una clase con el nombre "**EstudiantesModel**" dentro del paquete sv.edu.udb.www.model. Escriba el siguiente código fuente dentro de la clase.

```
package sv.edu.udb.www.model;

import sv.edu.udb.www.utils.JpaUtil;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.persistence.Query;
import sv.edu.udb.www.entities.EstudianteEntity;

public class EstudiantesModel {

    public List<EstudianteEntity> listarEstudiante() {
        //Obtengo una instancia de EntityManager
        EntityManager em = JpaUtil.getEntityManager();
        try {
            Query consulta = em.createNamedQuery("EstudianteEntity.findAll");
            //El método getResultList() de la clase Query permite obtener
            // la lista de resultados de una consulta de selección
            List<EstudianteEntity> lista = consulta.getResultList();
            em.close();// Cerrando el EntityManager
            return lista;
        } catch (Exception e) {
            em.close();
            return null;
        }
    }

    public EstudianteEntity obtenerEstudiante(String carnet) {
        EntityManager em = JpaUtil.getEntityManager();
        try {
            //Recupero el objeto desde la BD a través del método find
            EstudianteEntity estudiante = em.find(EstudianteEntity.class,
carnet);
```

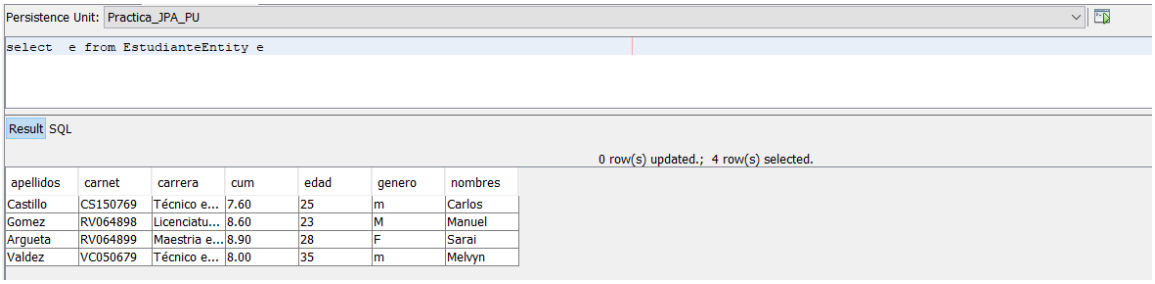
Esta consulta nombrada está en la clase EstudianteEntity. Siéntase libre de agregar sus propias consultas nombradas dentro de las clases de entidad.

```
            em.close();
            return estudiante;
        } catch (Exception e) {
            em.close();
            return null;
        }
    }

    public int insertarEstudiante(EstudianteEntity estudiante) {
        EntityManager em = JpaUtil.getEntityManager();
        EntityTransaction tran = em.getTransaction();
        try {
            tran.begin();//Iniciando transacción
            em.persist(estudiante); //Guardando el objeto en la BD
            tran.commit();//Confirmando la transacción
            em.close();
            return 1;
        } catch (Exception e) {
            em.close();
            return 0;
        }
    }

    public int modificarEstudiante(EstudianteEntity estudiante) {
        EntityManager em = JpaUtil.getEntityManager();
        EntityTransaction tran = em.getTransaction();
        try {
            tran.begin();//Iniciando transacción
            em.merge(estudiante); //Actualizando el objeto en la BD
            tran.commit();//Confirmando la transacción
            em.close();
            return 1;
        } catch (Exception e) {
            em.close();
            return 0;
        }
    }

    public int eliminarEstudiante(String carnet) {
        EntityManager em = JpaUtil.getEntityManager();
        int filasBorradas = 0;
        try {
            //Recuperando el objeto a eliminar
            EstudianteEntity est = em.find(EstudianteEntity.class, carnet);
            if (est != null) {
                EntityTransaction tran = em.getTransaction();
                tran.begin();//Iniciando transacción
                em.remove(est);//Borrando la instancia
                tran.commit();//Confirmando la transacción
                filasBorradas = 1;
            }
            em.close();
            return filasBorradas;
        } catch (Exception e) {
            em.close();
            return 0;
        }
    }
}
```

