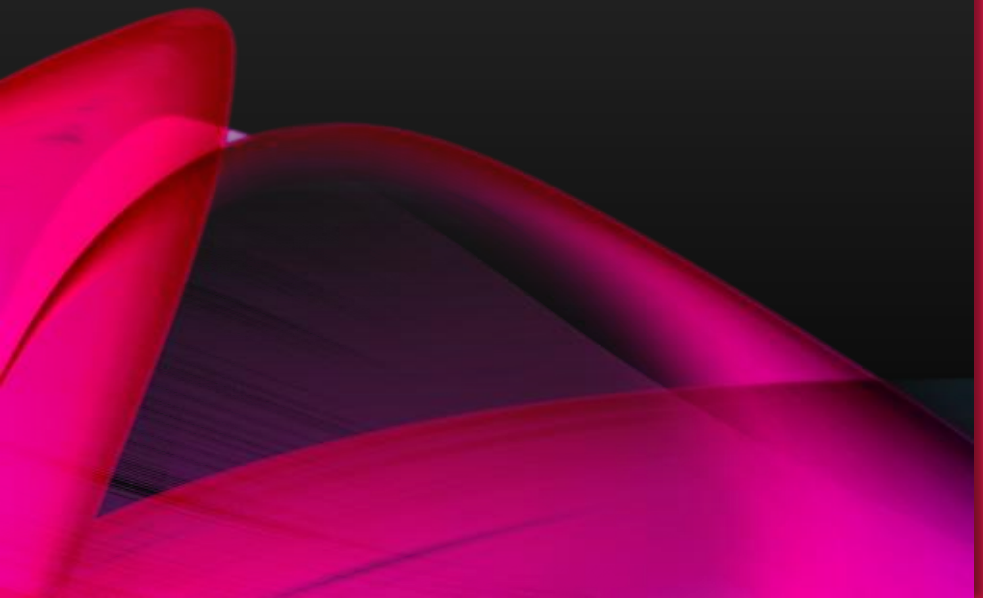
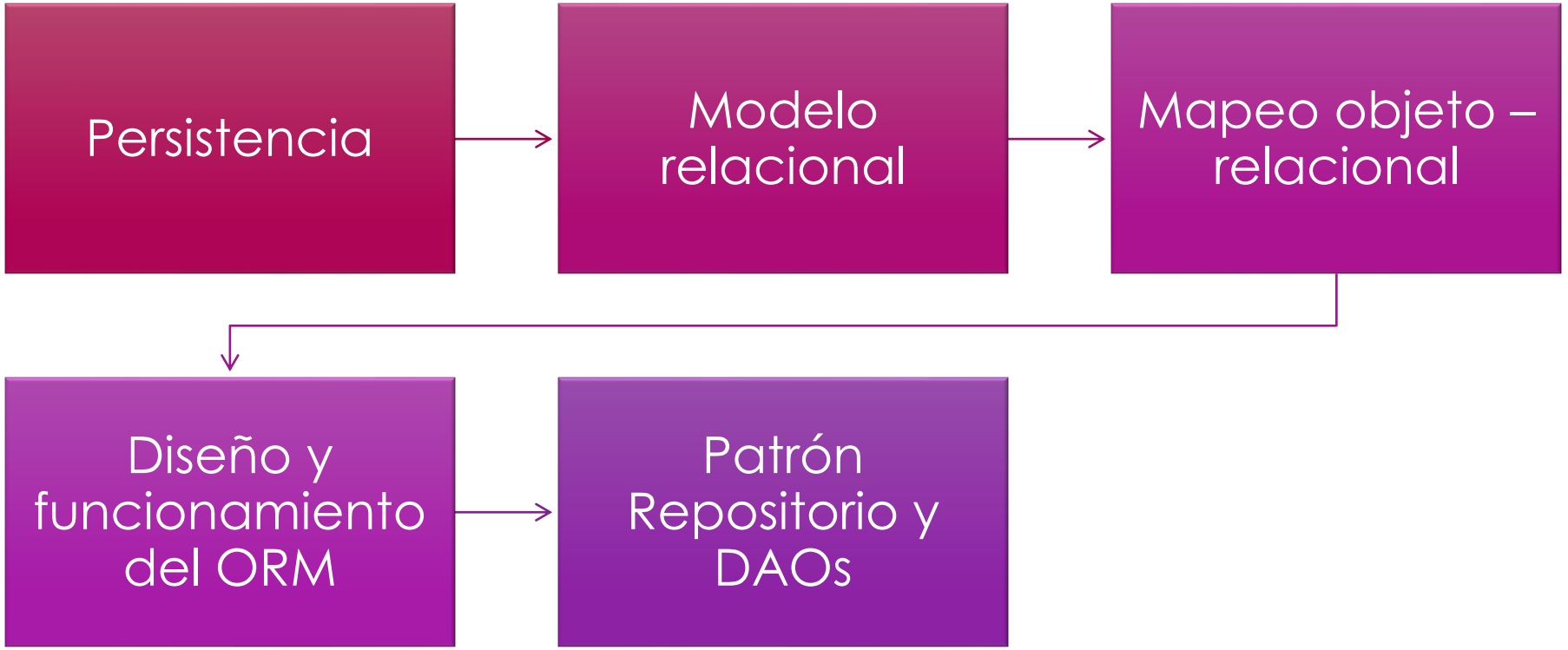
Persistencia en medios relacionales mediante ORM



PERSISTENCIA DE DATOS

TEMARIO GENERAL




Persistencia
Concepto
Estrategias de Persistencia
Persistencia en Memoria
Persistencia en Archivos
Prevalencia
Ortogonal
Persistencia en Base de datos
Bases de datos de Objetos
Bases de datos No-SQL
Bases de datos Relacionales

PERSISTENCIA

En términos informáticos, la persistencia se refiere a que el estado de un sistema sobrevive más allá del proceso que lo creó; o dicho en otras palabras, sobrevive más allá de una única ejecución.

Esto se logra, en la práctica, almacenando dicho estado en algún (o algunos) medio persistente, tal como una base de datos o archivos.

ESTRATEGIAS DE PERSISTENCIA

Existen varias estrategias de persistencia de datos un aplicativo:

- *Persistencia en Memoria*
- *Persistencia en Archivos*
- *Prevalencia*
- *Ortogonal*
- *Persistencia en Base de datos*
  - *Bases de datos de Objetos*
  - *Bases de datos No-SQL*
  - *Bases de datos Relacionales*

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Memoria

- *"La persistencia en memoria es la capacidad de un dato u objeto de seguir existiendo luego de ser utilizado en distintas operaciones."*

- Si bien la memoria RAM es un medio volátil, lo que se contradice con el concepto de persistencia puro; existen, también, memorias persistentes.

