



Hibernate para *Object/Relational Mapping* (ORM)



Bases de Datos - 38210
Master Oficial en Desarrollo
de Aplicaciones y Servicios
Web

Raúl Bernabeu Mansilla
Armando Suarez Cueto



Contenidos

- Introducción a ORM
- Hibernate
- El contexto de persistencia: Session
- Modelado de relaciones



Introducción a ORM



¿Qué es Object/Relational Mapping (ORM)?

- Mapeo de objeto-relacional
- Modelo de programación que permite “transformar” una base de datos, en una serie de entidades que simplifican al programador, las tareas de acceso a los datos de la base de datos
- Un ORM convierte los datos de los objetos definidos, en un formato correcto, que permite leer/escribir la información de una base de datos (mapeo). Creación de una base de datos virtual, donde los datos que se encuentran en nuestra aplicación, quedan asignados a la base de datos (persistencia)



¿Por qué Object/Relational Mapping (ORM)?

Estrategia: *persistencia* a través de bases de datos relacionales...



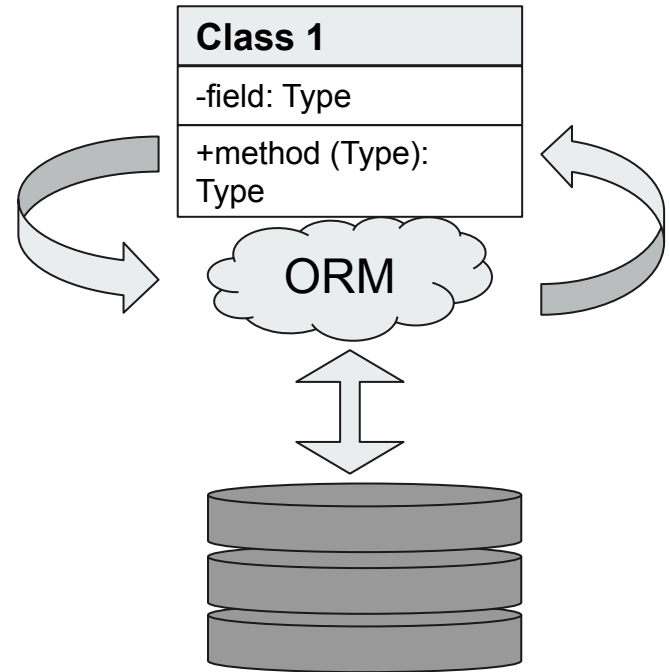
¿Por qué Object/Relational Mapping (ORM)?

Estrategia: *persistencia* a través de bases de datos relacionales...

... es decir, que los datos de nuestra aplicación continúen disponibles tras la ejecución.

¿Qué hace ORM exactamente?

Capa intermedia entre nuestra aplicación y la base de datos:





Ventajas de ORM

- Permite centrarse en el modelo lógico
- Optimiza el número y tipo de operaciones CRUD
- Independencia de la base de datos (MySQL, SQL Server, Oracle...)
- Escalable



MOO vs. MR

El modelo orientado a objetos (MOO) y el modelo relacional (MR) son muy diferentes y es difícil conciliar algunos aspectos:

- Granularidad
- Herencia
- Identidad
- Asociaciones
- Navegación de datos



MOO vs. MR

El modelo orientado a objetos (MOO) y el modelo relacional (MR) son muy diferentes y es difícil conciliar algunos aspectos:

- **Granularidad:** Un objeto puede componerse de otros objetos, mientras que en las tablas de la BDR, todos los datos son columnas de una misma fila
- Herencia
- Identidad
- Asociaciones
- Navegación de datos



MOO vs. MR

El modelo orientado a objetos (MOO) y el modelo relacional (MR) son muy diferentes y es difícil conciliar algunos aspectos:

- Granularidad
- **Herencia:** el propio concepto no existe en el ámbito de las bases de datos relacionales
- Identidad
- Asociaciones
- Navegación de datos



MOO vs. MR

El modelo orientado a objetos (MOO) y el modelo relacional (MR) son muy diferentes y es difícil conciliar algunos aspectos:

