

CURSO DE JAVA EE

ASOCIACIONES EN JPA



Por el experto: Ing. Ubaldo Acosta



CURSO DE JAVA EE

www.globalmentoring.com.mx

Hola, te saluda Ubaldo Acosta. Bienvenida o bienvenido nuevamente. Espero que estés listo para comenzar con esta lección.

Vamos a estudiar el tema de Asociaciones en JPA.

¿Estás listo? Ok, ¡Vamos!

RELACIONES EN JPA

TIPOS DE RELACIONES:

- ✓ Uno a Uno: @OneToOne
- ✓ Uno a Muchos: @OneToMany
- ✓ Muchos a Uno: @ManyToOne
- ✓ Muchos a Muchos: @ManyToMany

DIRECCIONALIDAD EN LAS RELACIONES:

- ✓ Unidireccional: Se define el atributo de relación solo en una clase.
- ✓ Bidireccional: Se define los atributos de relación en ambas clases.

CURSO DE JAVA EE

www.globalmentoring.com.mx

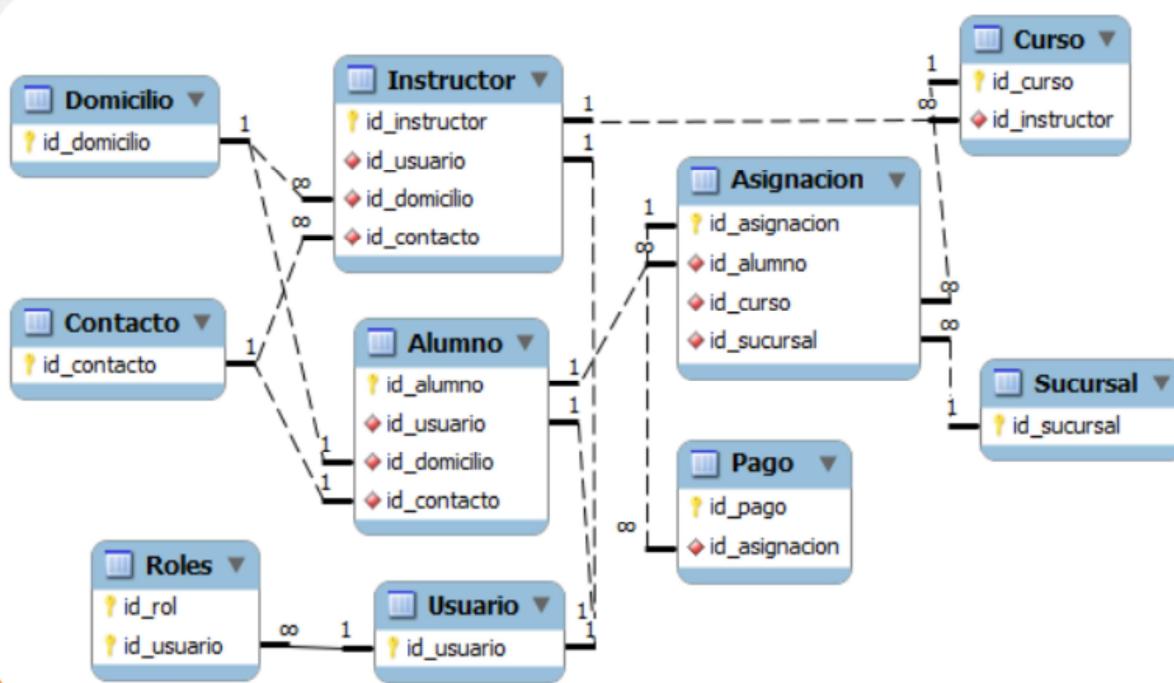
Normalmente los objetos de entidad, en un sistema con bases de datos relacionales, mantienen asociaciones con uno o más objetos. Los tipos de relaciones en JPA son las mismas que se manejan en la teoría de bases de datos relacionales.

- ✓ 1 a 1
- ✓ 1 a Muchos o Muchos a 1
- ✓ Muchos a Muchos

JPA soporta las relaciones mencionadas en los archivos de mapeo de cada clase de Entidad o en las clases Java utilizando anotaciones.