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Memoria

- *Este tipo de persistencia es utilizado mayormente para realizar tests en los sistemas.*
- *También existen implementaciones de Bases de Datos que persisten sus datos en memoria, tal como SQLite.*

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Archivos

La persistencia en Archivos involucra guardar el estado de un Sistema en uno o varios Archivos para que luego éste pueda ser recuperado, y el Sistema pueda continuar funcionando/ejecutando desde el mismo punto, es decir, como "si nada hubiera pasado".

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Archivos

Existen varios tipos de Archivos/Formatos de intercambio que son utilizados para persistir o transmitir datos, tales como:

- *XML*
- *CSV*
- *Ancho Fijo*
- *JSON*
- *Otros…*

ESTRATEGIAS DE PERSISTENCIA

Prevalencia

Es una técnica que almacena el Estado de un Sistema en la Memoria principal, pero que regularmente genera "snapshots" a disco para evitar la pérdida completa de los datos.

Las implementaciones de esta técnica tienen la capacidad de manejar transacciones.

ESTRATEGIAS DE PERSISTENCIA

Prevalencia

Una de las principales ventajas es que no existen transformaciones de datos, ya que éstos son persistidos en el formato que el aplicativo utiliza o "entiende". Como consecuencia se obtiene un alto grado de transparencia y una mejora notable en la performance.

ESTRATEGIAS DE PERSISTENCIA

Prevalencia

Como desventaja se destaca la falta de interoperabilidad de los datos, además de que debe considerarse una alta capacidad de Memoria ya que la misma debe poder alojar al aplicativo completo.

ESTRATEGIAS DE PERSISTENCIA

Prevalencia

Prevayler es una posible implementación para Java.

ESTRATEGIAS DE PERSISTENCIA

Ortogonal

La persistencia Ortogonal refiere a que la Persistencia del Estado de un Sistema se implementa como una propiedad intrínseca de su entorno de ejecución.

Por este motivo, no se requieren acciones específicas para poder guardar o recuperar datos, ya que esto ocurre de forma "transparente".

ESTRATEGIAS DE PERSISTENCIA

Ortogonal

Este tipo de persistencia es adoptado por los Sistemas Operativos, permitiendo funcionalidades como la Hibernación; y en Sistemas de Virtualización de Plataformas como VMware o VirtualBox.

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Base de Datos

- *En este caso, el Estado de un Sistema es persistido en una (o varias) Base de Datos.*
- *Los datos persistentes del aplicativo, dependiendo el tipo de Base de Datos, necesitan una transformación antes de ser persistidos.*

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Base de Datos

- *La interoperabilidad es un atributo de calidad que se maximiza en este caso, ya que los datos se persisten de forma independiente a los Sistemas que los manipulan.*

- *Las bases de datos otorgan mecanismos para recuperarse en caso de fallos, además de brindar reglas para resguardar la integridad de los datos.*

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Base de Datos Orientadas a Objetos

- *Cada una de las Bases de Datos Orientadas a Objetos está atada a un lenguaje de programación en particular, ya que en cada lenguaje los objetos tiene una representación interna diferente.*

- *Como ventaja se destaca la no transformación de los datos, cuya característica proporciona una mayor performance.*

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Base de Datos Orientadas a Objetos

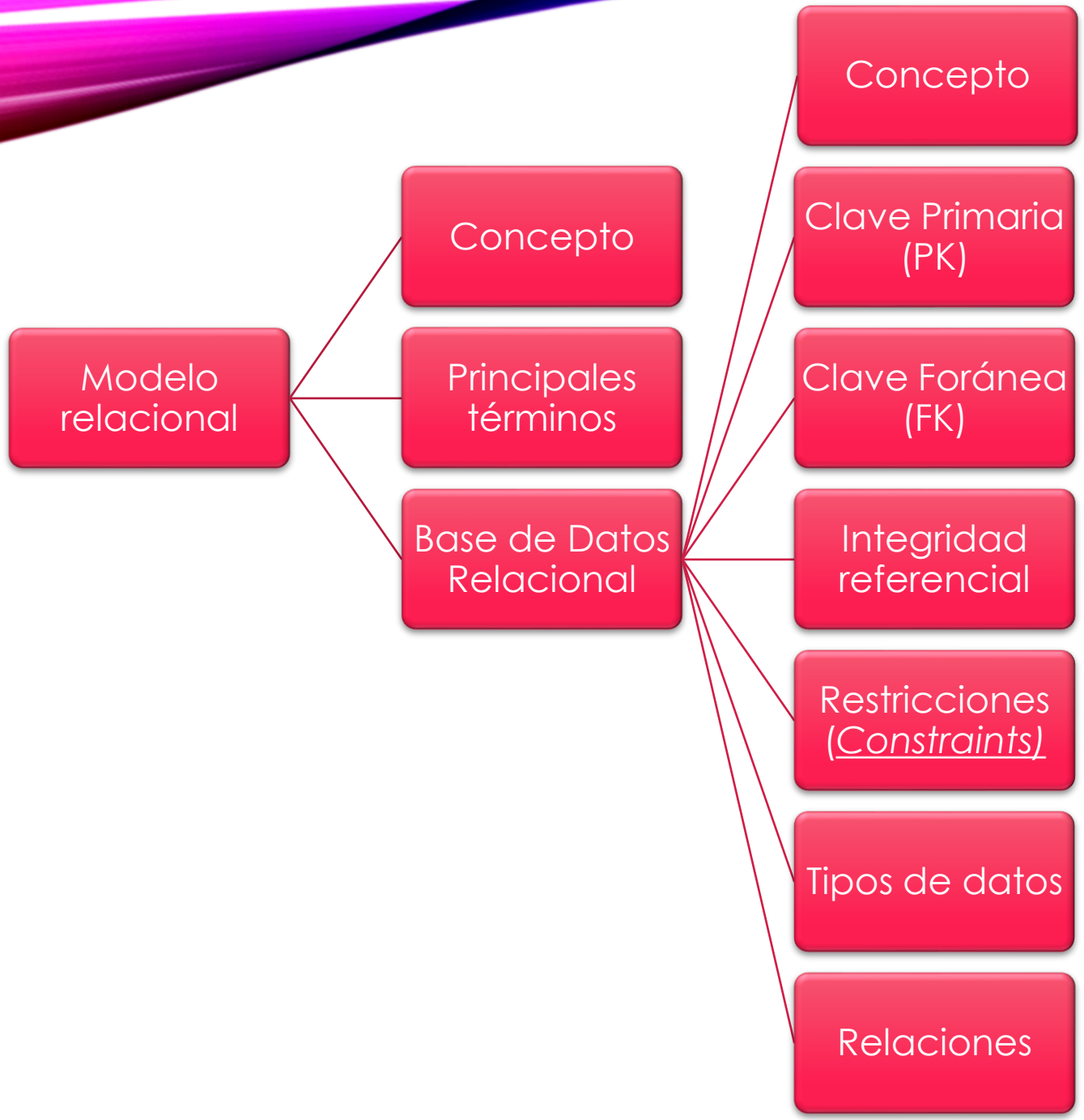
- *La principal desventaja radica en la interoperabilidad de los datos.*

ESTRATEGIAS DE PERSISTENCIA

Persistencia en Base de Datos Relacionales y No-SQL

Antes de entrar en esta sección, es necesario conocer Teoría sobre el Modelo Relacional.