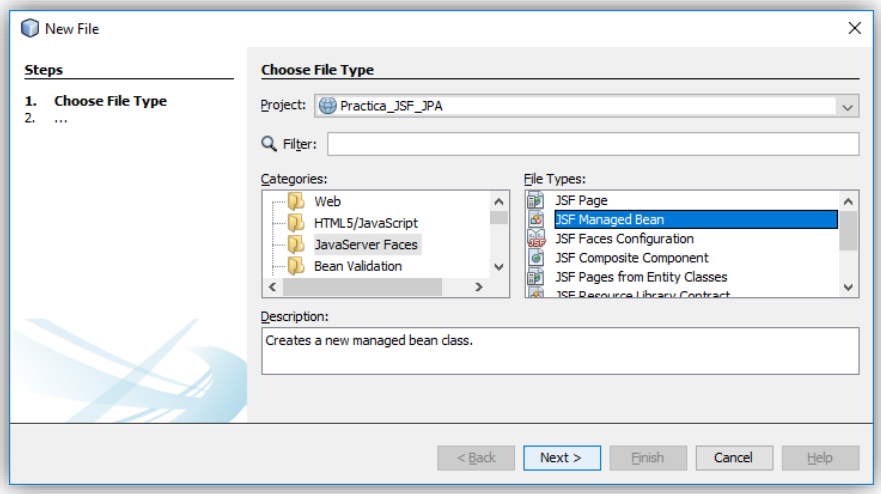
8. Proceda a crear una clase llamada "**JsfUtil**" dentro del paquete sv.edu.udb.www.utils. Ubique el siguiente código fuente dentro de la clase.

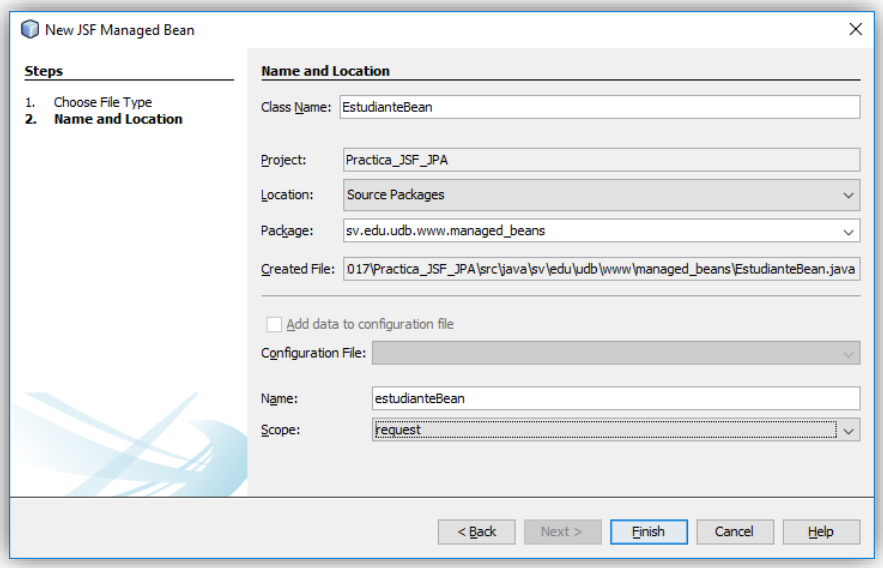
```
package sv.edu.udb.www.utils;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;

public class JsfUtil {
    /*Este método permite definir un mensaje de error.
        Recibe como parámetro del id del control asociado al error
       y el propio mensaje de error */
   public static void setErrorMessage(String idClient, String msg) {
        FacesMessage mensaje = new
FacesMessage(FacesMessage.SEVERITY_ERROR, msg, null);
        FacesContext.getCurrentInstance().addMessage(idClient, mensaje);
    }

    /*Este método permite definir un mensaje de tipo flash (mensaje que
se elimina de forma automatica en cuanto una vista lo muestre)*/
    public static void setFlashMessage(String name, String msg) {
        FacesContext.getCurrentInstance().getExternalContext()
                .getFlash().put(name, msg);
    }

    public static HttpServletRequest getRequest(){
        return (HttpServletRequest) FacesContext.getCurrentInstance()
                                .getExternalContext().getRequest();
    }
}
```

9. Hasta este punto, puede consultar con su docente como crear una clase de Test para realizar las pruebas respectivas, adicional a eso consultar como utilizar la consola de JPQL, que el IDE de Netbeans proporciona.



