Bases de Datos 2

Teoría 01

Julián Grigera

Hasta ahora usan BDs aisladas
Vamos a ver aplicabilidad - para que nos sirva
Ergo, vamos a ver qué **opciones tenemos** y **no** tenemos
Criterios para decidir

Presentación

Docentes

Profesores

- Julián Grigera
- Federico Orlando
- Alejandra Lliteras

Jefes de Trabajos Prácticos

- Natalia Correa
- Federico Di Claudio

Información General

Contactos

ACCESIBILIDAD	A.Paola Amadeo	accesibilidad@info.unlp.edu.ar pamadeo@info.unlp.edu.ar
ORIENTACIÓN AL ESTUDIANTE	Ana Ungaro	orientación.estudiantil@info.unlp.edu.ar anaungaro@info.unlp.edu.ar
GÉNERO	Sofía Martin	ddhhygenero@info.unlp.edu.ar smartin@info.unlp.edu.ar
ASESORAMIENTO PEDAGÓGICO	Anahí Almán	direccion.pedagogica@info.unlp.edu.ar

Presentación

Horarios, Aprobación etc

Moodle

https://catedras.linti.unlp.edu.ar/

Registrarse a Bases de Datos 2

Modalidad

- Trabajos en grupo de **3** personas (sin excepción).
- Un **ayudante designado** responsable de la evaluación
- Cada entrega tendrá **una** posibilidad de re-entrega
- Re-entrega disponible **solo** si se hizo entrega
- Cada alumno debe aprobar individualmente un coloquio por entrega
- Explicaciones de práctica en los horarios de práctica al arrancar cada etapa y/o según se requiera

Consultas

Inicio semana 25/03

- Martes 08:00 - 9:30 Aula 8
- Martes 19:00 - 20:30 Sala de PC
- Miércoles 19:00 - 20:30 Aula 4
- Viernes 08:00 - 9:30 Aula 10B

No son obligatorias
Se puede asistir a cualquier horario, pero el ayudante designado estará solamente en su horario

---

Presentación

Horarios, Aprobación etc

- Etapa 1
  24/3 - 16/4 (16/4 entrega - 2/5 reentrega)
  - Definición básica de una aplicación multilayer
  - Uso básico de un ORM y queries
- Etapa 2
  16/4 - 9/5 (9/5 entrega - 23/5 reentrega)
  - Spring Data
- Etapa 3
  12/5 - 6/6 (6/6 entrega - 19/6 reentrega)
  - NoSQL: MongoDB
- Etapa 4
  9/6 - 20/6 (20/6 entrega / 4/7 reentrega)
  - NoSQL: Redis

https://forms.gle/VxZRbpxwg1bJQPam9

Inscripción de grupos hasta el 4/4

Promoción

Condiciones

- Aprobar los trabajos prácticos y sus respectivos coloquios (en cualquiera de sus entregas)
- Aprobar examen de promoción al final de cursada

Bases de Datos 2

Teoría 01

Julián Grigera

Hasta ahora usan BDs aisladas, generalmente relacionales
Vamos a hablar de su aplicabilidad
Ergo, vamos a ver qué **opciones tenemos** y **no** tenemos
Desarrollar criterios para decidir

Bases de Datos

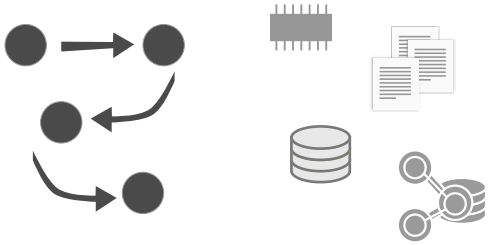
RDMBS

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

Orientación a Objetos:
- Muy presente en la industria

Persistencia

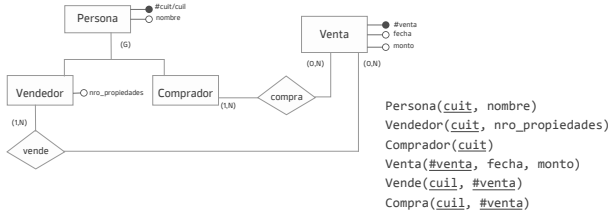
Por qué / dónde guardo los objetos?



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

Modelo Relacional

Cuántas relaciones hay?




```
Persona(cuit, nombre)
Vendedor(cuit, nro_propiedades)
Comprador(cuit)
Venta(#venta, fecha, monto)
Vende(cuil, #venta)
Compra(cuil, #venta)
```

Pensando en el modelo relacional ¿Cuántas relaciones hay en este diagrama?