MODELO RELACIONAL


Modelo relacional
Concepto
Principales términos
Base de Datos Relacional
Concepto
Clave Primaria (PK)
Clave Foránea (FK)
Integridad referencial
Restricciones (Constraints)
Tipos de datos
Relaciones

MODELO RELACIONAL

- Fue postulado por Edgar Frank Codd (IBM) en 1970.
- Surge como un nueva forma de organizar los datos.
- Todos los datos son almacenados en ***relaciones***. Cada relación es un conjunto de datos organizados, llamados ***tuplas***.
- Facilita la organización de grandes volúmenes de datos y garantiza la ***integridad*** de los mismos.

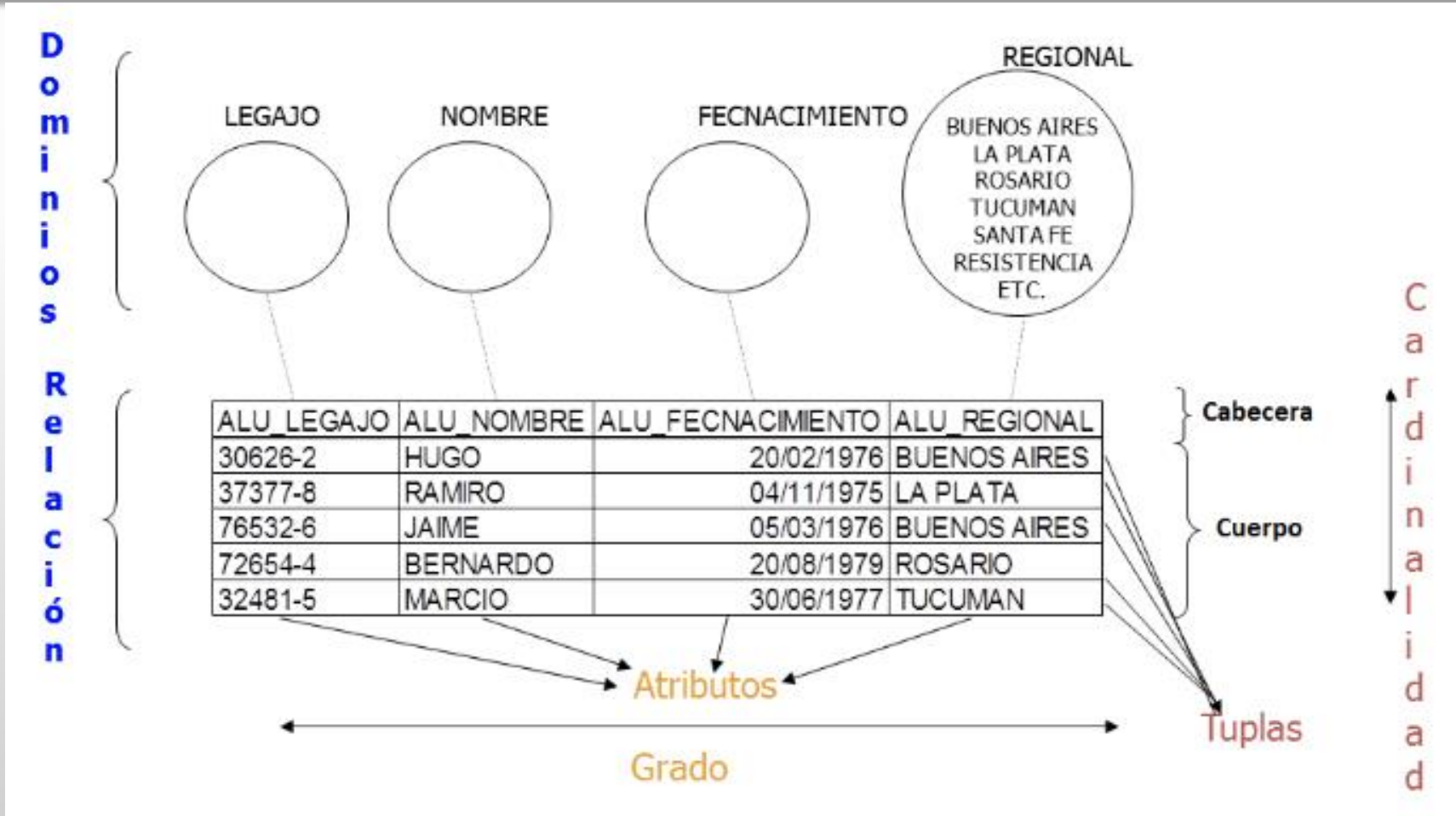
MODELO RELACIONAL

- ***Relación***: es la "entidad" donde se almacenan los datos.
- ***Tupla***: es el registro.
- ***Atributo***: es un Campo dentro de la Tupla.

MODELO RELACIONAL

- ***Dominio***: es el conjunto de valores posibles que puede tomar un Atributo. Es la menor unidad semántica de información.
- ***Cardinalidad***: número de tuplas.
- ***Grado***: número de Atributos.
- ***Clave Primaria***: es el identificador único de cada tupla.

MODELO RELACIONAL



BASE DE DATOS RELACIONAL

- *Es un tipo de Base de Datos que cumple con el Modelo Relacional.*
- *En una BDR, todos los datos se almacenan y se accede a ellos por medio de relaciones previamente establecidas.*
- *Las relaciones que almacenan datos son llamadas "**relaciones base**" y su implementación es llamada "**tabla**".*

BASE DE DATOS RELACIONAL

En una BDR los datos están organizados en relaciones llamadas Tablas, donde cada tupla es representada por una Fila, que a su vez contiene múltiples Columnas (atributos), y donde cada celda puede tomar alguno de los valores definidos en su dominio.

BASE DE DATOS RELACIONAL

Clave Primaria

Atributo que identifica de manera unívoca a cada fila de la tabla. Existen de 3 tipos: Naturales, Subrogadas o Compuestas.

BASE DE DATOS RELACIONAL

Índices

Es una estructura de datos que mejora la velocidad de las operaciones, por medio de un identificador único de cada fila de una tabla. Permite un rápido acceso a los registros de una tabla en una base de datos.

BASE DE DATOS RELACIONAL

Clave Foránea

Es un grupo de una o más columnas en una tabla que referencian a la clave primaria de otra tabla. Una Foreign Key puede ser parte de una Primary Key.