Las relaciones también tienen navegabilidad (directionality), esto quiere decir que podemos acceder a los objetos con los que tenemos relación de manera unidireccional o bidireccional. Esto lo logramos debido a que en los objetos de entidad manejamos un atributo que identifica el objeto(s) de entidad(es) con el que tenemos relación. Cuando cada objeto de entidad se apunta uno al otro por medio de este atributo o colección, se dice que es una relación bidireccional, y si por solamente una entidad apunta a la otra, la relación se conoce como unidireccional. Esto lo revisaremos a continuación.

EJEMPLO DIAGRAMA ENTIDAD RELACIÓN



En la figura podemos observar un diagrama entidad relación con el que analizaremos las relaciones mencionadas:

- ✓ 1 a 1
- ✓ 1 a Muchos o Muchos a 1
- ✓ Muchos a Muchos

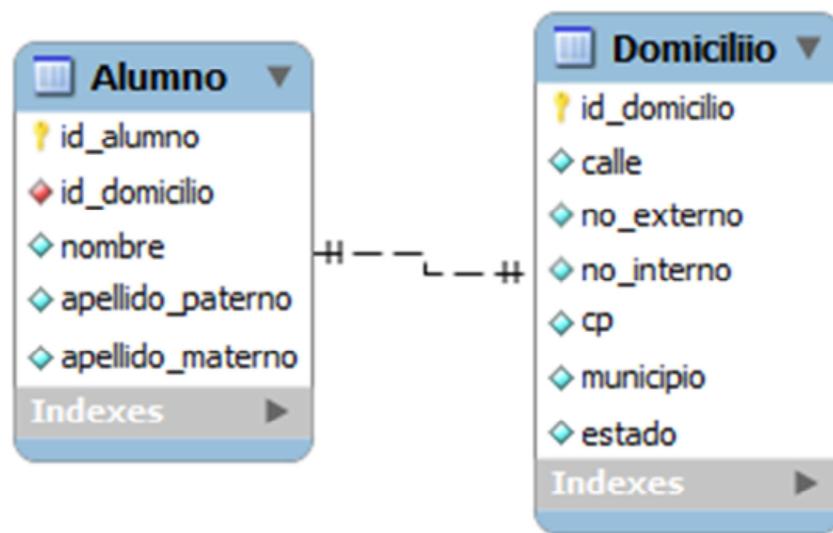
A continuación realizaremos un análisis de cada relación descrita.

Para ver la imagen completa pueden ver el siguiente link:

<http://icursos.net/cursos/JavaEE/Leccion05/EsquemaEntidadRelacionSGAFull.png>

EJEMPLO DE RELACIÓN 1 A 1

- Un Alumno tiene un Domicilio



www.globalmentoring.com.mx

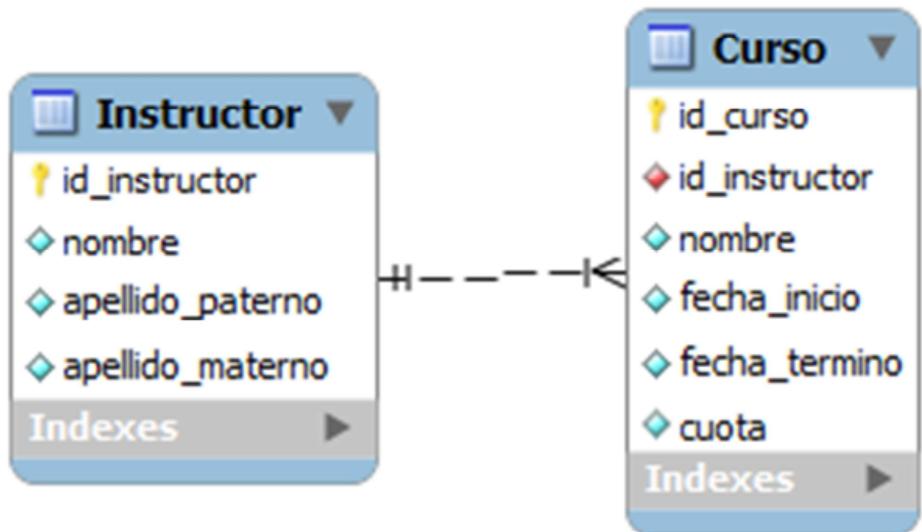
En la figura podemos observar una relación de 1 a 1, en la cual una entidad Alumno tiene una relación con sólo un domicilio, y viceversa, esto es, la cardinalidad entre las entidades es de 1 a 1.

