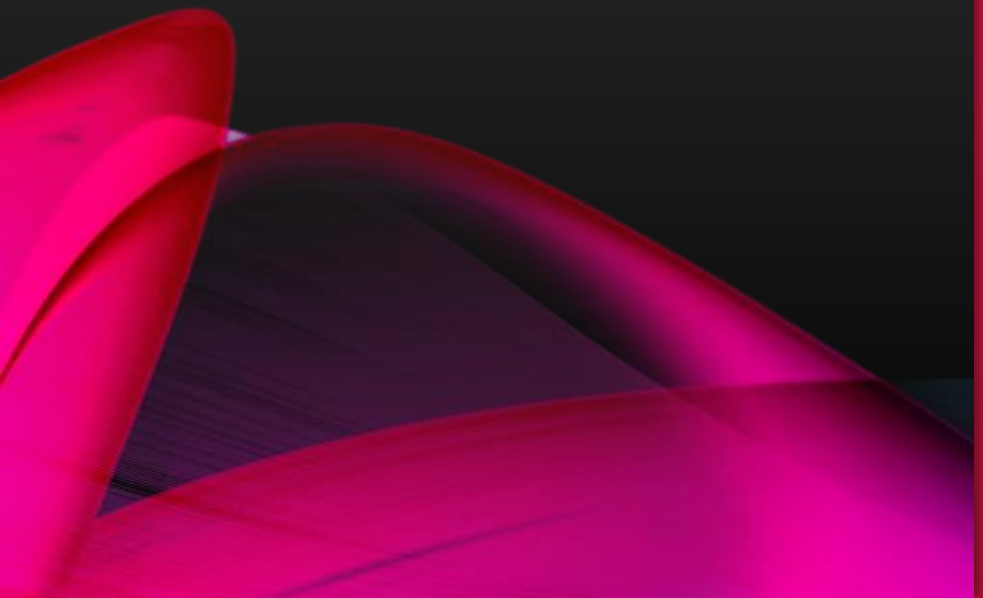
Persistencia en medios relacionales mediante ORM



PERSISTENCIA DE DATOS

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Bases de Datos Relacionales

Retomando desde el punto que desembocó en la anterior explicación…

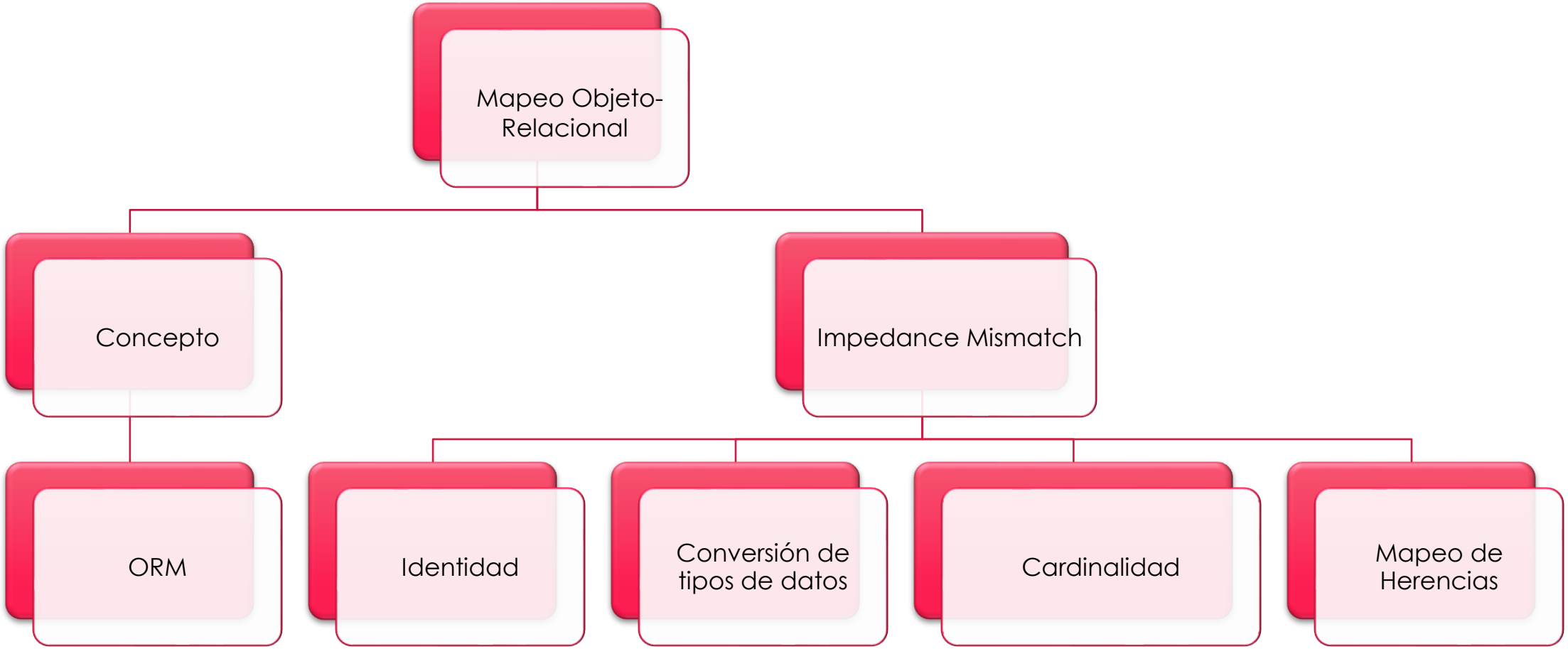
El estado de un Sistema puede ser persistido en una, o varias, Base de Datos Relacional.