- Granularidad
- Herencia
- **Identidad:** discrepancia entre el concepto de identidad en el MR (primary key) y en el MOO (igualdad por valor o por referencia)
- Asociaciones
- Navegación de datos



MOO vs. MR

El modelo orientado a objetos (MOO) y el modelo relacional (MR) son muy diferentes y es difícil conciliar algunos aspectos:

- Granularidad
- Herencia
- Identidad
- **Asociaciones:** relaciones 1:1 y 1:N en MOO, pero también N:M en MR
- Navegación de datos



MOO vs. MR

El modelo orientado a objetos (MOO) y el modelo relacional (MR) son muy diferentes y es difícil conciliar algunos aspectos:

- Granularidad
- Herencia
- Identidad
- Asociaciones
- **Navegación de datos:** En el MOO los datos se navegan a través de la red de objetos, mientras que en el MR se utilizan consultas (más eficientes)

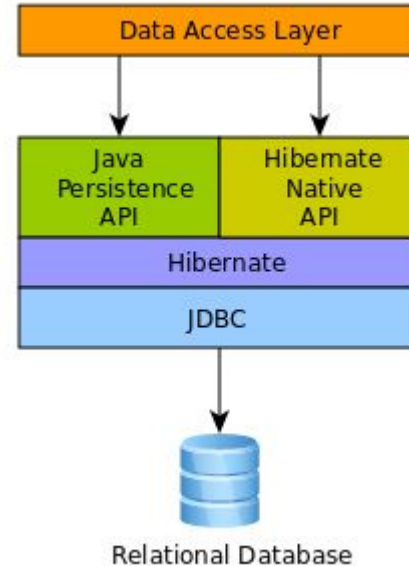


Hibernate

[Enlace al manual](#)

Hibernate

- Solución de código libre para Java
- Implementa dos APIs: JPA y su API nativa
- Utiliza Java Data Base Connectivity para comunicarse con la BDR (pero simplifica su uso)
- Dispone de repositorios en Maven: dependencias fáciles de definir
- Diversas herramientas complementarias





JPA vs Hibernate API

- La Java Persistence API (JPA) es una especificación que se definió después del lanzamiento de Hibernate
- Hibernate implementa JPA, pero añade funcionalidades adicionales que no forman parte de JPA
- Durante el curso veremos elementos de las dos APIs a través de Hibernate



Mapeo POJO - tabla BDR


Hibernate necesita saber dónde guardar cada dato de un objeto (POJO)

Un POJO (Plain Old Java Object) es una clase que...

- ... tiene un constructor por defecto
- ... tiene un atributo identificador (como la clave primaria de la BDR)
- ... los atributos deben ser privados, y tener un *getter* y un *setter*
- ... no es *final**

Cada POJO se corresponde con una entrada de una tabla

Ejemplo: tabla Asignatura

Subject	
	id
	name
	creation_time

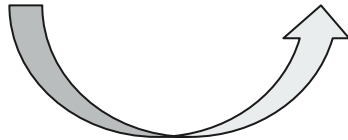
Ejemplo: POJO Asignatura

```
import java.util.Date;
```

```
public class Subject{  
  
    private int id;  
    private String name;  
    private Date creation_date;  
  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
}
```

...

```
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public Date getCreationTime() {  
        return creation_time;  
    }  
    public void setCreationTime(Date creation_time) {  
        this.creation_time= creation_time;  
    }  
}
```





Mapeando objetos y BDR con Hibernate

- Para persistir POJOs en una BDR se debe definir un mapeado
- Hay dos formas de mapear: con anotaciones en Java o con XML
- Las anotaciones en Java son compatibles con JPA
- Veamos unos [ejemplos sencillos de mapeo](#)



El contexto de persistencia: Session

[Enlace al manual](#)



