UNIVERSIDAD DEL QUINDÍO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
Actualizado:	Einer Zapata G.
Duración estimada en minutos:	240
Docente:	Christian Andrés Candela
Guía no.	05
Nombre de la guía:	Entidades – Relaciones Entre Entidades

Información de la Guía

Objetivo

Estudiar el uso de las entidades, las relaciones entre estas y su aplicación en el modelamiento de datos.

Conceptos Básicos

Manejo de Eclipse, Java, Bases de Datos, JDBC, XML, GlassFish

Contextualización Teórica

Uno de los aspectos más importantes es la definición de relaciones entre entidades. Las relaciones pueden ser bidireccionales o unidireccionales, es decir, se puede establecer la relación solo en una de las entidades (unidireccional), o se puede establecer la relación en ambas entidades a través de atributos permitiendo así la navegación en ambos sentidos (bidireccional).

En toda relación hay una entidad propietaria (owner), este aspecto de las relaciones define como se propagan las modificaciones. En las relaciones bidireccionales el lado opuesto al propietario es conocido como inverso, y debe referenciar al propietario de la relación usando el elemento mappedBy en la anotación de la relación.

Existen 4 tipos de relaciones básicos:

- Uno a uno: Para la creación de las relaciones uno a uno se debe marcar el o los atributos de la relación con la anotación javax.persistence.OneToOne. Este tipo de relación se establece entre entidades donde la relación entre la una y la otra es exclusiva. En esta relación la entidad propietaria es quien referencia a la otra. Ejemplo un país tiene un presidente. Existe una relación uno a uno entre país y presidente. Para este caso, la entidad propietaria es el presidente.

```
@Entity public class Pais implements Serializable{
    …
    @OneToOne(mappedBy=”pais”)
    private Presidente presidente;
}

@Entity public class Presidente extends Persona{
    …
```

```
    @OneToOne
    private Pais país;
}
```

- Uno a muchos: Para la creación de la relaciones uno a muchos se debe marcar el atributo de la relación con la anotación javax.persistence.OneToMany. Este tipo de relación se establece cuando una entidad se relaciona con varias instancias de otra entidad. Para esta relación la entidad propietaria al igual que en la anterior es quien referencia a la otra, es decir, quien está del lado de los muchos. Ejemplo un país tiene muchos departamentos. En este ejemplo existe una relación uno a muchos entre país y los departamentos. Para este caso, la entidad propietaria es el departamento.

```
@Entity public class Pais implements Serializable{
    …
    @OneToMany(mappedBy=”pais”)
    private List<Departamento> departamentos;
}

@Entity public class Departamento implements Serializable{
    …
    @ManyToOne
    private Pais país;
}
```

- Muchos a uno: Para la creación de las relaciones muchos a uno se debe marcar el o los atributos de la relación con la anotación javax.persistence.ManyToOne. Este tipo de relación se establece entre entidades cuando varias instancias de una de las entidades están relacionadas con una instancia de la otra entidad. Para esta relación la entidad propietaria es quien referencia a la otra, es decir, quien está del lado de los muchos. Por ejemplo, varios automóviles pueden ser de la misma marca. Para este caso la entidad propietaria es el Automovil.

```
@Entity public class Marca implements Serializable{
    …
    @OneToMany(mappedBy=”marca”)
    private List<Automovil> automoviles;
}

@Entity public class Automovil implements Serializable{
    …
    @ManyToOne
    private Marca marca;
}
```

- Muchos a muchos: Para la creación de las relaciones muchos a muchos se debe marcar el o los atributos de la relación con la anotación javax.persistence.ManyToMany. Este tipo de relación se establece entre entidades cuando una instancia de una de las entidades está relacionada con muchas instancias de la otra, y a su vez, una instancia de la otra entidad está relacionada con muchas instancias de la primera entidad. En este tipo de relación es particularmente difícil identificar cuál de las entidades es la propietaria de la relación, ya que cualquiera de las entidades puede referenciar a la otra. Por ejemplo, un estudiante puede estar registrado en muchas materias, y a su vez en una materia pueden estar registrados muchos estudiantes. Para este caso podemos decir que desde un punto de vista lógico la materia referencia al estudiante, por lo tanto, es la materia la entidad que hace las veces de propietaria de la relación.

```
@Entity public class Materia implements Serializable{
```

```
    …
    @ManyToMany
    private List<Estudiante> estudiantes;
}

@Entity public class Estudiante implements Serializable{
    …
    @ManyToMany(mappedBy=”estudiantes”)
    private List<Materia> materias;
}
```

Además de los 4 tipos de relaciones enunciadas, existe otro tipo de relación entre entidades el cual es conocido como herencia. Este tipo de relación es muy diferente a los anteriores, y se crea a través del lenguaje por medio del extends. Por ejemplo, un presidente es una persona, por lo tanto, existe una relación de herencia entre el presidente y la persona.