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Bases de Datos Relacionales

La persistencia de un Sistema que está pensado y escrito bajo el Paradigma Orientado a Objetos, en una Base de Datos Relacional no es directa.

MAPEO OBJETO - RELACIONAL


Mapeo Objeto-Relacional
Concepto
Impedance Mismatch
ORM
Identidad
Conversión de tipos de datos
Cardinalidad
Mapeo de Herencias

MAPEO OBJETO - RELACIONAL

El Mapeo Objeto-Relacional es un técnica usada para convertir los tipos de datos con los que trabaja un lenguaje orientado a objetos a tipos de dato con los que trabaja un sistema de base de datos relacional.

Mientras que **ORM** es el nombre de la técnica, también se suele conocer como ORMs a los frameworks que la implementan.

MAPEO OBJETO - RELACIONAL

Estas herramientas introducen una capa de abstracción entre la base de datos y el desarrollador, lo que evita que el desarrollador tenga que escribir consultas "*a mano*" para recuperar, insertar, actualizar o eliminar datos en la base.

MAPEO OBJETO - RELACIONAL

Al tratar de encontrar una correspondencia entre estos "mundos", sucede que existen características que son difíciles de imitar.

A este "*desajuste*" se lo conoce como ***Impedance Mismatch***

IMPEDANCE MISMATCH - IDENTIDAD

- *Un objeto cumple con la característica de Unicidad.*
- *Un registro, en una tabla de una base de datos relacional, necesita una identificación unívoca.*

IMPEDANCE MISMATCH - IDENTIDAD

Las claves posibles para una entidad en una BDR son:

- ***Clave natural**: algún campo propio de la entidad que lo identifique de forma unívoca. Por ejemplo, CUIT del cliente, número de teléfono del abonado, etc.*

- ***Clave subrogada**: clave ficticia (al azar, generada automáticamente, etc.).*

IMPEDANCE MISMATCH – CONVERSIÓN DE TIPOS DE DATOS

Los tipos de datos de ambos "mundos" no tienen una correlación directa.

- ***String*** *sin limitaciones de tamaño vs. el VARCHAR/CHAR que requiere definirle longitud.*
- ***Campos numéricos*** *respecto a la precisión de decimales, o el rango de valores máximos y mínimos; incluso cuando el dominio es "un número de 0 a 3000"*
- *Las fechas tienen tipos no siempre compatibles entre sí: LocalDate de Java vs. Date-Datetime-Time-tinyblob del motor.*
- *Booleanos*
- *Enumerados (como los enums de Java)*

IMPEDANCE MISMATCH – MANEJO DE LA CARDINALIDAD

- *En el POO, las relaciones pueden ser unidireccionales y bidireccionales; con navegabilidad bidireccional.*

- *Las relaciones entre las entidades de una base de datos relacional no son bidireccionales (salvo algunos casos).*

IMPEDANCE MISMATCH – 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 – 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 LA HERENCIA

Mapped Superclass

- 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.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Single Table

- Suponiendo que existe una única superclase y N clases hijas, el resultado de mapear la herencia con la estrategia Single Table generará una única tabla en la base de datos.