Tanto las PK como las FK son índices

BASE DE DATOS RELACIONAL

Integridad Referencial

La integridad referencial es una propiedad que establece que la clave externa de una tabla de referencia siempre debe corresponder a una fila válida de la tabla a la que se haga referencia. La integridad referencial garantiza que la relación entre dos tablas permanezca sincronizada durante las operaciones de actualización y eliminación.

BASE DE DATOS RELACIONAL

Tipos de Datos más utilizados

- *VARCHAR: Cadenas de caracteres compuestas por letras, números y caracteres especiales*
- *INTEGER: Números enteros naturales*
- *DOUBLE: Valores numéricos con punto flotante*
- *DATETIME: Formato para manejo de fechas*

BASE DE DATOS RELACIONAL

Constraints

Son restricciones al modelo que se utilizan para limitar el tipo de dato que puede ingresarse en una tabla. Se especifican cuando la tabla se crea por primera vez, o posteriormente realizando una actualización.

BASE DE DATOS RELACIONAL

Constraints más comunes

- ***Not Null**: No acepta valores nulos*
- ***Unique**: No acepta valores repetidos*
- ***Check** (o enum): Verifica que los valores cumplan determinada condición*
- ***Primary Key**: Clave Primaria*
- ***Foreign Key**: Clave Foránea*

BASE DE DATOS RELACIONAL

Relaciones

- Son asociaciones entre tablas.
- Se describen en la estructura de la Base de Datos.
- Las relaciones describen cómo se vinculan una o más tablas entre sí.

BASE DE DATOS RELACIONAL

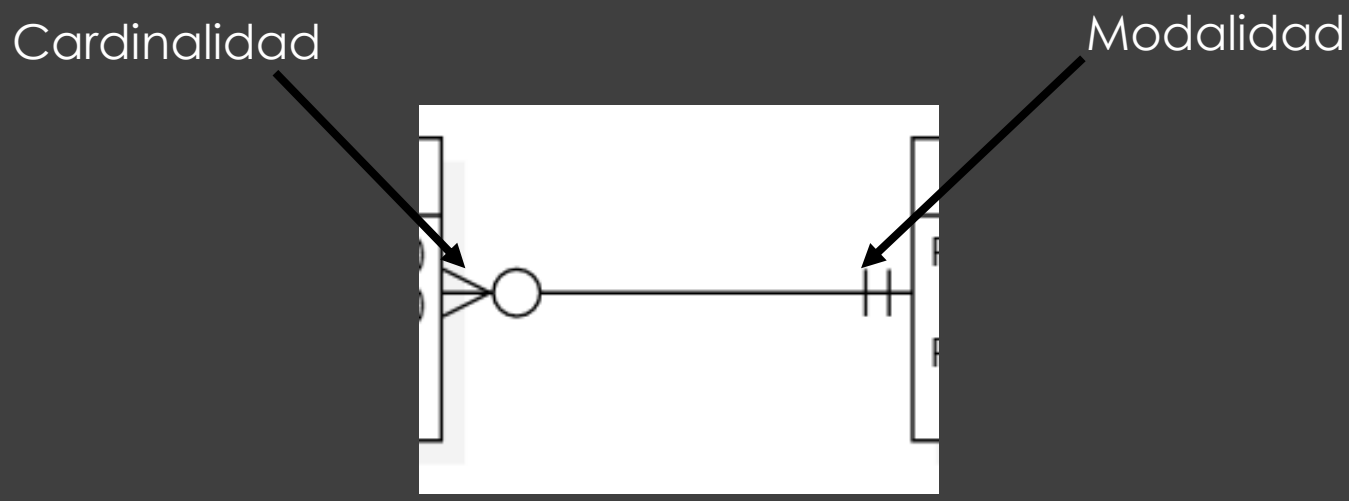
Cardinalidad y Modalidad

- La ***cardinalidad*** es la cantidad máxima de relaciones que tiene un registro de una entidad (tabla) con respecto a la otra tabla.

- La ***modalidad*** es la cantidad mínima de relaciones que tiene un registro de una entidad (tabla) con respecto a la otra tabla.

BASE DE DATOS RELACIONAL

Cardinalidad y Modalidad



BDR – ONE TO ONE

Relación One to One

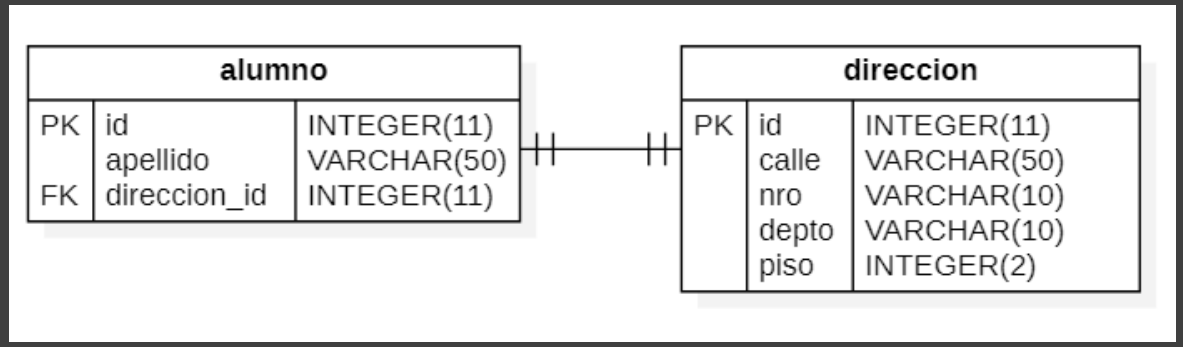
- Es una relación entre dos entidades (tablas) A y B, en la cual un registro de la entidad A se corresponde con un único registro de la entidad B.

- La relación puede ser unidireccional, en cualquier sentido, o bidireccional.

BDR – ONE TO ONE

Relación One to One

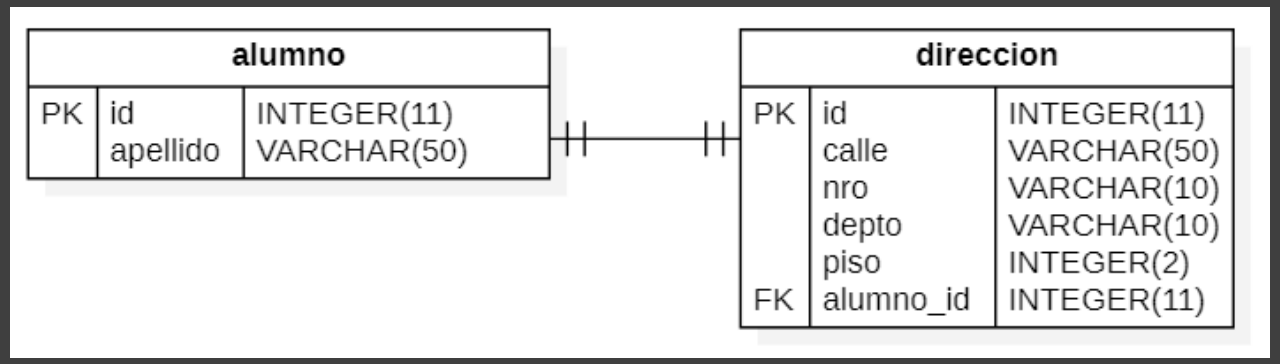
- ***Unidireccional desde el lado A***: existe un registro en la tabla B que está siendo referenciado por un único registro de la tabla A.