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. Este tema lo estudiaremos más adelante.

EJEMPLO DE RELACIÓN 1 A MUCHOS

- Un Instructor imparte muchos Cursos



CURSO DE JAVA EE

www.globalmentoring.com.mx

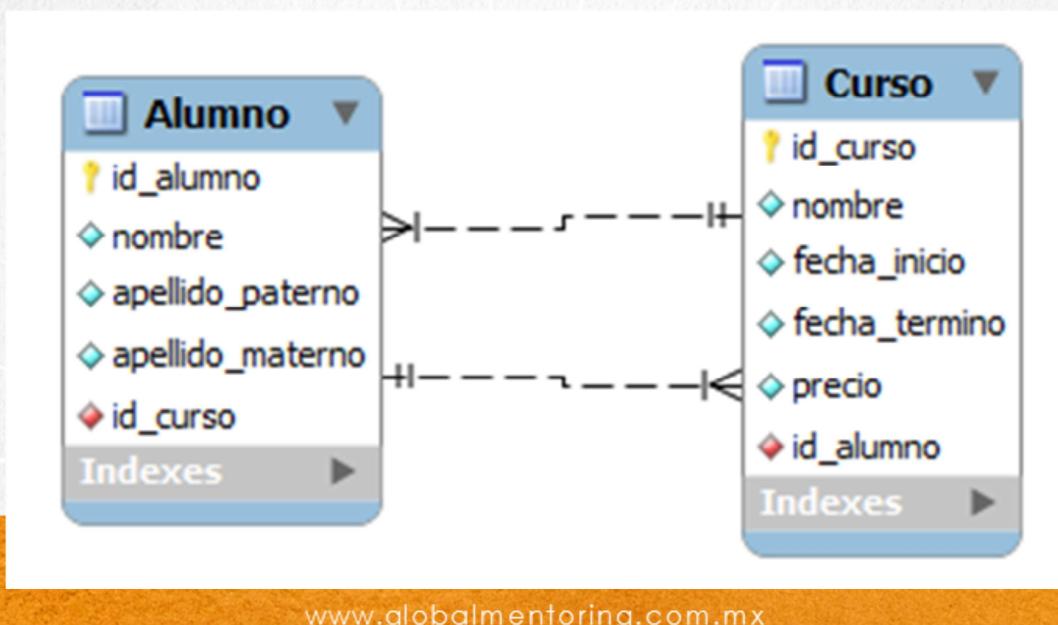
Cuando un objeto de Entidad está asociado con una colección de otros objetos de Entidad, es más común representar esta relación como una relación de Uno a Muchos. Por ejemplo, en la figura podemos observar la relación de un Instructor el cual puede tener asignados varios Cursos (relación de 1 a muchos).

Si queremos saber desde la clase **Curso** qué instructor tiene asociado, deberemos agregar el mapeo bidireccional (**ManyToOne**) hacia **Alumno**. Y en la clase **Alumno**, 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**, dependiendo si queremos manejar orden o no, respectivamente.

Si no queremos manejar una relación bidireccional, basta con eliminar la definición de alguna de las clases y así tendremos una relación unidireccional.

EJEMPLO DE RELACIÓN MUCHOS A MUCHOS

- Un Alumno tiene Muchos Cursos y un Curso tiene Muchos Alumnos



www.globalmentoring.com.mx

A continuación observamos un ejemplo de muchos a muchos (* a *).

Aquí podemos observar que un Alumno puede estar relacionado a un Curso, pero a su vez un Curso puede tener muchos Alumnos.

Este tipo de relaciones Muchos a Muchos se pueden representar también con JPA, sin embargo este tipo de relaciones es mejor aplicar el concepto de normalización de base de datos para simplificar este tipo de relaciones y en lugar de tener relaciones muchos a muchos, tengamos relaciones Uno a Muchos o Uno a Uno.

Si no normalizamos este tipo de relaciones podemos caer en referencias circulares, debido a que un Alumno puede tener un Curso, pero un Curso puede tener a su vez al mismo Alumno. Para corregir esto vamos a normalizar (simplificar) esta relación, para que quede como una relación de Uno a muchos o de Uno a Uno. Veamos como hacer esto.