10. Dentro del paquete **sv.edu.udb.www.managed_bean** debe crear un nuevo bean gestionado (con alcance de request) con el nombre **EstudianteBean**. Para ello ubíquese sobre el paquete, de click derecho y seleccione la opción "New>Other>JavaServer Faces>JSF Managed Bean".





11. Agregue el siguiente código fuente en el bean gestionado que acaba de crear.

```
package sv.edu.udb.www.managed_beans;

import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import sv.edu.udb.www.entities.EstudianteEntity;
import sv.edu.udb.www.model.EstudiantesModel;
import sv.edu.udb.www.utils.JsfUtil;

@ManagedBean
@RequestScoped
public class EstudianteBean {

    EstudiantesModel modelo = new EstudiantesModel();
    private EstudianteEntity estudiante;
    private List<EstudianteEntity> listaEstudiantes;

    public EstudianteBean() {
```

```
        estudiante = new EstudianteEntity();
    }

    public EstudianteEntity getEstudiante() {
        return estudiante;
    }

    public void setEstudiante(EstudianteEntity estudiante) {
        this.estudiante = estudiante;
    }

    public List<EstudianteEntity> getListaEstudiantes() {
        /* Notese que se llama al método listarEstudiantes
           para obtener la lista de objetos a partir de la bd */
        return modelo.listarEstudiante();
    }


    public String guardarEstudiante() {
        if (modelo.insertarEstudiante(estudiante) != 1) {
            JsfUtil.setErrorMessage(null, "Ya se registró un alumno con este
carnet");
            return null;//Regreso a la misma página
        } else {
            JsfUtil.setFlashMessage("exito", "Alumno registrado
exitosamente");
            //Forzando la redirección en el cliente
            return "registroEstudiantes?faces-redirect=true";
        }
    }

}
```

12. Ubique la carpeta “resources” proporcionada por su docente, dentro del directorio Web Pages de su proyecto.
13. Agregue el JAR de JSTL a las librerías de su proyecto.
14. Agregue una nueva JSF page a su proyecto. El nombre de la página deberá ser “**registroEstudiantes.xhtml**” y deberá estar colocada en la raíz del directorio web pages. La página debe tener el siguiente contenido:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:c="http://java.sun.com/jsp/jstl/core">
  <h:head>
    <title>Registro de estudiantes</title>
    <h:outputStylesheet name="css/bootstrap.min.css" />
    <h:outputStylesheet name="css/alertify.core.css" />
    <h:outputStylesheet name="css/alertify.default.css" />
    <h:outputScript name="js/alertify.js"/>
  </h:head>
```

```
    <h:body>
        <div class="container">
            <div class="row">
                <div class="col-md-12">
                    <h2>Registro de estudiantes</h2>
                    <h:form id="miFormulario">
                        <h:panelGroup layout="block" styleClass="alert alert-danger"
                                rendered="#{not empty facesContext.messageList}">
                            <h:messages  />
                        </h:panelGroup>
                        <h:panelGrid columns="4" styleClass="table">
                            <h:outputLabel for="carnet" value="Carnet:"/>
                            <h:inputText id="carnet" styleClass="form-control"
                                    value="#{estudianteBean.estudiante.carnet}"
                                    required="true"
                                    requiredMessage="Debes ingresar el carnet"
                                    validatorMessage="Carnet no valido">
                                <f:validateRegex pattern="[A-Z]{2}[0-9]{6}"/>
                            </h:inputText>
                            <h:outputLabel for="nombre" value="Nombres:"/>
                            <h:inputText id="nombre" styleClass="form-control"
                                    value="#{estudianteBean.estudiante.nombres}"
                                    required="true"
                                    requiredMessage="Debes ingresar el nombre"/>

                            <h:outputLabel for="apellidos" value="Apellidos:"/>
                            <h:inputText id="apellidos" styleClass="form-control"
                                    value="#{estudianteBean.estudiante.apellidos}"
                                    required="true"
                                    requiredMessage="Debes ingresar el apellido"/>