BDR – ONE TO ONE

Relación One to One

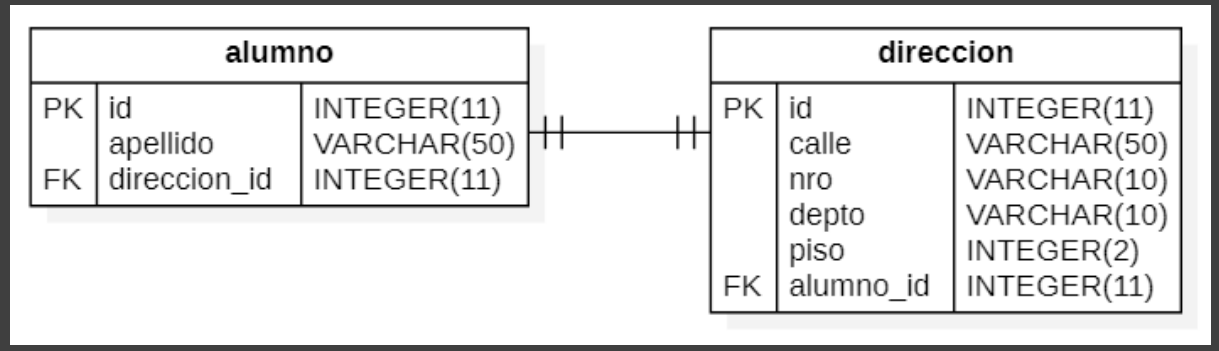
- ***Unidireccional desde el lado B***: existe un registro en la tabla A que está siendo referenciado por un único registro de la tabla B.



BDR – ONE TO ONE

Relación One to One

- ***Bidireccional***: existe un registro en la tabla A que está siendo referenciado por un único registro en la tabla B; y a su vez, este registro de la tabla B está siendo referenciado por el registro de la tabla A.



BDR – ONE TO MANY

Relación One to Many

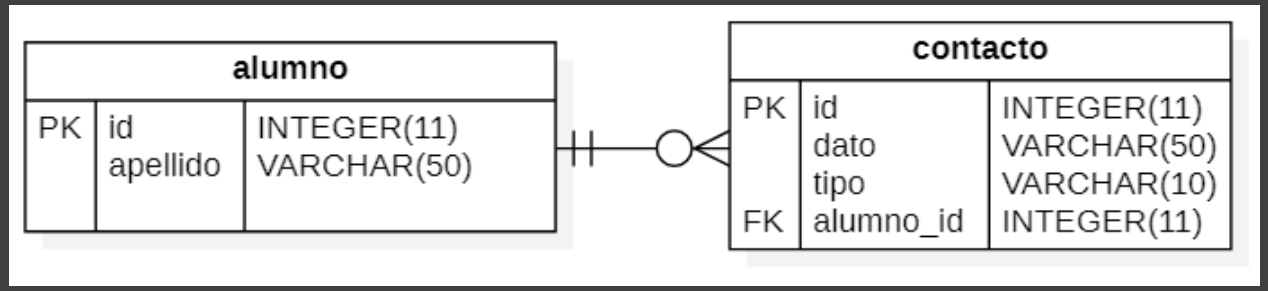
- Es una relación entre dos entidades (tablas) A y B en la cual un registro de la entidad A está siendo referenciado por muchos registros de la entidad B.

- Significa que "***A tiene muchos B***", porque cada registro de la tabla B pertenece a un registro de la tabla A.

BDR – ONE TO MANY

Relación One to Many

"Un alumno puede tener muchos contactos"



BDR – MANY TO ONE

Relación Many to One

- Es una relación entre dos entidades (tablas) A y B en la cual muchos registros de la entidad A están referenciado al mismo registro de la entidad B.

- Significa que "***A pertenece a un B, y B tiene muchos A***", porque cada registro de la tabla A referencia a un registro de la tabla B, pero B puede ser referenciado muchas veces.

- También se puede leer como "***Muchos A pertenecen al mismo B***"

BDR – MANY TO ONE VS ONE TO MANY

Many to One vs One to Many

Si se tienen en cuenta dos entidades A y B podríamos afirmar que:

- Si A tiene una relación One to Many con la entidad B, entonces B tiene una relación Many to One contra la entidad A.
- Si A tiene una relación Many to One contra la entidad B, entonces B tiene una relación One to Many contra la entidad A.

BDR – MANY TO ONE VS ONE TO MANY

Many to One vs One to Many

La relación entre las entidades A y B, en este caso, pueden llamarse de una u otra forma dependiendo el "lado" que miremos. En otras palabras, es una cuestión de perspectiva.

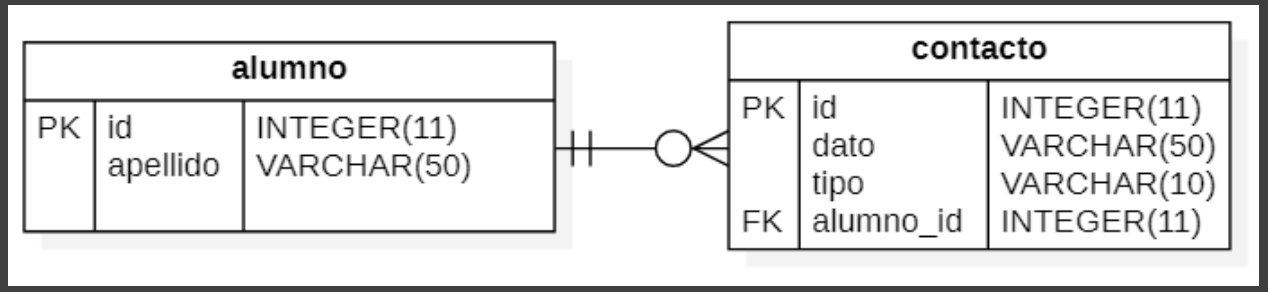
Regla práctica: si existe una relación One to Many, visto desde A hacia B, entonces "*del otro lado*" existe una relación Many to One (visto desde B hacia A). La inversa también es válida.

BDR – MANY TO ONE VS ONE TO MANY

Many to One vs One to Many

"Un contacto pertenece a un alumno, y un alumno puede tener muchos contactos".