- Esta única tabla contendrá una columna por cada uno de los atributos persistentes de la superclase + una columna por cada atributo persistentes de cada una de las clases hijas.

- Además, tendrá una columna que actuará de "***campo discriminador***", la cual indicará a qué clase pertenece la instancia/fila en cuestión.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Single Table

Si consideramos una superclase con N atributos persistentes; una clase hija A con M atributos persistentes; y otra clase hija B con P atributos persistentes; entonces:

- Si se persiste una instancia de la clase A (con todos sus atributos seteados, inclusive los pertenecientes a la superclase), quedarán P columnas nulas.

- Si se persiste una instancia de la clase B (con todos sus atributos seteados, inclusive los pertenecientes a la superclase), quedarán M columnas nulas.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Single Table

- Como consecuencia de la utilización de esta estrategia de mapeo de herencia, varias columnas de la tabla resultante podrían ser nulas.

- Pero, en contraparte, se obtiene una buena performance ya que solamente se debe consultar una única tabla.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Joined

- Suponiendo que existe una única superclase y N clases hijas, el resultado de mapear la herencia con la estrategia Joined generará, en la base de datos, una tabla por la superclase y N tablas más, una por cada clase hija.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Joined

- La tabla que representa a la superclase contendrá, únicamente, tantas columnas como atributos persistentes existan en dicha clase. Además, puede contener opcionalmente el campo discriminador.

- Cada una de las tablas que representan a las clases hijas contendrán tantas columnas como atributos persistentes existan en dichas clases.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Joined

- Cada una de las PKs de las tablas que representan a las clases hijas, a su vez serán una FK a la tabla que representa a la superclase.

- Para recuperar un objeto con todos sus atributos (los propios + los que están en la superclase), el ORM debe *joinear* las tablas.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Table per Class

- Suponiendo que existe una única superclase y N clases hijas concretas, el resultado de mapear la herencia con la estrategia Table per Class generará, en la base de datos, N tablas: una por cada clase hija.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

Table per Class

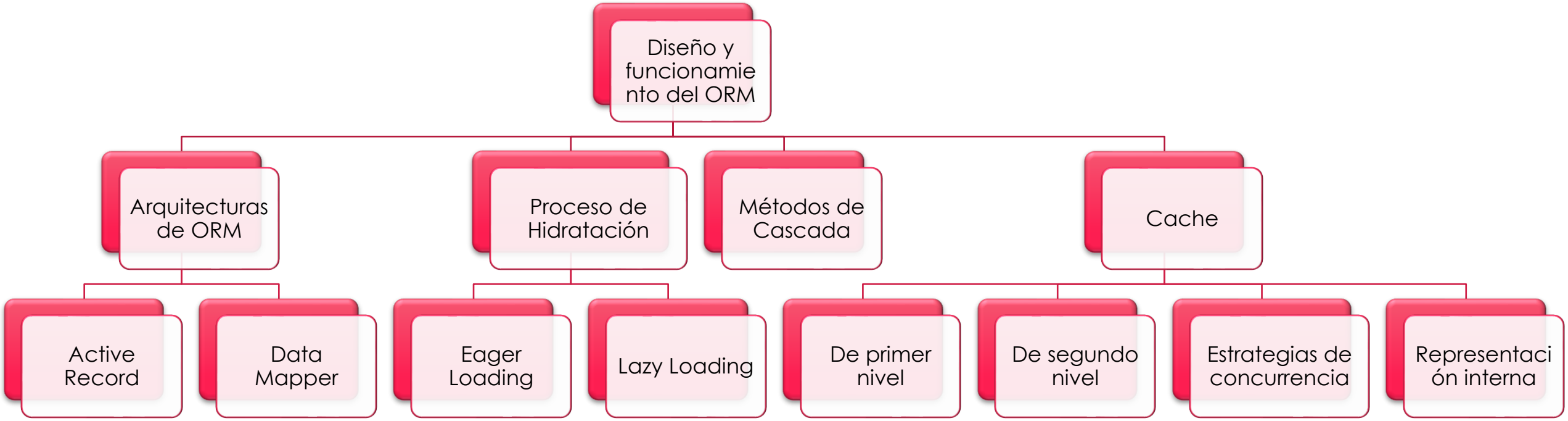
- Todos los atributos persistentes de la superclase serán persistidos en cada una de las tablas que mapean contra las clases hijas.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