- 2
- 6
- 5
- **ninguna**

El modelo relacional es muy poderoso, pero también muy simple respecto de OO

ORM

Diferencia de Impedancia

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

ORM

Diferencia de Impedancia

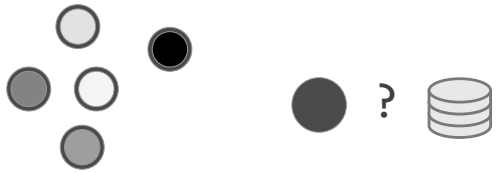
Se podría pensar que estamos volviendo al paradigma procedural -> separando comportamiento de los datos

No exactamente:

- Seguimos programando OO
- Buscamos **olvidarnos** de la BD subyacente

Existen principios para reforzar esto

ORM
Principios



Ortogonalidad

Independencia

Principio de Ortogonalidad: todo objeto debe poder ser persistido más allá de su tipo (más allá de los *transient* objects que es una elección)
Principio de Independencia: el sistema OO debe saber lo menos posible sobre la tecnología de persistencia subyacente

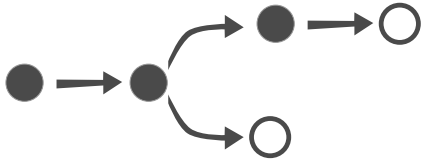
---

ORM
Principio de Independencia

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

ORM
Persistencia por Alcance



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

Bases de Datos 2

Teoría 02

Julián Grigera

Demo 01

Hibernate + MySQL

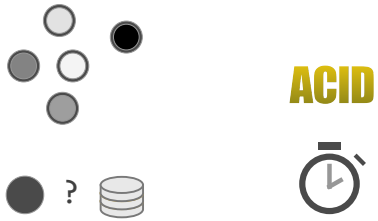
En esta demo se ve un ejemplo de uso de Hibernate con MySQL

1. Proyecto
2. Clase
3. POM
4. Main Save (naïve)
5. CFG (auto create)
6. Main Save (snippet) + Query

Esto se puede volver completo >> necesitamos una guía

Mapeo Objeto-Relacional

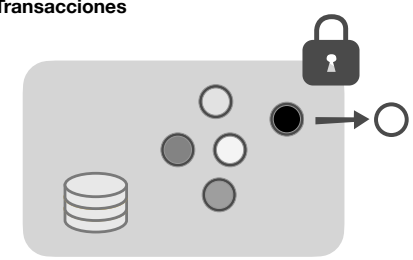
ORM
Principios



Principio de independencia -> la BD tiene que adecuarse al mundo OO
Pero, al incorporar persistencia, el mundo OO tiene que adquirir aspectos de BBDD.
- Para cumplir con las propiedades ACID necesitamos **tx**. ¿Cómo se ven en OO?
- **Performance**: necesitamos, al costo de otras propiedades.
- Problemas con **scope de tx** - Locking Optimista con Versiones

---

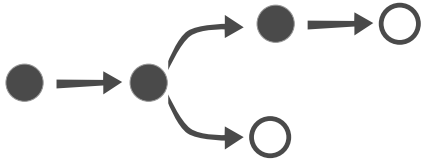
ORM
Transacciones



Para cumplir los principios ACID necesitamos tx. ¿Cómo se ven en OO?
1. Se delimita con **begin/commit**, igual que en SQL
2. No se pueden anidar (al igual que en RDBMS's)
3. Problema de **locking** - no se puede dejar sin acceso a otros por mucho tiempo

---

ORM
Persistencia por Alcance



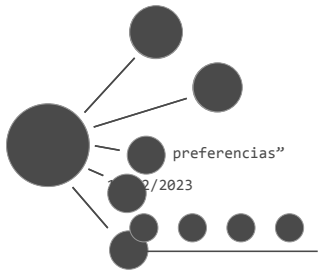
Más aclaraciones sobre la persistencia por alcance:
¿Complicaciones posibles? Por ejemplo, la memoria es un recurso limitado
¿Implementación? Uso del patrón Proxy
- Ahora cada objeto pasa por distintos estados - ¿qué problemas agrega?