"Muchos contactos pueden pertenecer al mismo alumno"



BDR – MANY TO MANY

Relación Many to Many

- Es una relación entre dos entidades (tablas) A y B en la cual un registro de la entidad A puede estar siendo apuntado por muchos registros de la entidad B, y un registro de la entidad B puede estar siendo apuntado por muchos registros de la entidad A.

- Significa que "***A tiene muchos B, y B tiene muchos A***".

BDR – MANY TO MANY

Relación Many to Many

- Esta relación no es posible realizarla en el modelo relacional, ya que sería necesario tener múltiples FKs (sin saber el número exacto) en ambas tablas involucradas en la relación.

BDR – MANY TO MANY

Relación Many to Many

- Para poder implementar esta relación se debe partir la misma en dos relaciones One To Many.

- La entidad A debe tener una relación One to Many contra la entidad C, considerada "*entidad intermedia*"; al igual que la entidad B.

BDR – MANY TO MANY

Relación Many to Many

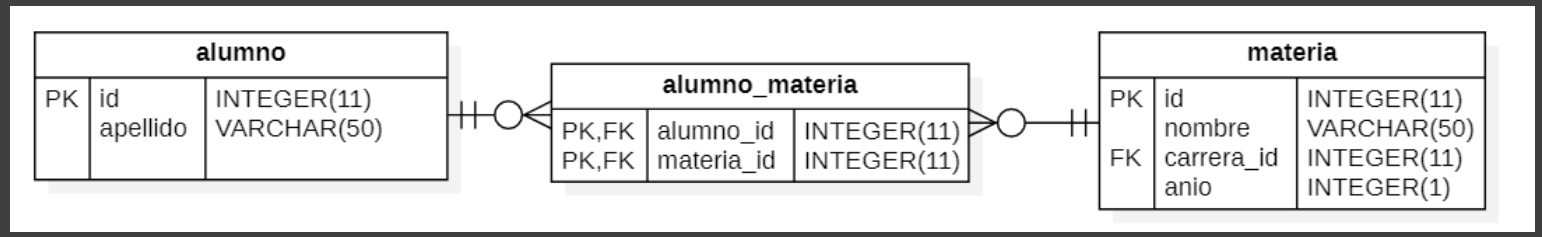
- La entidad C, famosa por ser "*la tabla intermedia*", debe poseer una FK a la entidad A y una FK a la entidad B.

- Además, la entidad C podría guardar algún dato extra necesario de la relación.

BDR – MANY TO MANY

Relación Many to Many

"Un alumno puede estar cursando varias materias, y una materia puede ser cursada por muchos alumnos"



BDR – MANY TO MANY

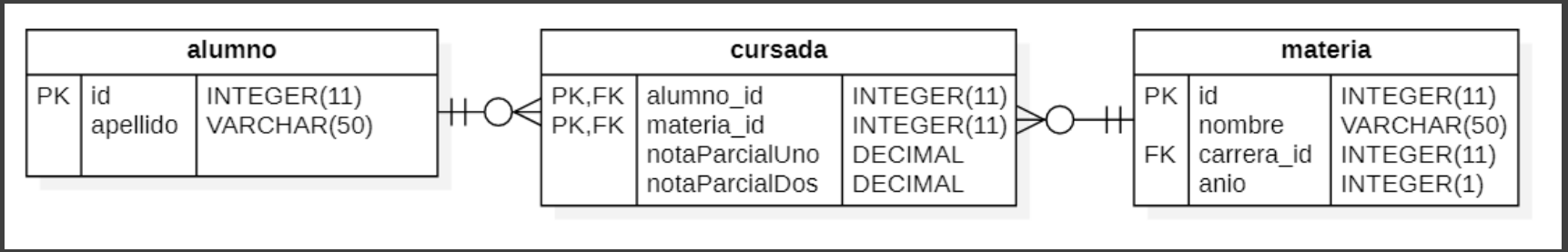
Relación Many to Many

- En el caso anterior, suponemos que no interesa guardar ningún dato extra sobre las materias que está cursando un alumno en particular.

- Si el dominio lo amerita y necesitamos guardar algún dato extra, como las notas del primer y segundo parcial, el diseño se podría mejorar y "*alumno_materia*" dejaría de ser una simple "*tabla intermedia*".

BDR – MANY TO MANY

Relación Many to Many



ESTRATEGIAS DE PERSISTENCIA

Persistencia en Bases de Datos Relacionales

Retomando desde el punto que desembocó en la anterior explicación…

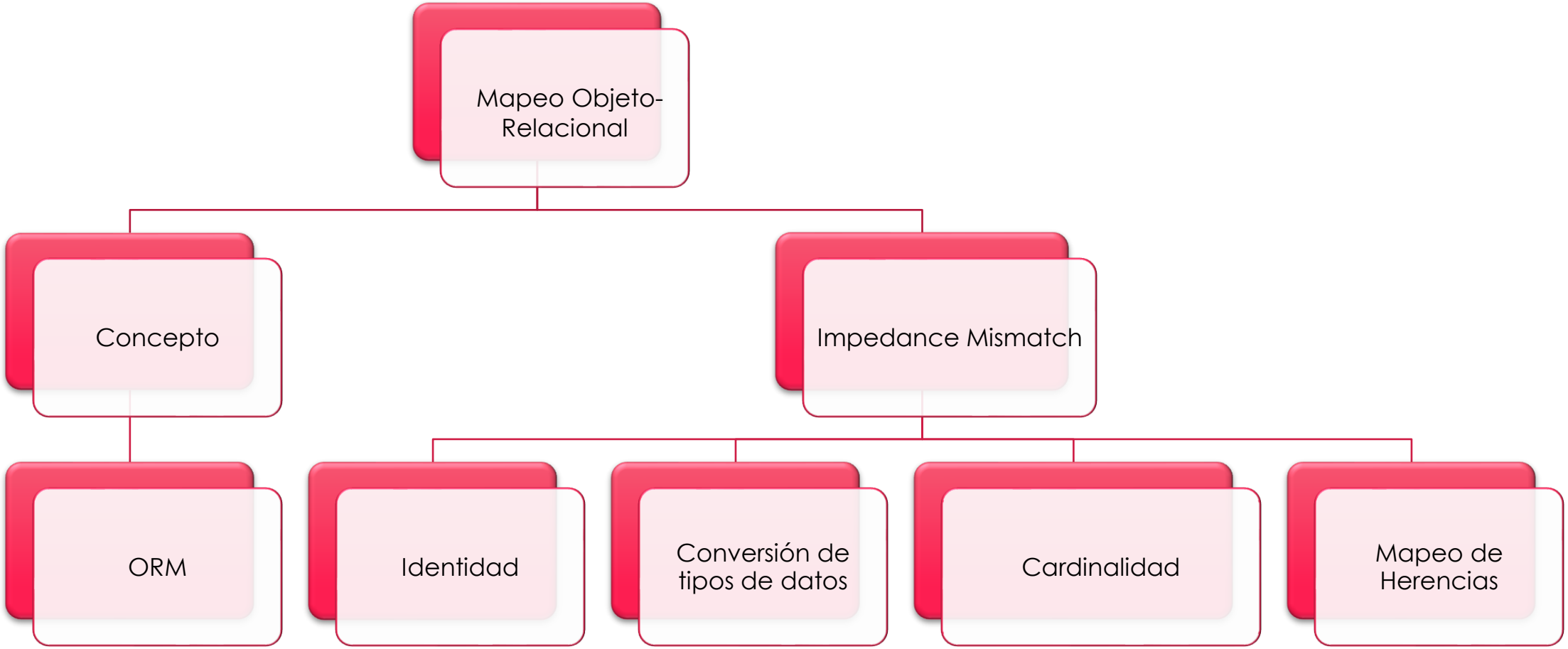
El estado de un Sistema puede ser persistido en una, o varias, Base de Datos Relacional.

