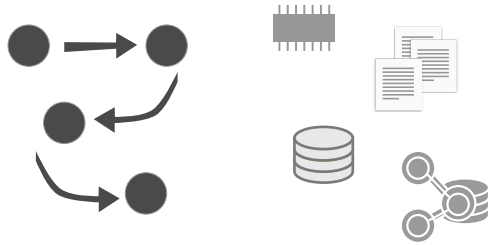
Objetos y Persistencia

Clase invitada

Julián Grigera

Persistencia

Por qué / dónde guardo los objetos?

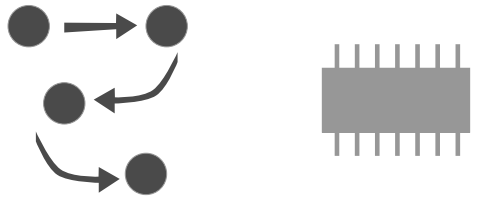


Problemas posibles:

- ¿Dónde puedo guardar objetos?
    - Memoria/Imagen: no permite distribución
    - Archivos: poco soporte y utilidades (ej. no hay índices, transacciones)
    - BBDD (Relacionales): diferencia de impedancia con sistemas OO
    - BDOO: pocas opciones, poca experiencia/recursos

Persistencia

En memoria



En memoria los objetos pueden guardarse.
En ciertos casos hasta tiene sentido - hoy existen BBDD que se usan como cache y no tienen durabilidad.
Hay dos problemas que no resuelven:

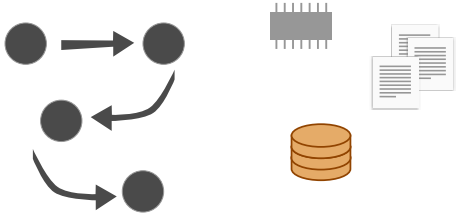
- durabilidad
- distribución, y con ella la escalabilidad

Persistencia
Archivos

Archivos:
- Solución simple y durable
- Se podría incluso distribuir

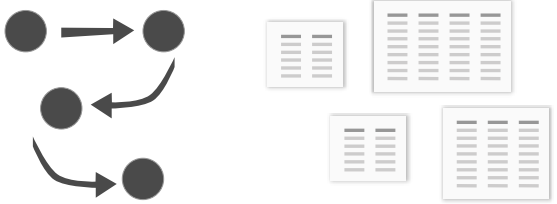
Pero
- No hay integración con OO, hay que desarrollarla
- Poco soporte para performance y distribución (ej. no hay índices, transacciones)

Persistencia
Por qué / dónde guardo los objetos?

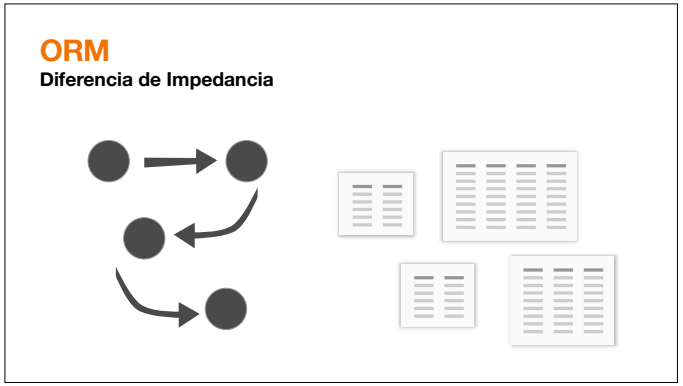


Estas soluciones pueden servir en casos específicos pero no escalan.
Sirven para ver limitaciones y motivos de utilizar otras soluciones de persistencia.
Veamos las más realistas.

Bases de Datos
RDMBS

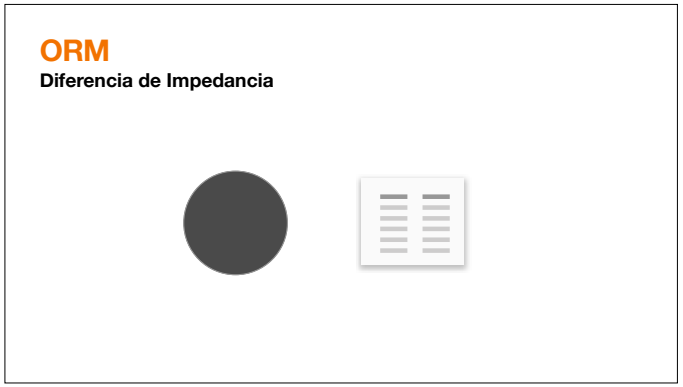


Bases de Datos relacionales, se pueden usar y tienen ventajas:
- SQL es un estándar que permite migrar fácilmente
- Es un paradigma muy conocido y utilizado
- Hay muchos sistemas “legacy"



Diferencia de OO respecto de RDBMS
- OO no contempla normalmente manejo de tx
- Se representan grafos de objetos sin límite aparente

