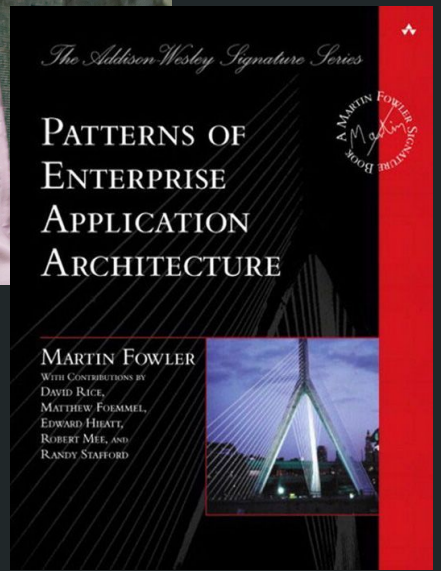
Patterns of Enterprise Application Architecture

“PEAA” para los amigos - 1er cuatrimestre 2023

Generalidades

Martin Fowler

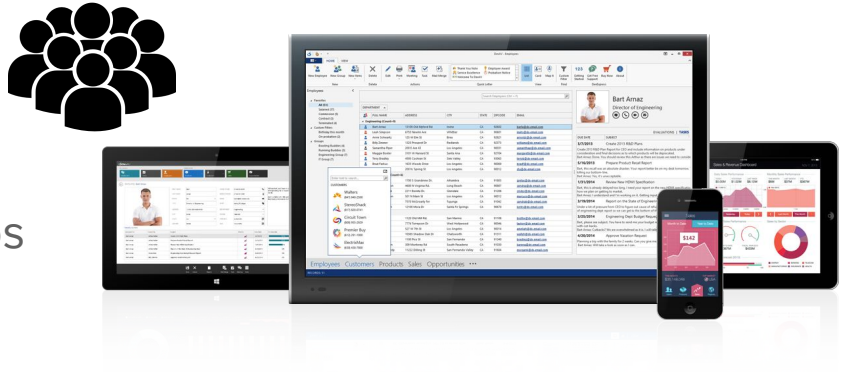
El Señor de los Patrones



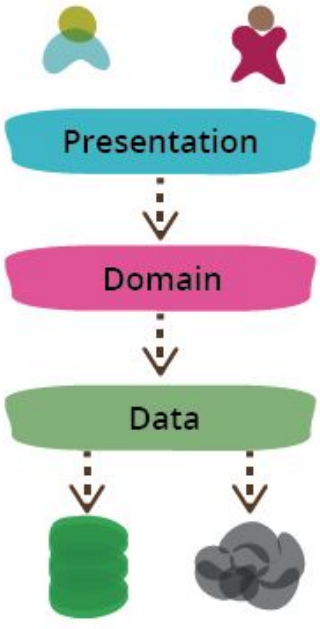
2002

¿”Enterprise” Application?

- Datos
  - persistentes
  - muchos
  - concurrencia en su acceso
- Interfaces de usuario
  - muchas
  - distintas vistas de los mismos datos
- Integración con otras apps “enterprise”
- Lógica de negocio
  - compleja, por lo general
- Disonancia conceptual