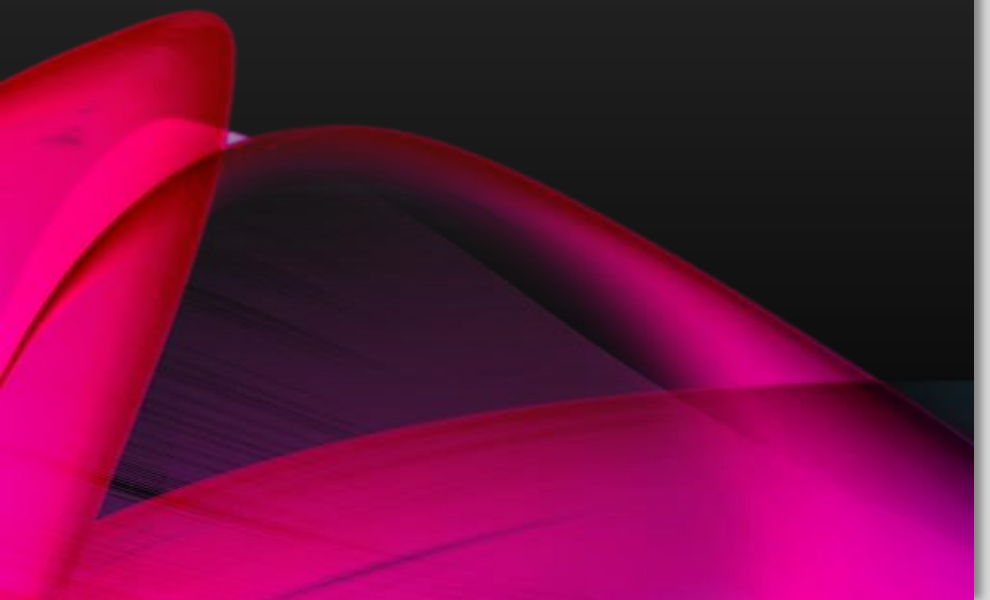
Table per Class

- Para recuperar polimórficamente todos los objetos, el ORM debe realizar *unions* entre todas las tablas que mapean contra las clases hijas.

DISEÑO Y FUNCIONAMIENTO DEL ORM


Diseño y funcionamiento del ORM
Arquitecturas de ORM
Proceso de Hidratación
Métodos de Cascada
Cache
Active Record
Data Mapper
Eager Loading
Lazy Loading
De primer nivel
De segundo nivel
Estrategias de concurrencia
Representación interna