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

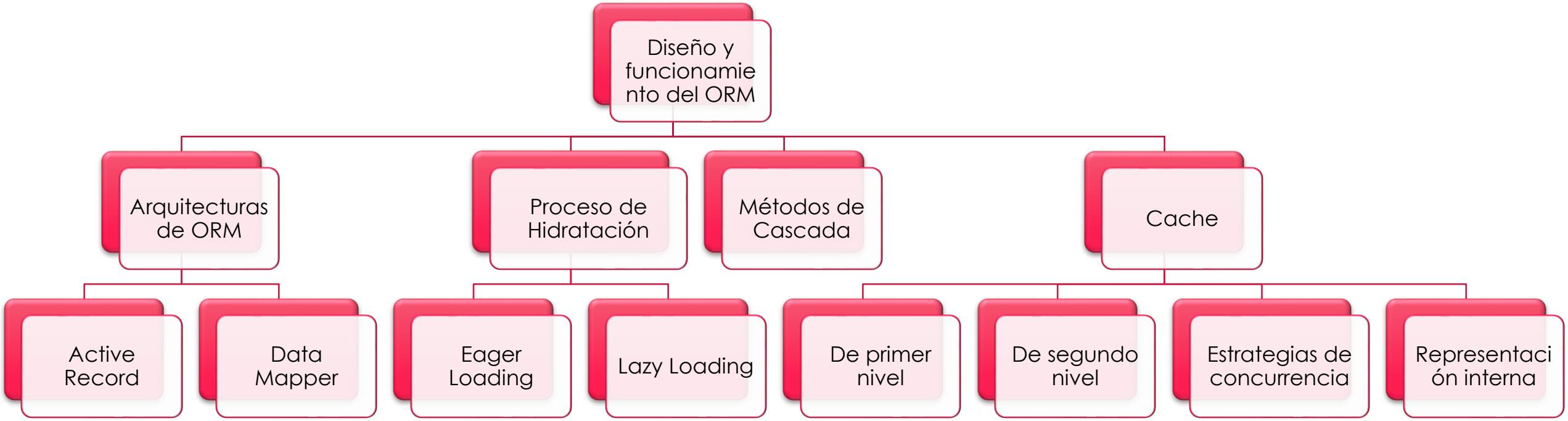
- Todos los atributos persistentes de la superclase serán persistidos en cada una de las tablas que mapean contra las clases hijas.