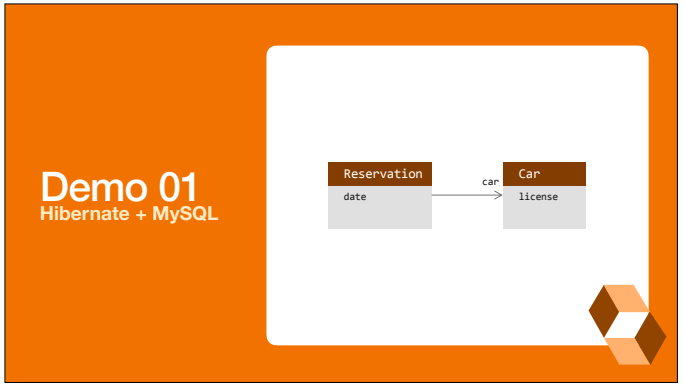
Diferencias de RDBMS respecto de OO
- Cuando hay múltiples relaciones y recursión no es eficiente
- Recuperar datos dispersados incurre en varios JOINs
- No soporta Jerarquías - se traduce en más JOINs



Se podría pensar que estamos volviendo al paradigma procedural -> separando comportamiento de los datos

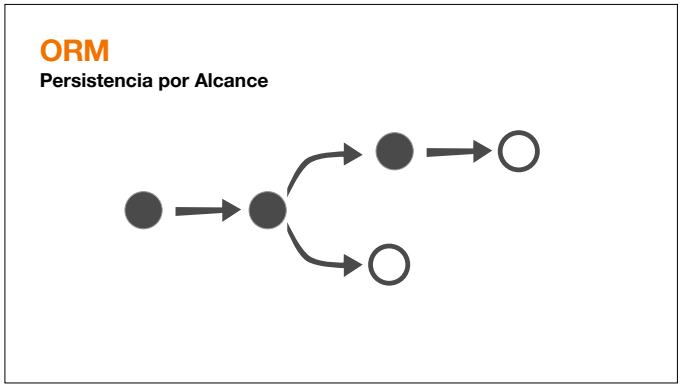
No exactamente:
- Seguimos programando OO
- Buscamos **olvidarnos** de la BD subyacente

En esta demo se ve un ejemplo de uso de Hibernate con MySQL
1. Clase y Mapping
2. Main Save (naïve)
3. Save - persistencia por alcance

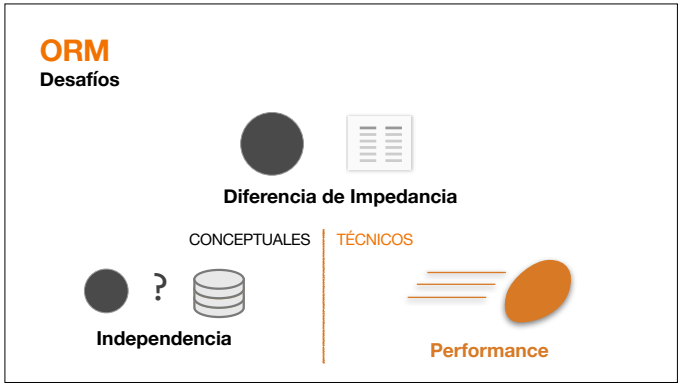


En esta demo se ve un ejemplo de uso de Hibernate con MySQL

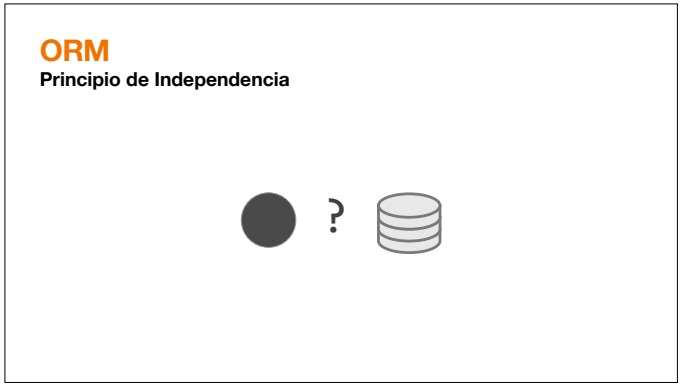
1. Clase y Mapping
2. Main Save (naïve)
3. Save - persistencia por alcance



Reduce el nro de saves
Ideal para independencia
¿Complicaciones posibles?
¿Implementación?

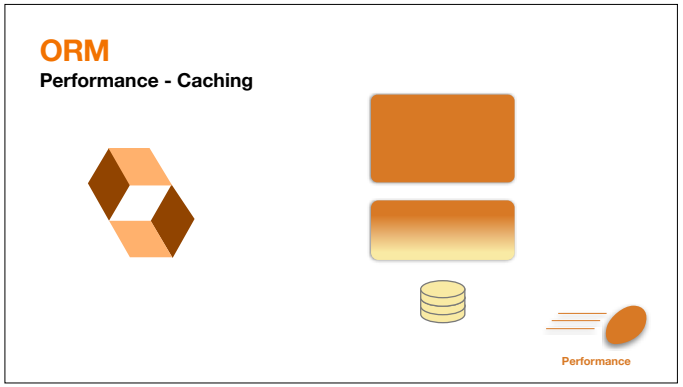


Los ORMs tienen varios desafíos.
La diferencia de impedancia es solo uno de ellos.
Hay otros, tanto **conceptuales** (ej. independencia) como **técnicos** (performance).