                            <h:outputLabel for="edad" value="Edad:"/>
                            <h:inputText id="edad" value="#{estudianteBean.estudiante.edad}"
                                    required="true" styleClass="form-control"
                                    requiredMessage="Debes ingresar la edad"
                                    validatorMessage="La edad debe ser mayor o igual a 18 años"
                                    converterMessage="La edad debe ser númerica">
                                <f:validateLongRange minimum="18"/>
                            </h:inputText>
                            <h:outputLabel for="cum" value="CUM:"/>
                            <h:inputText id="cum" value="#{estudianteBean.estudiante.cum}"
                                    required="true"  styleClass="form-control"
                                    requiredMessage="Debes ingresar el cum"
                                    validatorMessage="El cum debe estar entre 0.0 y 10.0"
                                    converterMessage="El cum debe ser numerico">
                                <f:validateDoubleRange minimum="0.0" maximum="10.0"/>
                            </h:inputText>
                            <h:outputLabel for="genero" value="Genero:"/>
                            <h:selectOneMenu id="genero" styleClass="form-control"
value="#{estudianteBean.estudiante.genero}">
                                <f:selectItem itemValue="f" itemLabel="Femenino"/>
                                <f:selectItem itemValue="m" itemLabel="Masculino"/>
                            </h:selectOneMenu>
```

```
                        <h:outputLabel for="carrera" value="Carrera:"/>
                        <h:selectOneMenu id="carrera" styleClass="form-control"
value="#{estudianteBean.estudiante.carrera}">
                            <f:selectItem itemValue="Técnico en computación"/>
                            <f:selectItem itemValue="Técnico en electrónica"/>
                            <f:selectItem itemValue="Ingeniería aeronáutica"/>
                            <f:selectItem itemValue="Ingeniería en telecomunicaciones"/>
                            <f:selectItem itemValue="Licenciatura en filosofia"/>
                        </h:selectOneMenu>

                    </h:panelGrid>
                    <h:commandButton  value="Guardar" styleClass="btn btn-success"
                                      action="#{estudianteBean.guardarEstudiante()}" style="margin-right:
10px"/>
                    <h:commandButton type="reset" value="Limpiar" styleClass="btn btn-danger"/>
                </h:form>
            </div>
        </div>

        <div class="row">
            <h:form >
                <h:dataTable styleClass="table" value="#{estudianteBean.listaEstudiantes}"
var="est" rendered="#{estudianteBean.listaEstudiantes.size()>0}">
                    <h:column>
                        <f:facet name="header">
                            <h:outputLabel value="Carnet"/>
                        </f:facet>
                        <h:outputText value="#{est.carnet}"/>
                    </h:column>
                    <h:column>
                        <f:facet name="header">
                            <h:outputLabel value="Nombre"/>
                        </f:facet>
                        <h:outputText value="#{est.nombres}"/>
                    </h:column>
                    <h:column>
                        <f:facet name="header">
                            <h:outputLabel value="Apellido"/>
                        </f:facet>
                        <h:outputText value="#{est.apellidos}"/>
                    </h:column>

                    <h:column>
                        <f:facet name="header">
                            <h:outputLabel value="Carrera"/>
                        </f:facet>
                        <h:outputText value="#{est.carrera}"/>

                    </h:column>
                    <h:column>
                        <f:facet name="header">
                            <h:outputLabel value="Edad"/>
                        </f:facet>
```

```
                    <h:outputText value="#{est.edad}"/>
                </h:column>
                <h:column>
                    <f:facet name="header">
                        <h:outputLabel value="CUM"/>
                    </f:facet>
                    <h:outputText value="#{est.cum}"/>
                </h:column>
                <h:column>
                    <f:facet name="header">
                        <h:outputLabel value="Genero"/>
                    </f:facet>
                    <h:outputText value="#{(est.genero== 'm'.charAt(0))? 'Masculino' :
'Femenino'}"/>
                </h:column>

            </h:dataTable>
        </h:form>
    </div>
</div>
<h:outputScript>
    <c:if test = "#{not empty flash.exito}" >
        alertify.success('#{flash.exito}');
    </c:if>
</h:outputScript>
</h:body>
</html>
```

15. Hasta este punto la aplicación debería ser capaz de registrar nuevos alumnos y mostrar los alumnos registrados mediante un datatable. La página creada debería lucir de la siguiente manera:

Pruebe las validaciones y asegúrese que su aplicación funcione correctamente.

16. Ahora procederemos a agregar la funcionalidad “eliminar”. En nuestro modelo, el método eliminar ya está listo, por tanto, solo basta con realizar los siguientes cambios:

- Agregar el siguiente método dentro de su bean gestionado EstudianteBean.