EJEMPLO DE RELACIÓN MUCHOS A MUCHOS NORMALIZADA

- Un Alumno tiene Muchos Cursos y un Curso tiene Muchos Alumnos



CURSO DE JAVA EE
www.globalmentoring.com.mx

Podemos observar la normalización de la relación anterior, en este caso la tabla de Alumno que se relacionaba directamente con la tabla de Curso.

Ya no se relaciona directamente, si no que ahora hemos puesto una tabla intermedia, esta tabla intermedia se le conoce como tabla transitiva y lo que estamos haciendo es convertir la relación de Muchos a Muchos a una relación de Uno a Muchos de Alumnos hacia Asignación y una relación de Uno a Muchos de la tabla de Curso hacia la tabla Asignación.

Las llaves primarias de **id_alumno** y de **id_curso** las agregamos como llaves foráneas a la tabla de Asignación y a su vez combinándolas se convierte en una llave primaria compuesta para la tabla de Asignación, esas son algunas de las mejores prácticas que vamos a utilizar en JPA. Sin embargo también se puede agregar una llave primaria simple a la tabla de Asignación y así no manejar llaves primarias compuestas. Cualquiera de las dos opciones es factible.

A pesar de que pudimos haber manejado una relación de Muchos a Muchos directamente con JPA, normalizar este tipo de relaciones no permite administrar más fácilmente las relaciones de Uno a Muchos o Uno a Uno, y por lo tanto más sencillas de dar mantenimiento.

Además, una vez que hemos generado una tabla transitiva para el Alumno y el Curso, podemos agregar ciertos atributos de la nueva relación, por ejemplo un Alumno al estar tomando un Curso puede estar en cierto horario, entre otro tipo de atributos. En los ejercicios de mapeo de relaciones veremos cada uno de los casos estudiados hasta el momento.

FETCHING RELACIONES

Lazy Loading: Carga Retardada

```

@Entity
public class Alumno implements Serializable {
    private static final long serialVersionUID = 1L;

    //Atributos...

    //bi-directional many-to-one association to Domicilio
    //relación de tipo Lazy, No se recuperan los datos
    //del objeto Domicilio, sino hasta que son solicitados
    @OneToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="id_domicilio")
    private Domicilio domicilio;
}

```

Eager Loading: Carga Inmediata

```

@Entity
public class Alumno implements Serializable {
    private static final long serialVersionUID = 1L;

    //Atributos...

    //bi-directional many-to-one association to Domicilio
    //relación de tipo Eager, SI se recuperan los datos
    //del objeto Domicilio desde que se realiza la consulta
    @OneToOne(fetch=FetchType.EAGER)
    @JoinColumn(name="id_domicilio")
    private Domicilio domicilio;
}

```

CURS
www.globomentoring.com.mx

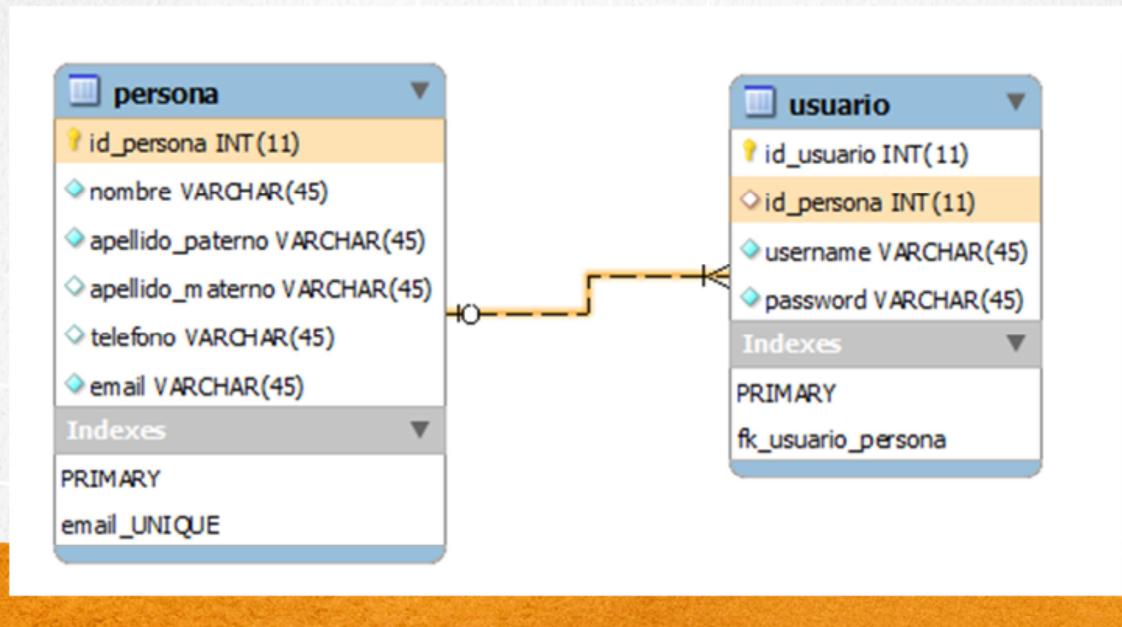
Con JPA podemos especificar el comportamiento de las colecciones de objetos relacionadas a nuestra clase de Entidad. Existen dos tipos:

