Argentina programa 4.0

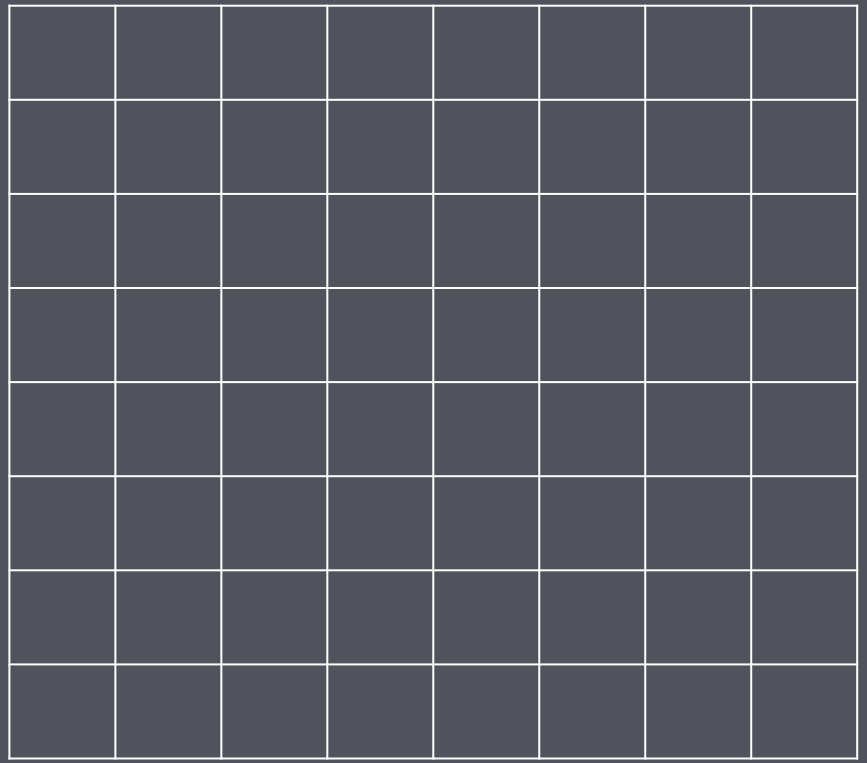
UNIVERSIDAD TECNOLOGICA NACIONAL

ORM II

Etapa 2 - Desarrollador Java Intermedio

ORM II

- Mapeo - Parte II



Proceso de Hidratación

Argentina programa 4.0

¿Cómo recupera un ORM un objeto desde la Base de Datos?

Cuando el ORM recupera un objeto desde la base de datos ejecuta el proceso de Hidratación.

Proceso de Hidratación

Argentina programa 4.0

El proceso de Hidratación consiste en:

1. ***Instanciar la clase*** del objeto que se quiere recuperar (previamente habiendo recuperado los datos mínimos mediante la ejecución de una sentencia SQL).

2. ***Popular el objeto***, es decir, asignarle a cada uno de sus atributos (aquellos no marcados como “lazy loading”) los valores recuperados desde la base de datos.

Hidratación Lazy vs Eager

Argentina programa 4.0

Cuando el ORM está realizando el proceso de Hidratación debe prestar atención a si debe, o no, popular un determinado atributo.

- Si el atributo está marcado como "***eager***", entonces lo seteará.
- Si el atributo está marcado como "***lazy***", entonces no lo seteará.

Hidratación Lazy vs Eager

Argentina programa 4.0

En el caso de que un atributo esté marcado como “lazy”, éste solamente será populado por el ORM, de forma transparente para el desarrollador y usuario, cuando sea llamado explícitamente.

- ***Lazy loading*** realiza la carga en memoria de los objetos sólo al momento de su utilización.
- ***Eager Loading*** realiza la carga en memoria de los objetos independientemente de si van a ser utilizados o no.

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@ElementCollection

Se utiliza cuando una entidad persistente cuenta con una colección de Strings, números o enumerados; que necesita ser persistida. A nivel DB generará una nueva tabla, la cual contendrá dos columnas: una con una FK a la entidad contenedora y otra columna con el valor que toma el String/número/enumerado.

- Debemos acompañarla con la anotación *@CollectionTable* para determinar cómo se llamará la nueva tabla.
- Debemos acompañarla, además, con *@Column* para determinar cómo se llamará la columna que guardará el valor.

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@ElementCollection

```
import javax.persistence.*;

@Entity
@Table(name=”empleado”)
public class Empleado {
    @ElementCollection
    @CollectionTable(name=”empleado_email”, joinColumns = @JoinColumn(name
= "empleado_id"))
    @Column(name=”email”)
    private List<String> emails;
}
```

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Enumerated

Se utiliza cuando queremos persistir un atributo cuyo tipo de dato es un Enumerado. La persistencia de este atributo generará una nueva columna en la DB, la cual puede tomar los valores ordinales o texto:

- ***EnumType.STRING***: permite persistir la enumeración por su nombre, lo que significa que será una columna alfanumérica.