```
public String eliminarEstudiante() {
        // Leyendo el parametro enviado desde la vista
        String carnet= JsfUtil.getRequest().getParameter("carnet");

        if (modelo.eliminarEstudiante(carnet) > 0) {
            JsfUtil.setFlashMessage("exito", "Estudiante eliminado
exitosamente");
        }
        else{
            JsfUtil.setErrorMessage(null, "No se pudo borrar a este alumno");
        }
        return "registroEstudiantes?faces-redirect=true";

    }
```

- Agregar una columna al datatable que muestra la información. En esta nueva columna colocaremos un botón que permitirá borrar los datos de un estudiante.

```
<h:column>
        <f:facet name="header">
              <h:outputLabel value="Operaciones"/>
        </f:facet>
         <h:commandButton value="Eliminar" styleClass="btn btn-danger"
                                  action="#{estudianteBean.eliminarEstudiante}"
                                  onclick="return confirmarEliminacion();">
                    <f:param name="carnet" value="#{est.carnet}"/>
          </h:commandButton>
 </h:column>
```

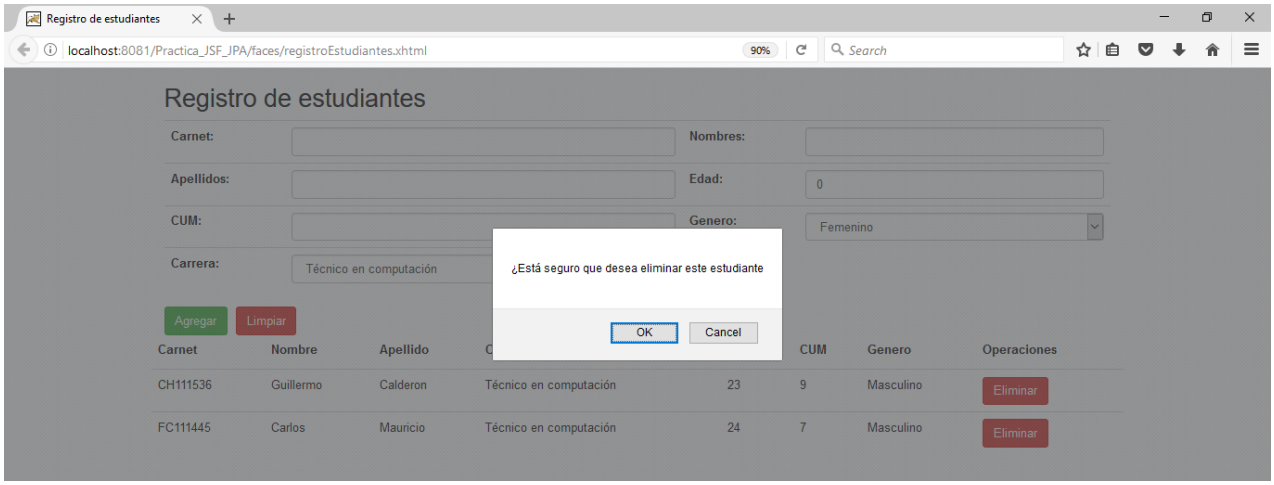
Antes de ejecutar el action se ejecutará el método javascript

Pasándole el carnet como parámetro al método eliminarEstudiante() del bean gestionado.

- Dentro de la etiqueta <h:outputScript> (parte inferior de la página) debe colocar el siguiente codigo js:

```
function confirmarEliminacion(){
        return confirm("¿Está seguro que desea eliminar este estudiante");
 }
```

Realizados estos cambios, la operación eliminar debería funcionar correctamente.



IV. EJERCICIOS COMPLEMENTARIOS

- Implemente la operación “modificar” en el ejemplo de la guía. Para esta tarea, ya tiene los métodos listos en la clase EstudiantesModel, por tanto, únicamente deberá realizar los siguientes cambios:
  - Agregar un botón “modificar” a cada uno de las filas del datatable. Al pulsar este botón, los datos del estudiante seleccionado deben colocarse en el formulario de la parte superior para su edición.
  - Después de pulsar el botón de “modificar” y editar los datos del estudiante, el botón “guardar” deberá actualizar el registro. Es decir, se debe modificar el método guardarEstudiante() del bean gestionado de tal forma que si el carnet no está registrado en la bd se proceda a insertar y si el carnet ya ha sido registrado anteriormente se proceda a modificar el registro.