- 1) Lazy Loading (Carga Retardada): La carga retardada significa que la colección definida en la clase de Entidad respectiva no va a ser recuperada, de esta manera evitamos realizar demasiados queries por cuestiones de las colecciones de objetos asociadas. Este es el comportamiento por default en las relaciones OneToMany y ManyToMany. En otros casos debemos especificarlo de manera explícita como el ejemplo mostrado.
- 2) Eager Loading (Carga Inmediata): La carga inmediata significa que las colecciones asociadas a una Entidad son recuperadas, y por lo tanto debemos tener cuidado con este comportamiento debido a que las colecciones marcadas con Eager son recuperadas junto con el objeto de Entidad, realizando más consultas que si solo se recuperara el Objeto de Entidad en cuestión. Sin embargo este no es el comportamiento por default por lo que debemos especificar el tipo de carga que deseamos realizar para que el tipo Eager o carga inmediata de la asociación respectiva se ejecute.

Este tipo de comportamiento de carga de las colecciones, también conocido como Fetch, lo podemos especificar ya sea en las relaciones definidas en nuestra clase de entidad o en queries de tipo JPQL o Criteria de JPA.

Más adelante veremos algunos ejemplos de cómo utilizar esta característica.

GUARDADO EN CASCADA



www.globalmentoring.com.mx

Una de las características de persistencia en JPA es la posibilidad de persistir un objeto de Entidad junto con sus relaciones. Como podemos observar en la figura, tenemos el objeto Persona, y a su vez tenemos la relación con el objeto Usuario, poner un ejemplo.

Por default si queremos persistir un objeto Usuario, no guardará los datos de Persona, sin embargo si indicamos en el mapeo de estas relaciones que también guarde la información, utilizando la anotación de persistencia en cascada, entonces se guardará no solamente el objeto Usuario, sino que también se podría guardar la información asociada con la entidad de Persona.

La anotación es similar a esta: `@OneToOne(cascade=CascadeType.PERSIST)`, o también `@OneToMany(cascade=CascadeType.PERSIST)`, dependiendo del tipo de relación que tengamos.

Vamos a ver más adelante un ejercicio incluyendo la persistencia en cascada.

CURSO ONLINE

JAVA EMPRESARIAL JAVA EE

Por: Ing. Ubaldo Acosta

**CURSO DE JAVA EE**www.globalmentoring.com.mx

En Global Mentoring promovemos la Pasión por la Tecnología Java. Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados, y así te conviertas en un experto programador Java.

Además agregamos nuevos cursos para que continúes con tu preparación como programador Java profesional. A continuación te presentamos nuestro listado de cursos:

- | | |
|---|--|
| <ul style="list-style-type: none">✓ Lógica de Programación✓ Fundamentos de Java✓ Programación con Java✓ Java con JDBC✓ HTML, CSS y JavaScript✓ Servlets y JSP's✓ Struts Framework | <ul style="list-style-type: none">✓ Hibernate Framework & JPA✓ Spring Framework✓ JavaServer Faces✓ Java EE (EJB, JPA y Web Services)✓ JBoss Administration✓ Android con Java✓ HTML5 y CSS3 |
|---|--|

Datos de Contacto:Sitio Web: www.globalmentoring.com.mxEmail: informes@globalmentoring.com.mx