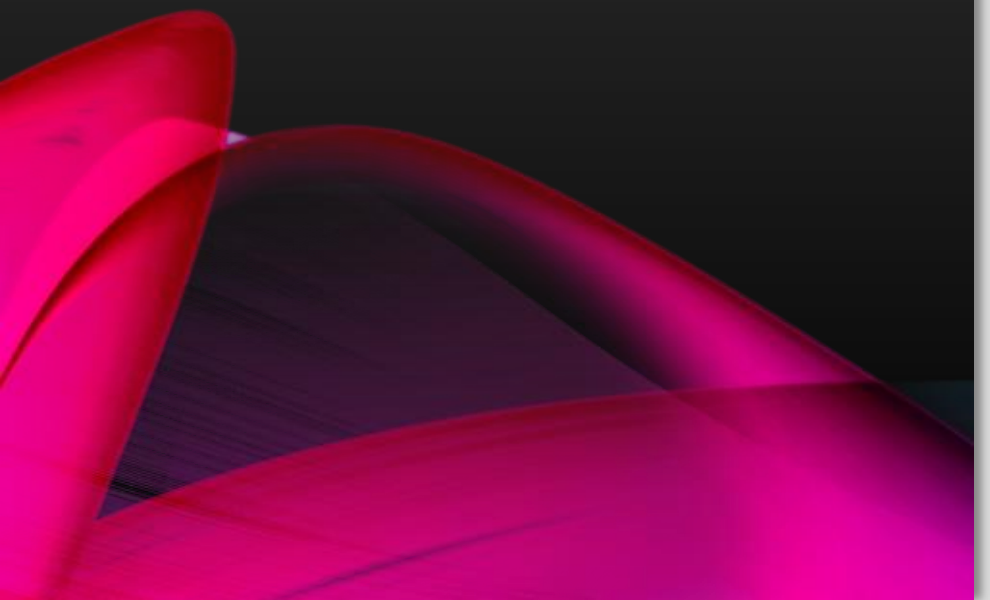
PROCESO DE HIDRATACIÓN



¿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



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

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

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.

HIDRATACIÓN LAZY VS EAGER

- ***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.

ARQUITECTURAS DE ORM

Los ORM se clasifican en dos tipos de arquitecturas…

ARQUITECTURAS DE ORM

Active Record

Data Mapper

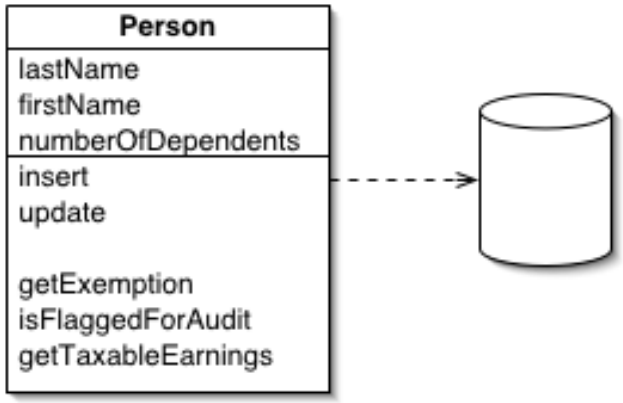
ARQUITECTURAS DE ORM

Active record

En esta arquitectura los ORMs decoran las clases de entidades de dominio agregándoles funcionalidades/responsabilidades. Estas responsabilidades están relacionadas a las acciones de Alta (save), Baja (delete) y Modificación (update) de la entidad, así como también las funcionalidades para permitir la búsqueda de dicha entidad (find, findBy, findAll, entre otras).

ARQUITECTURAS DE ORM

Active record



ARQUITECTURAS DE ORM

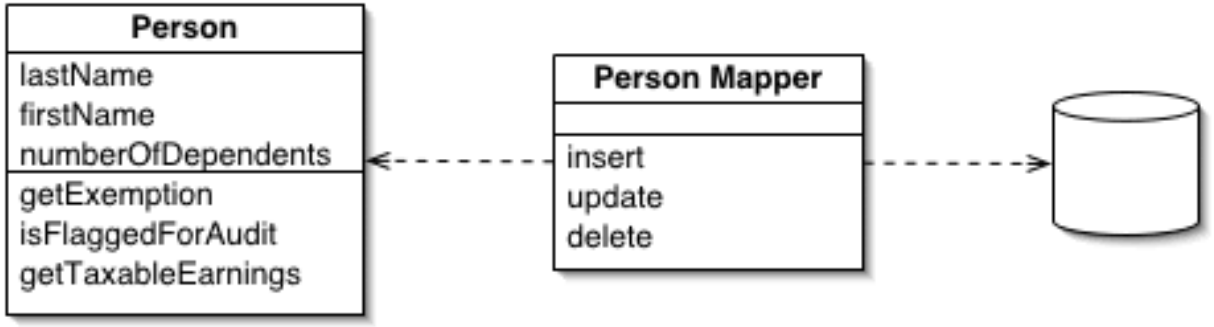
Data mapper

- En esta arquitectura los ORMs agregan un nuevo componente intermedio entre la Base de Datos y las entidades de dominio.

- Este componente se llama "*entity manager*" y lleva la responsabilidad de buscar, agregar, modificar y eliminar (en otras operaciones) objetos de las entidades persistentes.