IMPEDANCE MISMATCH – MAPEO DE LA HERENCIA

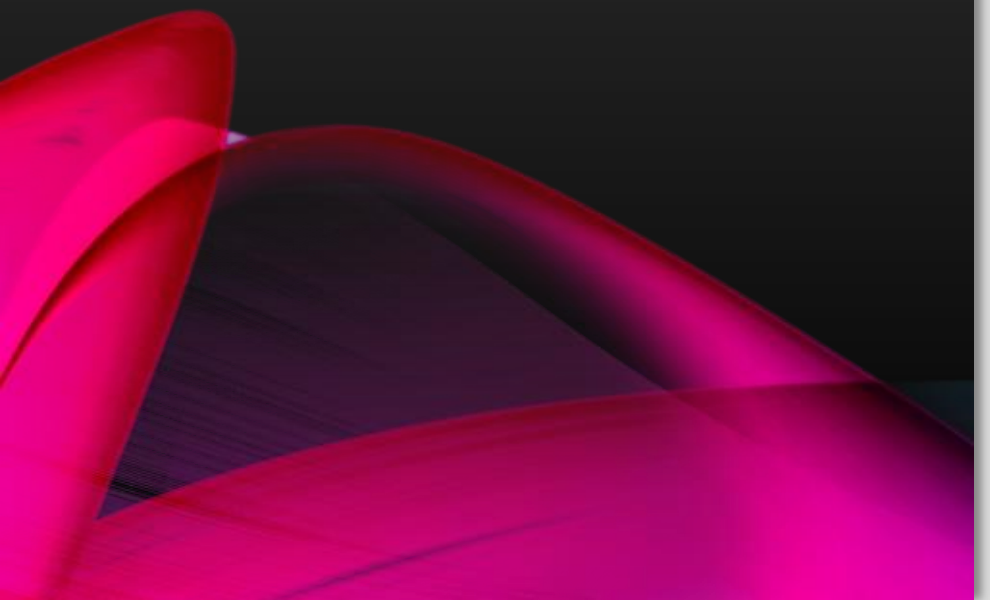
Table per Class

- 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

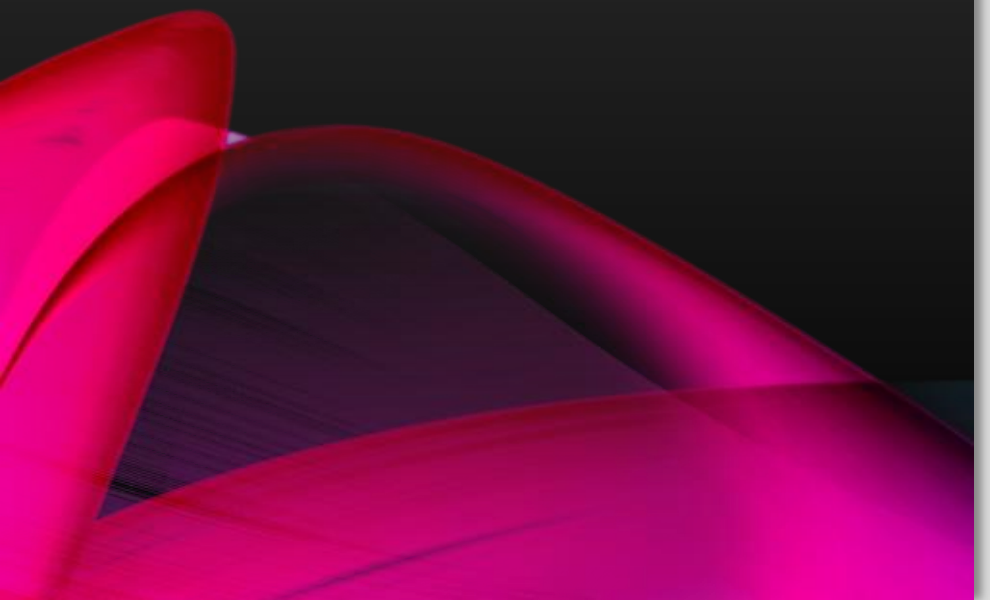
PROCESO DE HIDRATACIÓN



¿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

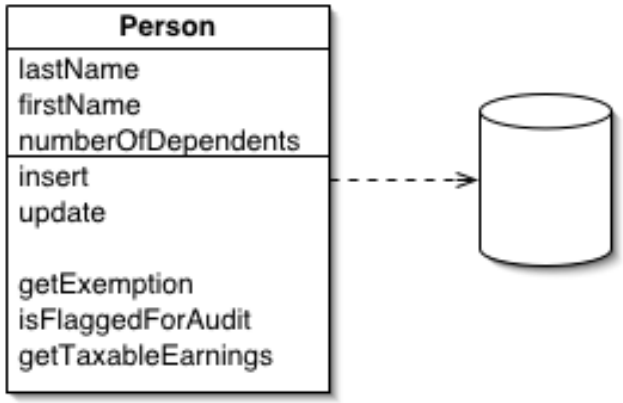
ARQUITECTURAS DE ORM

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

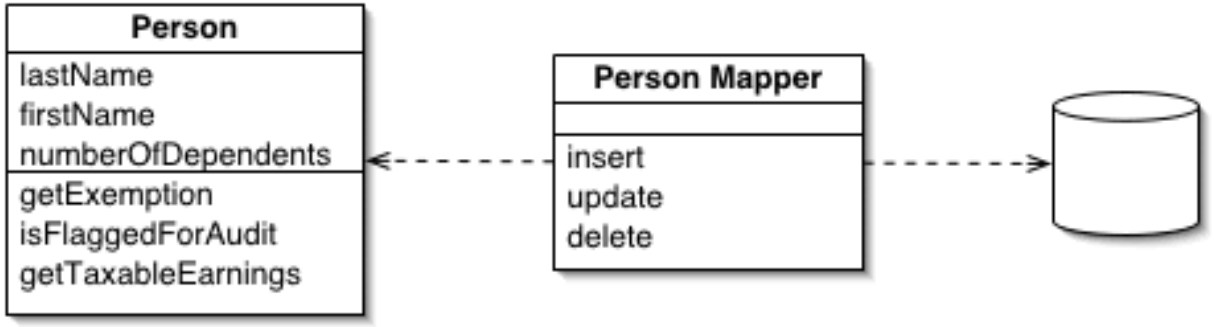
Data mapper

- 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

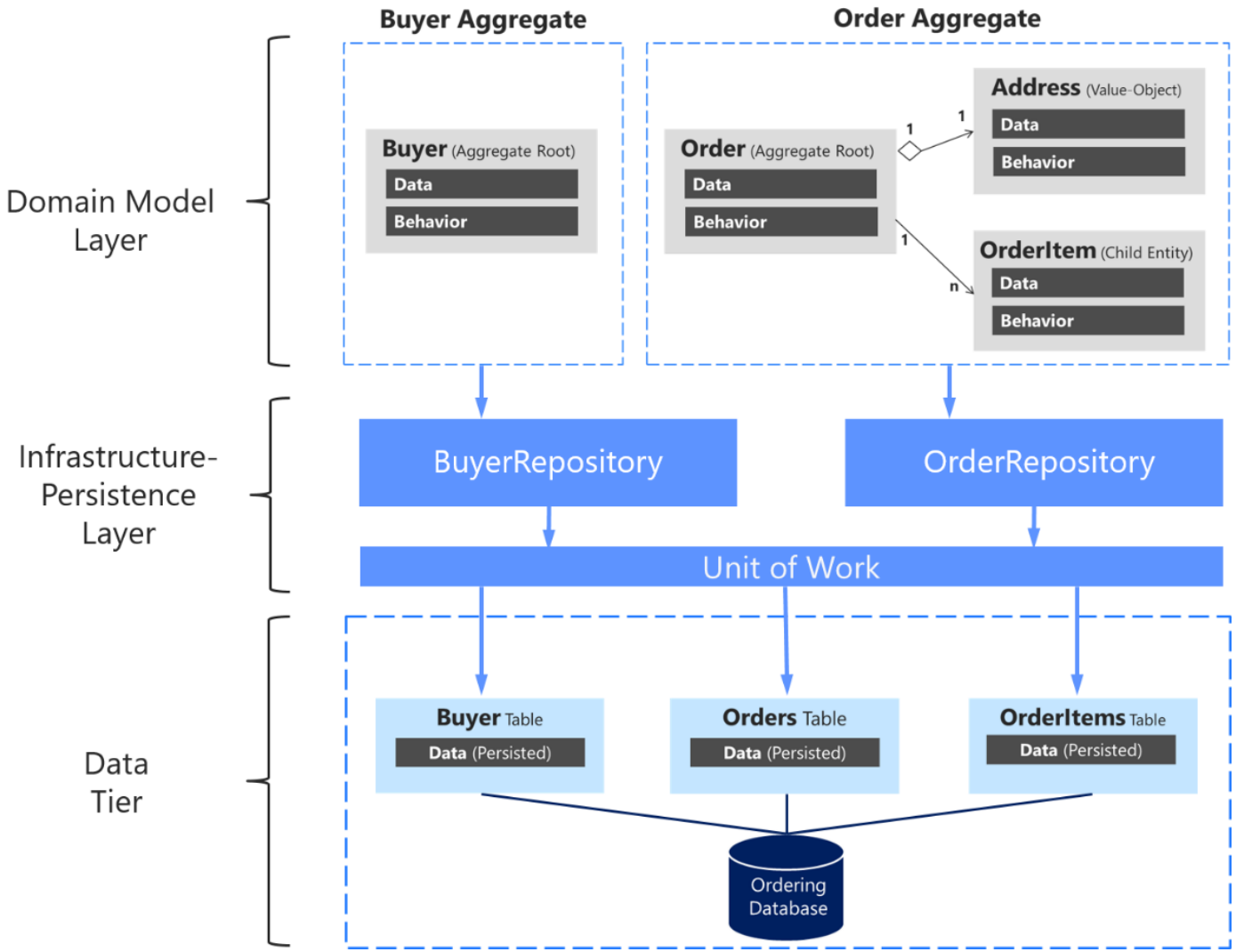
- *Propone que en la capa de persistencia existan objetos "Repository" .*

- *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]*