- ***EnumType.ORDINAL***: esta estrategia persiste un valor entero que corresponde al valor ordinal o posición de valor en la enumeración.

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Enumerated

```
import javax.persistence.*;

@Entity
@Table(name=”empleado”)
public class Empleado {
    @Enumerated(value = EnumType.STRING)
    private Turno turnoTrabajado;
}
```

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Convert

Se utiliza cuando queremos convertir un atributo en “otra cosa” justo antes de persistirlo en la DB. Por ejemplo, queremos convertir un Enumerado “Day” que tiene los valores en inglés a Strings con los valores en español para que en nuestra DB se persistan en este último idioma.

Debemos especificar una clase “conversora”, responsable de convertir el dato inicial en el dato esperado; y de convertir el dato “modificado” en el dato “original” cuando lo recuperamos de la DB.

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Convert

```
import javax.persistence.*;

@Entity
@Table(name=”empleado”)
public class Empleado {
    @Convert(converter = DayEnumConverter.class)
    private Day diaTurnoCompleto;
}
```

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Convert

```
@Converter
public class DayEnumConverter implements AttributeConverter<Day, String> {
      @Override
    public String convertToDatabaseColumn(Day diaOriginal) {
           if(diaOriginal == Day.MONDAY){
                return “Lunes”;
           }
           //SIGUE
      }

      @Override
    public PersonName convertToEntityAttribute(String diaDesdeBase) {
           if(diaDesdeBase == “Lunes”) return Day.Monday;
           //SIGUE
    }
}
```

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Embedded *&* ***@Embeddable***

Se utiliza cuando una entidad A tiene como atributo a una entidad B, pero no interesa que B tenga una propia tabla; sino que se pretende que todos los atributos de B estén en la tabla de A. En este caso, B sería Embeddable y A tendría su atributo B como Embedded.

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Embedded & @Embeddable

```
import javax.persistence.*;

@Entity
@Table(name=”empleado”)
public class Empleado {
    @Embedded
    private Direccion direccion;
}
```

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@Embedded & @Embeddable

```
import javax.persistence.*;

@Embeddable
public class Direccion {
    @Column(name=”calle”)
    private String calle;
}
```

Impedance Mismatch - Parte II

Argentina programa 4.0

Mapeo de la Herencia

En el POO existe el mecanismo de herencia, el cual nos permite reutilizar lógica de una clase y/o extender su comportamiento; pero este concepto no existe en el mundo relacional.

¿Qué sucedería si queremos persistir una herencia? ¿Cuántas tablas se generarían en el modelo relacional? ¿Cómo sabe el ORM cuál es la clase específica que tiene que instanciar e hidratar?

Impedance Mismatch - Parte II

Argentina programa 4.0

Mapeo de la Herencia

Existen, al menos, cuatro estrategias de mapeo de herencia:

- ***SINGLE TABLE***: Única tabla.
- ***JOINED***: Una tabla por la superclase y una tabla por cada una de las clases hijas.
- ***TABLE PER CLASS***: Una tabla por cada clase concreta.
- ***MAPPED SUPERCLASS***: Los atributos de la superclase son persistidos en las tablas de las clases hijas.

Impedance Mismatch - Mapeo de Herencia - Mapped Superclass

Argentina programa 4.0

- Los atributos de la superclase son persistidos en las tablas de las clases hijas; y la superclase no es considerada una Entidad.
- Generalmente utilizado cuando la superclase exhibe, únicamente, comportamiento y/o uno o “pocos” atributos en común.
- Uno de los usos más comunes suele ser cuando todas las clases persistentes tienen una PK subrogada y/o un campo “activo” (booleano); pero entre ellas no existe nada más en común.

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@MappedSuperclass

```
import javax.persistence.*;

@Entity
@Table(name=”empleado”)
public class Empleado extends Persistente {
    //atributos
}
```

Mapeo Objeto Relacional - *Annotations*

Argentina programa 4.0

@MappedSuperclass

```
import javax.persistence.*;

@MappedSuperclass
public abstract class Persistente {
    @Id
    @GeneratedValue
    private Integer id;
}
```

Bibliografía

Argentina programa 4.0

https://www.callicoder.com/hibernate-spring-boot-jpa-element-collection-demo/

https://www.baeldung.com/jpa-attribute-converters

https://www.baeldung.com/jpa-embedded-embeddable

https://docs.jboss.org/hibernate/jpa/2.1/api/javax/persistence/MappedSuperclass.html

Gracias!