ARQUITECTURAS DE ORM

Data mapper



MÉTODOS DE CASCADA

"Las relaciones entre entidades, a menudo, dependen de la existencia de otra entidad. Por ejemplo, la relación Persona – Dirección, suponiendo que una dirección le pertenece siempre a una persona. Sin la Persona , la entidad Dirección no tiene ningún significado propio. Cuando eliminamos la entidad Persona , nuestra entidad Dirección también debería eliminarse."

MÉTODOS DE CASCADA

"La cascada implica que, cuando realicemos alguna acción en la entidad objetivo, la misma acción se aplicará a la entidad asociada."

MÉTODOS DE CASCADA

Existen varios tipos de Cascadas (no todos disponibles en todos los ORMs):

- *ALL*
- *PERSIST*
- *MERGE*
- *REMOVE*
- *REFRESH*
- *DETACH*

MÉTODOS DE CASCADA

Tipo de cascada "ALL"

- Este tipo de cascada propagará todas las operaciones desde la entidad principal a la entidad secundaria.

MÉTODOS DE CASCADA

Tipo de cascada "PERSIST"

- Este tipo de cascada propagará la operación de persistencia de una entidad principal a una entidad secundaria.

- Cuando guardemos la entidad principal, la entidad secundaria también se guardará.

MÉTODOS DE CASCADA

Tipo de cascada "MERGE"

- Este tipo de cascada propagará la operación de actualización de una entidad principal a una entidad secundaria.

- Cuando actualicemos la entidad principal, la entidad secundaria también se actualizará.

MÉTODOS DE CASCADA

Tipo de cascada "REMOVE"

- Este tipo de cascada propagará la operación de eliminación de una entidad principal a una entidad secundaria.

- Cuando eliminemos la entidad principal, la entidad secundaria también se eliminará.

MÉTODOS DE CASCADA

Tipo de cascada "REFRESH"

- La operación "refresh" refresca todos los atributos de un objeto, es decir, "re-popula" la entidad.
- Este tipo de cascada propagará la operación "refresh" de una entidad principal a una entidad secundaria.
- Cuando la entidad principal sea refrescada, la entidad secundaria también lo será.

MÉTODOS DE CASCADA

Tipo de cascada "DETACH"

- La operación "detach" quita la entidad del contexto persistente.
- Este tipo de cascada propagará la operación "detach" de una entidad principal a una entidad secundaria.
- Cuando la entidad principal sea quitada del contexto persistente, la entidad secundaria también lo será.

CACHE

- Los ORM tienen la capacidad de almacenar en cache, de forma transparente, los datos recuperados desde el medio persistente.

- Esto ayuda a reducir los costos de acceso al medio persistente, en cuestiones de tiempo y recursos, para aquellos datos que son consultados con una alta frecuencia.

CACHE

- Existen dos tipos de caches en los ORM:
  - Cache de Primer Nivel
  - Cache de Segundo Nivel (no todos los ORM cuentan con él)

CACHE DE PRIMER NIVEL

- La cache de primer nivel tiene un alcance de sesión que garantiza que cada instancia de una entidad se cargue solamente una vez en el contexto persistente.

- Una vez que se cierra la sesión, la cache de primer nivel termina.

- Esta cache permite que las sesiones concurrentes trabajen con instancias de forma aislada.

CACHE DE SEGUNDO NIVEL

- La cache de segundo nivel tiene un alcance de SessionFactory (en Hibernate), lo que significa que es compartida por todas las sesiones creadas con la misma factory.

CACHE DE SEGUNDO NIVEL

Cuando se busca una instancia de una entidad por su id y la cache de segundo nivel está habilitada, sucede alguno de los siguientes escenarios:

- Si la instancia ya está presente en la cache de primer nivel, es devuelta desde allí.

- Si la instancia no está presente en la cache de primer nivel pero el estado de la misma está almacenado en la cache de segundo nivel, entonces se obtienen los datos desde allí, se hidrata el objeto y se devuelve la instancia.

CACHE DE SEGUNDO NIVEL

- Si no se encuentra en ninguno de los anteriores casos, entonces se realiza el proceso de búsqueda e hidratación común y corriente.