Al momento de crear la relación se puede hacer uso de 3 diferentes estrategias que si bien no representan una diferencia entre las entidades si origina cambios significativos en cómo se generan las tablas en la base de datos.

- Una Tabla: En este tipo de estrategia se crea en la base de datos una única tabla sin importar cuantas entidades extiendan de la entidad principal.

```
@Entity
@Inheritance
@DiscriminatorValue("Mamifero")  //OPCIONAL
@DiscriminatorColumn(name="tipo" ) //OPCIONAL
public class Mamifero {
    …
}

@Entity
@DiscriminatorValue("Gato")  //OPCIONAL
public class Gato extends Mamifero{
    …
}
```

- Una tabla por clase concreta: En este tipo de estrategia se crea una tabla por cada entidad presente en la relación, salvo en los casos en los que la entidad principal sea marcada como superclase (@MappedSuperclass), en cuyo caso no existirá como tabla. Los atributos de la entidad principal se duplicarán en las tablas de sus subtipos.

```
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS )
@MappedSuperclass
public class Mamifero {
    …
}

@Entity
public class Gato extends Mamifero{
    …
}
```

- Relación por medio de joins: En este tipo de estrategia se crea una tabla por cada una de las entidades que participan de la relación, pero no se duplican los atributos de la entidad principal, en su lugar se crea una llave foránea en las tablas generadas por las entidades hijas.

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED )
public class Mamifero {
      …
}

@Entity
@DiscriminatorValue("Mamifero ")  //OPCIONAL
public class Gato extends Mamifero{
      …
}
```

Precauciones y Recomendaciones

Verifique que al crear las relaciones haga uso de List, Collection, Map… pero no de tipos específicos como ArrayList o HashMap.

Artefactos

Se requiere tener instalado el JDK y un IDE para el desarrollo de aplicaciones (Eclipse JEE en su última versión), un servidor de aplicaciones que cumpla con las especificaciones de JEE, para esta práctica Glassfish y el motor de base de datos Mysql.

Evaluación o Resultado

Se espera que el alumno logre modelar entidades lógicas de una aplicación y por medio de ellas se cree tablas en la base de datos que las representen de forma adecuada.

Procedimiento

1. Para el desarrollo de esta guía necesitara una base de datos en mysql, un proyecto de tipo maven con soporte para el uso de JPA. Y una conexión a dicha base de datos para ser usada en la generación de las tablas.

2. Para continuar con esta guía se hará uso de las clases elaboradas al modelar el sistema propuesto como trabajo final. Se ira identificando y trabajando con las relaciones existentes entre dichas clases.

3. Identifique y marque en su diagrama las relaciones 1 a 1.

4. Identifique y marque en su diagrama las relaciones n a n.

5. Identifique y marque en su diagrama las relaciones 1 a n y n a 1.

6. Identifique las relaciones de herencia en su diagrama.

7. **Para todas las relaciones de herencia en su diagrama (si las hay) realice los pasos del 8 al 11. En**

caso de no tener relaciones de herencia proceda con el paso 10.

8. Identifique un par de clases cuya relación sea de herencia, la herencia es un tipo de relación uno a uno entre las clases. En la clase madre o clase principal adicione la anotación @Entity o @MappedSuperClass según considere necesario. Tras estos pasos deberá seleccionar el nombre de la clase y en la pestaña JPA Detalis busque las propiedades relacionadas a la "inheritance" y en la propiedad estrategia seleccione la estrategia que considere adecuada, por ejemplo, ***Joined.***

9. En la clase hija o secundaria adicione la anotación @Entity. Configúrela adecuadamente para que funcione bajo la estrategia seleccionada en el punto anterior. Con esto deberá haber establecido la relación uno a uno de herencia entre ambas clases.

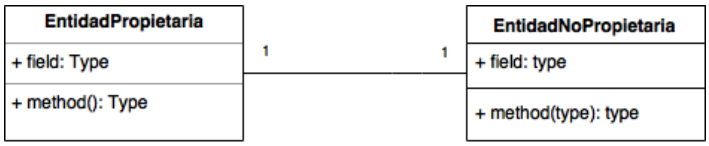
10. Para probar los resultados genere las tablas correspondientes a las entidades creadas. Para ello ejecute las pruebas de generación de tablas creadas en las guías pasadas.

11. Identifique dentro de sus clases todas aquellas en las que aplique este tipo de relación y configúrelas de forma adecuada.

12. **Para todas las relaciones uno a uno en su diagrama (si las hay) realice los pasos del 13 al 16. En caso de no tener relaciones 1 a 1 proceda con el paso 17**

13. Identifique dos clases entre las que haya una relación uno a uno.

14. Ahora en la entidad propietaria de la relación cree un atributo que representa a la otra entidad y en la pestaña JPA Details de clic en el vínculo que dirá click here (default one to one) y seleccione la relación uno a uno. Repita este proceso en la otra entidad perteneciente a la relación. Ejemplo