Mapeo
Ids

Al mapearlo como clave primara administrada por la BBDD relacional considerar:
- Que sea único
- Que no esté relacionado con el dominio

Estrategias
- Manejada por BD (AUTO, INC | long, UUID)
- Por la Aplicación (ventaja: no requiere esperar al commit)

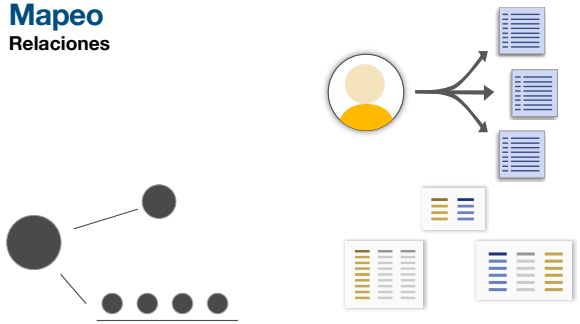
Mapeo
Atributos



En OO, todo es un objeto.
En RDBMS existen muchos tipos de atributos
- Al usar mapping, podemos diferenciar los “atómicos” de las relaciones con otros objetos

---

Mapeo
Relaciones



Relaciones 1 a 1 - simples
Unidireccionales 1 a N: tener en cuenta persistencia por alcance
Bidireccionales 1 a N: Considerar *owning side* (quien guarda la FK).
Ejemplo: Cliente > Reservas. Reserva sería owner porque guarda FK. En el **modelo** esto hace que **cliente.agregarReserva(cena)** NO guarde la relación.
Ejemplos de colecciones: List y Set

Demo 02

Relaciones en Hibernate

1) Naïve

```
Compra
      @OneToMany
      protected List<Item> items = new ArrayList<Item>();
Main
      manager.persist(excursion);
```

2) Con Persistencia por Alcance

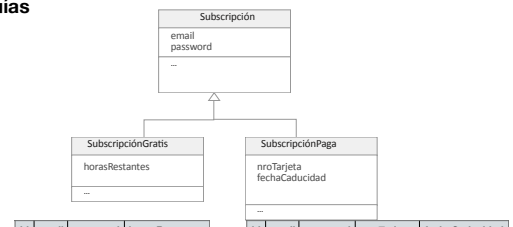
```
Compra
       @OneToMany(cascade = CascadeType.ALL)
       protected List<Item> items = new ArrayList<Item>();
```

3) Optimizado

```
@JoinColumn(name = "compra_id")
```

Mapeo

Jerarquías



id	email	password	horasRestantes
1	x@y.z	*******	15
2	x@y.z	*******	15
3	x@y.z	*******	15

id	email	password	nroTarjeta	fechaCaducidad
1	x@y.z	*******	12341234	12/4
2	x@y.z	*******	4324123	23/5
3	x@y.z	*******	43123442	1/6

Tabla por clase concreta:

- UNION
- Cambios en Clase Padre

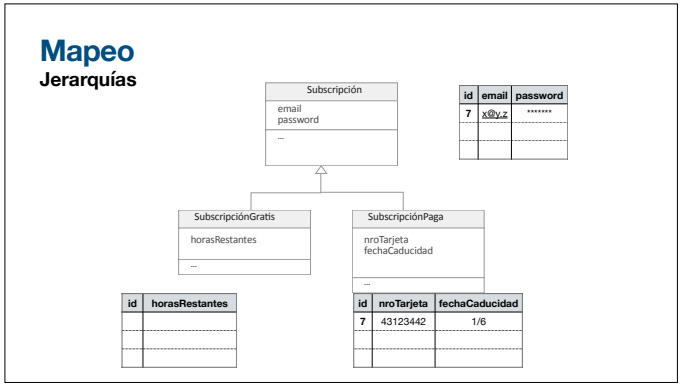


Tabla por clase del modelo
- JOINs

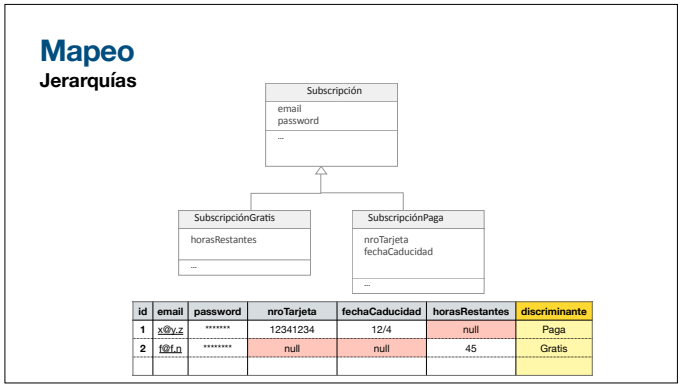


Tabla por jerarquía
- Combinatoria
- Discriminante (ver performance en queries - ix / tipo)

1) Default
```
@Inheritance
```
2) Joined
```
@Inheritance(strategy = InheritanceType.JOINED)
```
2) Tabla por clase
```
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
```