Persistencia con Hibernate

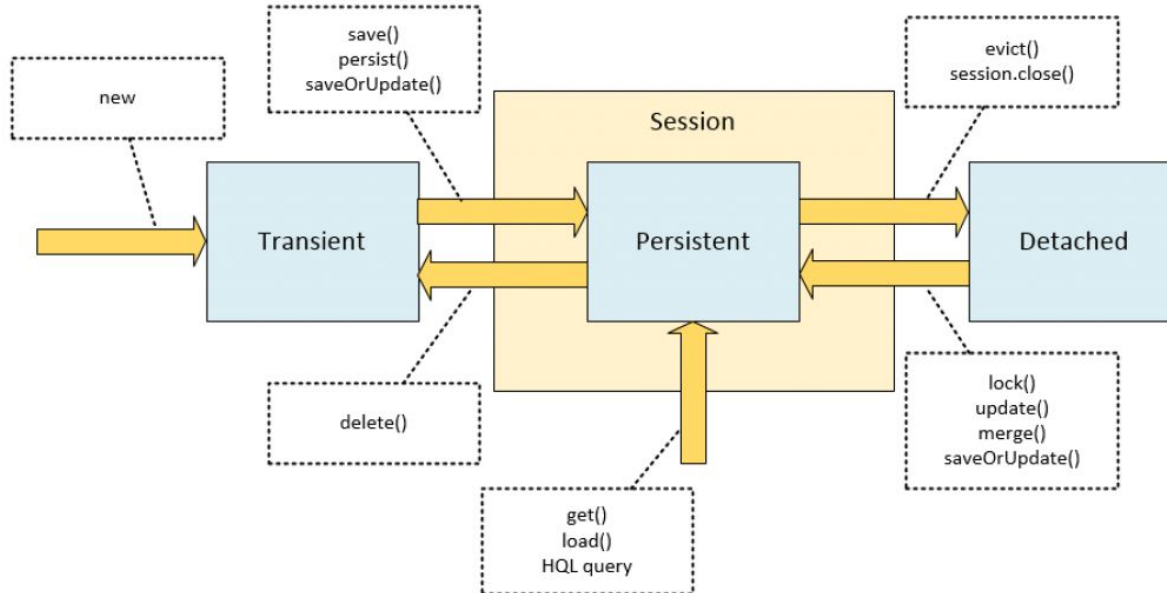
- Hibernate permite gestionar la persistencia de las entidades definidas a través de la interfaz `Session`
- Un objeto `Session` proporcionan un contexto de persistencia y debe ser único para una misma interacción con la base de datos
- Las instancias de `Session` se obtienen a través de `SessionFactory`, que se encarga de crear conexiones con la base de datos de acuerdo con el fichero de configuración
- Suele haber un sólo objeto `SessionFactory` por base de datos y fichero de configuración



Estados de las entidades

- Los objetos de Java correspondientes a una entidad pueden estar en tres estados diferentes respecto de la sesión de Hibernate:
 - **desasociado** (*detached*): se ha guardado en la base de datos pero se ha *desconectado* de ella, de forma que no sabemos si el contenido del objeto se corresponde con el de la base de datos. Por ejemplo al forzar el borrado de la memoria caché.
 - **efímero** (*transient*): aún no se ha guardado en la base de datos. Por ejemplo cuando el objeto se crea (mediante `new`).
 - **persistido** (*persistent*): está guardado en la base de datos y se encuentra actualizado. Hibernate guarda el objeto en la caché interna de primer nivel.


Estados de las entidades






Transient ↔ Persistent

- `save()/persist()`: la entidad es persistida y, por tanto, insertada en la tabla(s) correspondiente(s) de la BD
- `saveOrUpdate()`: independientemente de si el objeto está en estado *transient* o *detached*, se persiste su contenido en la BD
- `delete()`: el contenido de la entidad es borrado de la BD



Persistent → Detached

- `evict()`: el objeto es explícitamente desasociado de la BD y se borra de la caché
- `clear()`: todos los objetos que contienen entidades de la sesión son desasociados y se borran de la caché
- `close()`: se cierra la sesión y desaparece el contexto de persistencia



Detached → Persistent

- `lock()`: el objeto se bloquea a escritura o lectura; generalmente no se utiliza explícitamente sino internamente
- `update/merge()`: los datos que contiene el objeto son actualizados en la(s) tabla(s) correspondiente(s) de la BD
- `saveOrUpdate()`: independientemente de si el objeto está en estado *transient* o *detached*, se persiste su contenido en la BD