```
public class EntidadPropietaria implements Serializable {
        …
        @OneToOne
         private EntidadNoPropietaria instancia;
        …
}

public class EntidadNoPropietaria implements Serializable {
        …
         @OneToOne
         private EntidadPropietaria variable;
        …
}
```

15. Ahora en la clase secundaria (inverso), seleccione el atributo que representa la otra clase. En la propiedad **Join Strategy** cambie la estrategia a **Mapped by**. Ahora seleccione en el combobox el atributo que representa dicha clase en la otra. Ejemplo:

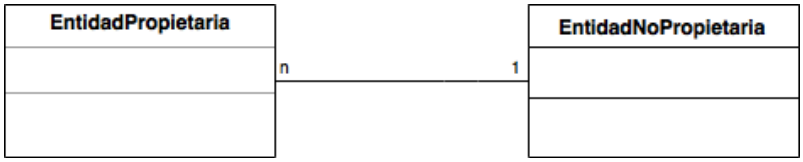
```
public class EntidadNoPropietaria implements Serializable {
        …
```

```
        @OneToOne(mappedBy= "instancia")
        private EntidadPropietaria variable;
        …
}
```

16. Para probar los resultados genere las tablas correspondientes a las entidades creadas. Verifique que las tablas se hayan creado correctamente.

17. **Para todas las relaciones uno a muchos – muchos a uno de su diagrama, realice los pasos del 18 al 21.**

18. Ahora identifique dos clases entre las que haya una relación uno a muchos. En la entidad propietaria cree un atributo que representa la instancia de la entidad con la que está relacionado y márquelo con la anotación @ManyToOne. De igual forma en la entidad secundaria de la relación cree un atributo de tipo lista que contenga las instancias de la otra entidad con las cuales está relacionada y márquelo con la entidad @OneToMany.




```
public class EntidadPropietaria implements Serializable {
        …
        @ManyToOne
        private EntidadNoPropietaria instancia;
        …
}

public class EntidadNoPropietaria implements Serializable {
        …
        @OneToMany
        private List< EntidadPropietaria > variable;
        …
}
```

19. Ahora en la entidad secundaria seleccione el atributo de tipo lista que lo asocia a la otra entidad. En la propiedad **Join Strategy** cambie la estrategia a **Mapped by** y seleccione en el combobox el atributo que representa en la instancia de la entidad en la entidad propietaria de la relación.

```
public class EntidadNoPropietaria implements Serializable {
        …
        @OneToMany( mappedBy= "instancia" )
        private List< EntidadPropietaria > variable;
        …
}
```

20. Para probar los resultados genere las tablas correspondientes a las entidades creadas. Verifique que las tablas se hayan creado correctamente.

21. Identifique dentro de sus clases todas aquellas en las que aplique este tipo de relación y configúrelas de forma adecuada.

22. **Para todas las relaciones muchos a muchos de su diagrama realice los pasos del 23 al 28.**

23. Identifique dos clases cuya relación sea de muchos a muchos. Determine cuál de las entidades pertenecientes a la relación es la propietaria de la misma. En ella cree un atributo de tipo lista que representa a la otra entidad, y en la pestaña JPA Details seleccione el tipo de relación que representa el atributo. En este caso muchos a muchos.


```
public class EntidadPropietaria implements Serializable {
        …
        @ManyToMany
        private List< EntidadNoPropietaria >instancia;
        …
}
```

24. Tome la otra entidad y cree el atributo de tipo lista que representa a la otra clase. En la pestaña JPA Details seleccione el tipo de relación que representa el atributo. En este caso muchos a muchos. En la Propiedad **Joining Strategy** cambie la estrategia a **Mapped By** y en el combobox de atributos que quedará a su disposición seleccione el atributo de la otra entidad que representa la relación con la entidad seleccionada.

```
public class EntidadNoPropietaria implements Serializable {
        …
        @ManyToMany( mappedBy= "instancia" )
        private List< EntidadPropietaria > variable;
        …
}
```

25. Para probar los resultados genere las tablas correspondientes a las entidades creadas. Verifique que las tablas se hayan creado correctamente.

26. Identifique dentro de sus entidades todas aquellas en las que aplique este tipo de relación y configúrelas de forma adecuada.

Para la próxima clase

Lea en y comprenda en que consiste y para qué es usado el formato de texto ".json".
Leer y entender en qué consiste Arquillian.