CACHE – ESTRATEGIAS DE CONCURRENCIA

Existen varias estrategias de concurrencia de cache de segundo nivel:

- ***READ_ONLY***: se debería utilizar con entidades que nunca cambian. Muy adecuado

- ***NONSTRICT_READ_WRITE***: la cache se actualiza después de que se haya confirmado una transacción que cambió los datos afectados. Existe una pequeña ventana de tiempo en la que se pueden obtener datos obsoletos desde la cache. Esta estrategia es adecuada si se puede tolerar una mínima inconsistencia eventual.

CACHE – ESTRATEGIAS DE CONCURRENCIA

- ***READ_WRITE***: esta estrategia garantiza una fuerte consistencia que se logra mediante la utilización de bloqueos "suaves". Cuando se actualiza una entidad en la cache, también se almacena un bloqueo suave para esa entidad en la cache, que se libera después de que se confirma la transacción. Todas las transacciones concurrentes que acceden a registros con bloqueo suave, obtendrán los datos correspondientes directamente desde el medio persistente.

CACHE – REPRESENTACIÓN INTERNA

- Las entidades no se almacenan en la cache como objetos, sino que solamente su guarda su estado (valores de los atributos).
- Los atributos transitorios no se guardan.
- Las colecciones no se guardan (a menos que se haya explicitado lo contrario)
- En las relaciones xToOne solamente se almacena el id de la entidad externa.

PATRÓN REPOSITORIO

PATRÓN REPOSITORIO

- *No es un patrón de Diseño*
- *Es un patrón arquitectónico*
- *Nos ayuda a estructurar el aplicativo para lograr una buena separación de concerns*

PATRÓN REPOSITORIO

- *El patrón repositorio se apoya sobre el estilo en capas para estructurar el aplicativo.*

- *Propone crear una capa de persistencia para que la misma se encargue del acceso a los datos.*

PATRÓN REPOSITORIO

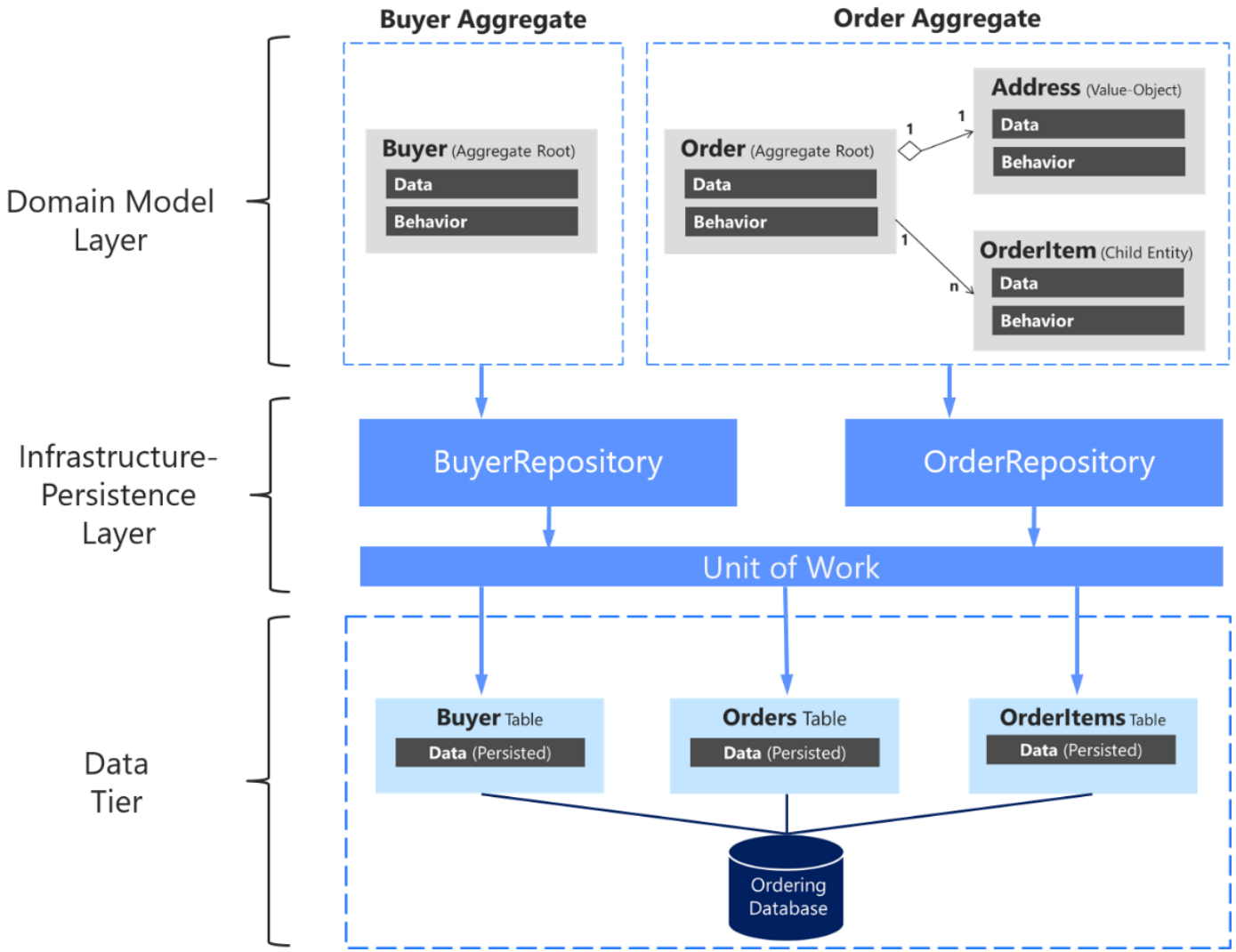
- *Propone que en la capa de persistencia existan objetos "Repository" .*