Actualizaciones en cascada

- JPA y Hibernate permiten propagar el cambio de estado entre entidades relacionadas con la anotación `@Cascade(name="..")`
 - `name= "MERGE"`
 - `name= "PERSIST"`
 - `name= "REFRESH"`
 - `name= "DETACH"`
 - `name= "REMOVE" / name= "DELETE"`
 - `name= "LOCK"`
 - `name= "SAVE_UPDATE"`
 - `name= "ALL"`



Modelado de relaciones

[Enlace al manual](#)



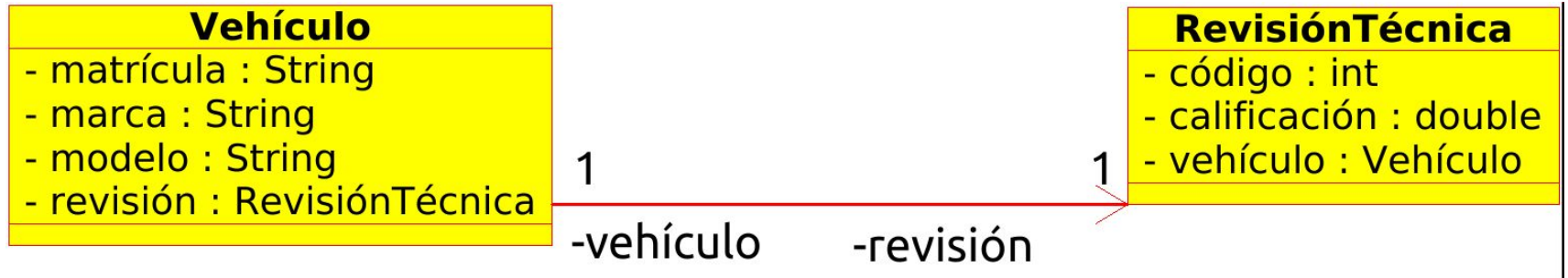
Discrepancia de relaciones entre modelos

Ejemplo: Supongamos un taller que tiene un registro de:

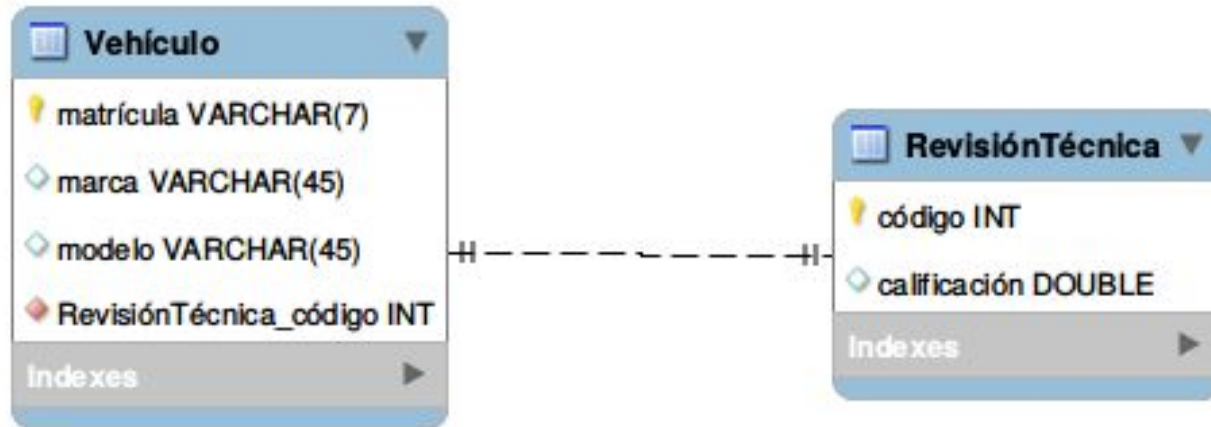
- *Vehículos:* con su nombre, marca, modelo y matrícula
- *Informes de revisión técnica:* con un código y una calificación

Qué tipo de relación se observa? Cómo se implementaría en un modelo OO? Y en uno ER?