Si vemos un código típico (aunque antiguo) de Hibernate encontramos:
- H/SQL embebido
- Annotations
- Tx Explícitas
- “Saves”

Todo esto va contra el principio de independencia

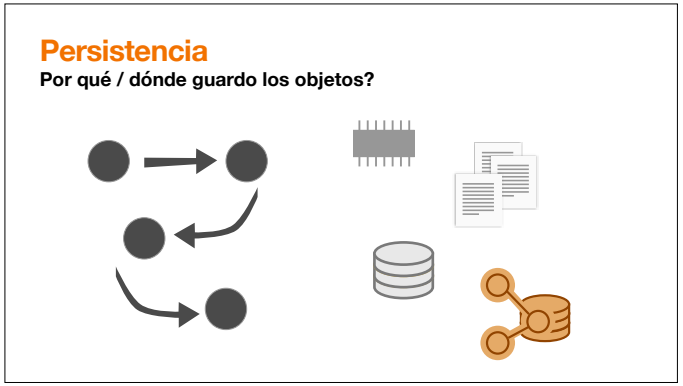


Ejemplo - Hibernate Cache Nivel 1

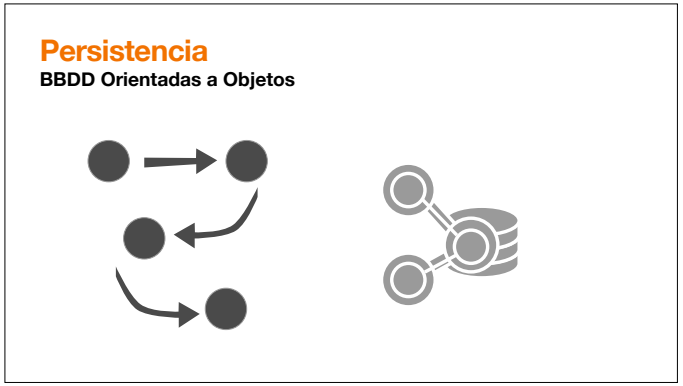
Cuando una entidad se carga o actualiza por primera vez en una sesión, se almacena en la caché a nivel de sesión.

Solicitudes posteriores de la misma entidad en la misma sesión se sirven desde la caché

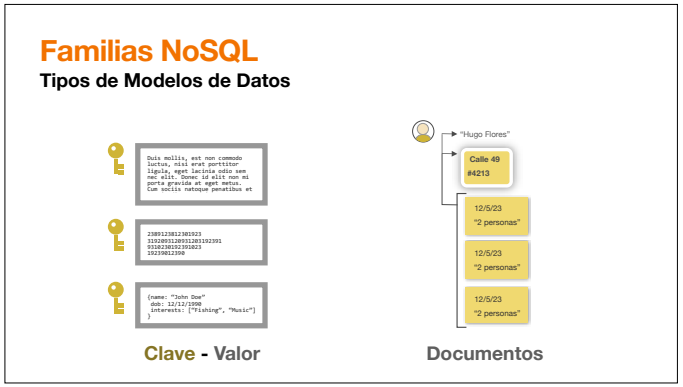
Se minimizan así accesos a la BD



Queda otra opción - BBDD OO



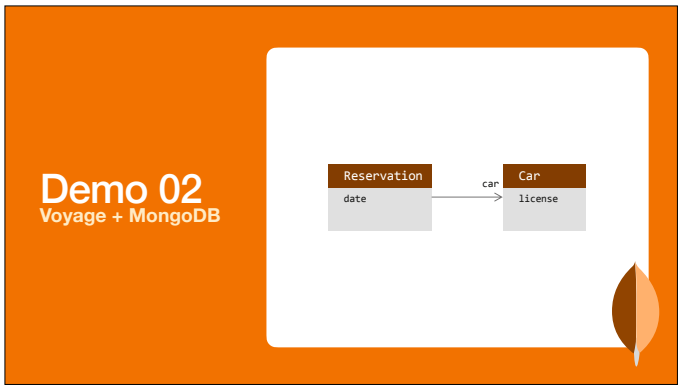
BBDD Orientadas a Objetos
- Resuelven casi todo lo ligado a la diferencia de impedancia en RDBMS
- Ejemplo: Gemstone/S



Clave-Valor (Redis, DynamoDB, Riak)
- estilo tablas hash, el valor suele ser libre - no da mucho lugar a queries
- útiles para guardar información básica (sin relaciones externas) con pocos *updates*
- casos de uso frecuentes: sesiones y cachés

Documentos
- agregaciones con estructura - permite queries
- sin *schema* fijo

En esta demo vemos un mapeador OO -> MongoDB
1. Proyecto
2. Clase (isVoRoot)
3. Persistencia por Alcance



En esta demo vemos un mapeador OO -> MongoDB

1. Proyecto
2. Clase (isVoRoot)
3. Persistencia por Alcance