- *Cada Repositorio debería poder:*

  - *agregar(unObjeto)*
  - *modificar(unObjeto)*
  - *eliminar(unObjeto)*
  - *buscar*
  - *buscarTodos*

PATRÓN REPOSITORIO

Gráficamente, se podría representar de la siguiente forma…


Buyer Aggregate
Order Aggregate
Domain Model Layer
Buyer (Aggregate Root)
Data
Behavior
Order (Aggregate Root)
Data
Behavior
1
1
Address (Value-Object)
Data
Behavior
1
n
OrderItem (Child Entity)
Data
Behavior
Infrastructure-Persistence Layer
BuyerRepository
OrderRepository
Unit of Work
Data Tier
Buyer Table
Data (Persisted)
Orders Table
Data (Persisted)
OrderItems Table
Data (Persisted)
Ordering Database

PATRÓN REPOSITORIO

- *El patrón repositorio puede implementarse haciendo uso de los objetos DAOs.*
- *DAO es la abreviatura de Data Access Object*
- *Los DAO son los responsables reales de acceder a los datos. Pueden acceder a una base de datos relacional, a una base de datos no relacional, a la memoria, a archivos, etc.*

PATRÓN REPOSITORIO

- *El patrón repositorio puede combinarse con el patrón de diseño Strategy para utilizar los DAOs.*
- *Cada repositorio podría delegar su responsabilidad en un DAO concreto.*
- *Para clientes de los repositorios, es indistinto el DAO concreto que se utiliza. En otras palabras, los clientes de los repositorios no deberían enterarse cuál es el medio persistente.*

BIBLIOGRAFÍA

- *Design the infrastructure persistence layer – Microsoft – En línea [Artículo]*
- *Diseño de Datos: Modelo Relacional – Zaffaroni Juan – En línea [Documento]*
- *Diseño de Datos: Mapeo Objetos Relacional – Dodino Fernando, Bulgarelli Franco – En línea [Documento]*
- *Guía de persistencia de JPA – Bulgarelli Franco, Prieto Gastón – En línea [Documento]*
- *Hibernate Inheritance Mapping – Baeldung – En línea [Sitio Web]*
- *Hibernate Second Level Cache – Baeldung – En línea – [Sitio Web]*
- *Introducción a las bases NoSQL – Dodino Fernando, Tesone Pablo, Bulgarelli Franco – En línea [Documento]*
- *Introducción a los Sistemas de Base de Datos, C.J Date, Edit: Pearson, 2001.*
- *Overview of JPA/Hibernate Cascade Types – Baeldung - En línea [Sitio Web]*