Relación 1 a 1 OO



Relación 1 a 1 ER





Discrepancia de relaciones entre modelos (II)

Ejemplo: Supongamos ahora la relación entre:

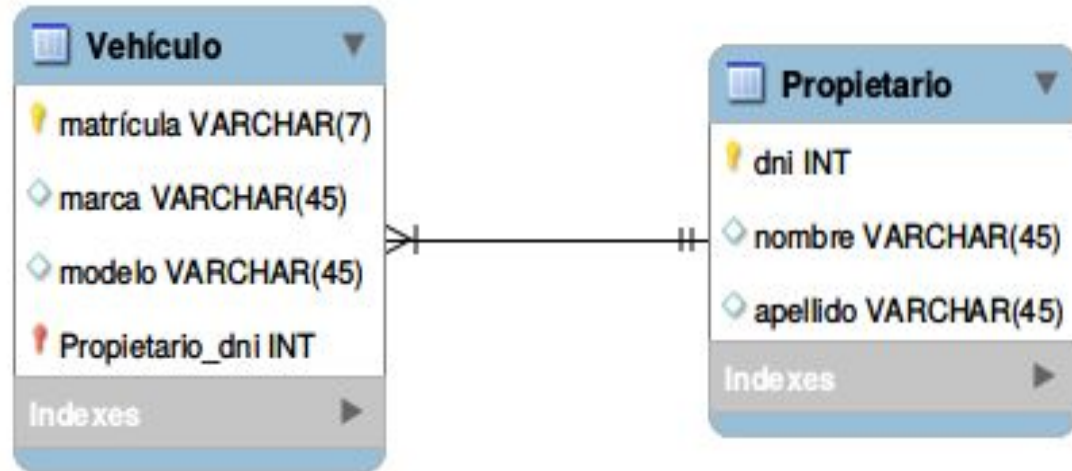
- *Vehículos*: nombre, marca, modelo y matrícula
- *Propietarios*: dni, nombre, apellidos

Qué tipo de relación se observa? Cómo se implementaría en un modelo OO? Y en uno ER?

Relación 1 a N OO



Relación 1 a N ER





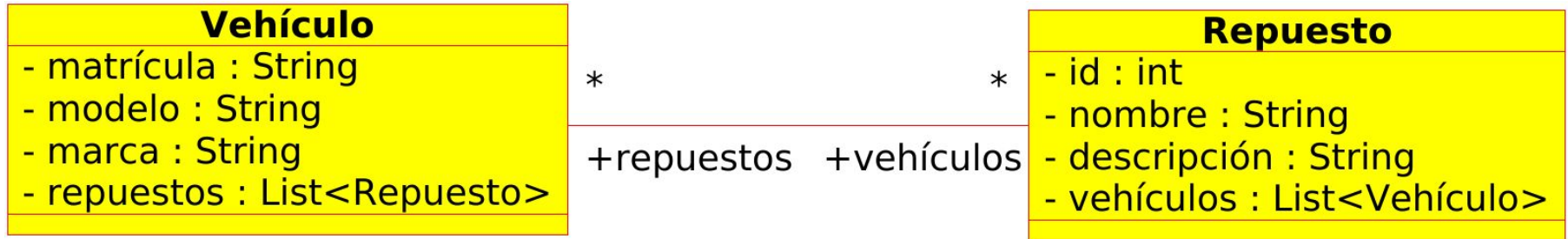
Discrepancia de relaciones entre modelos (II)

Ejemplo: Supongamos ahora la relación entre:

- *Vehículos*: nombre, marca, modelo y matrícula
- *Recambios*: código, nombre y descripción

Qué tipo de relación se observa? Cómo se implementaría en un modelo OO? Y en uno ER?

Relación N a M OO



Relación N a M ER





Implementación de relaciones en Hibernate

Hibernate permite mapear todo tipo de relaciones. [Ejemplos de relaciones en hibernate](#)