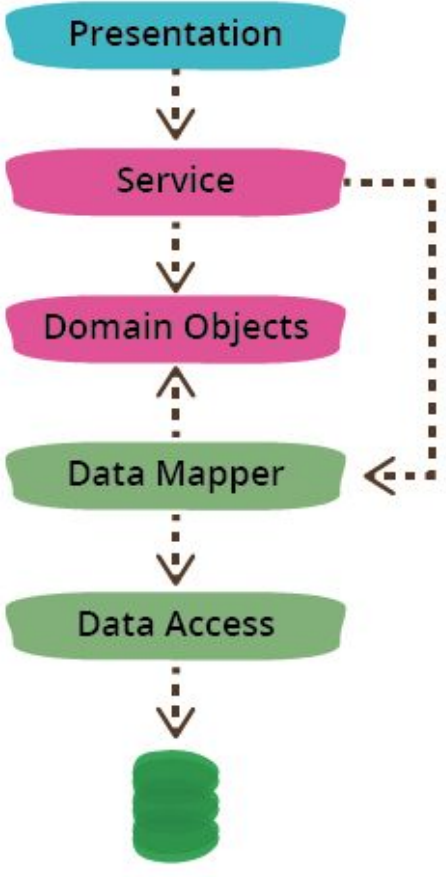


Arquitectura lógica: Layers

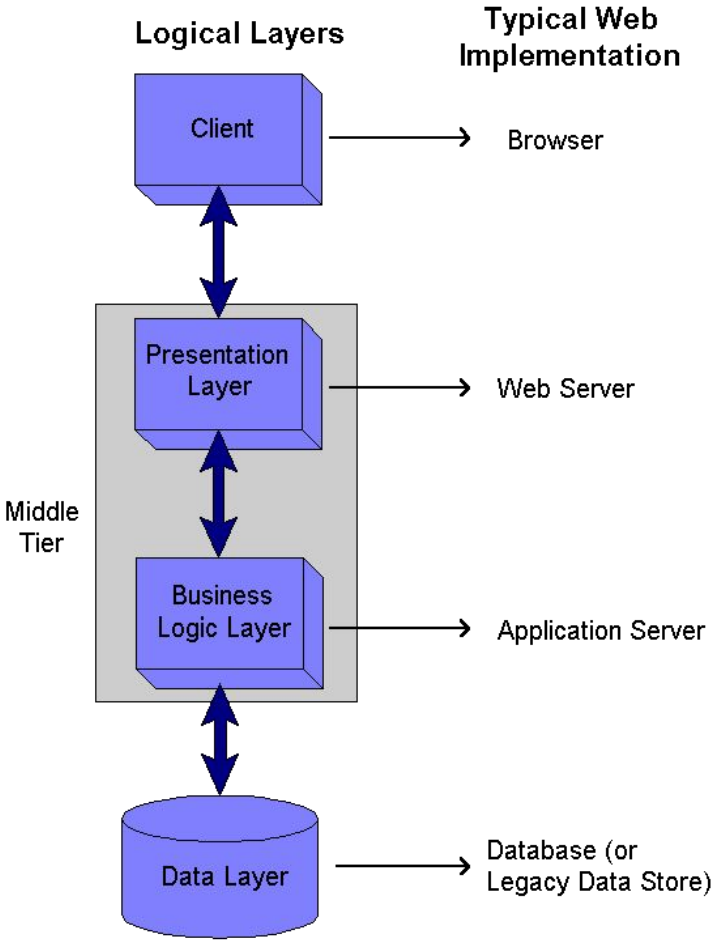


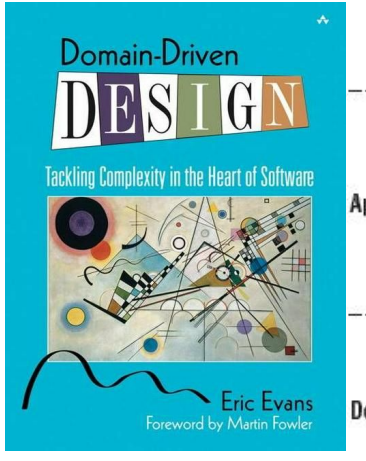
- “Presentation-Domain-Data” layering
  - Permite focalizarse en un área a la vez de forma relativamente independiente
  - Modifiability
  - Testability

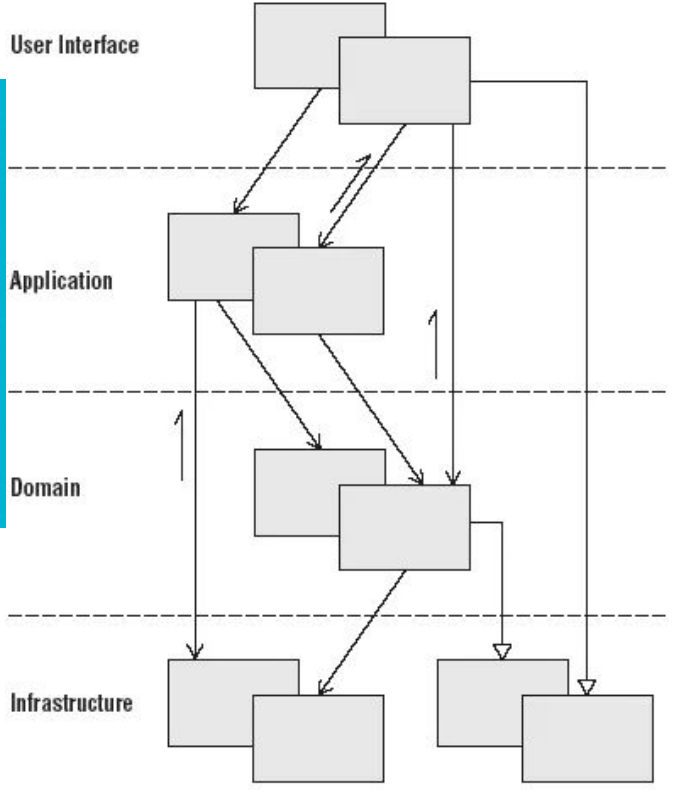
Existen diversas variantes que agregan layers →



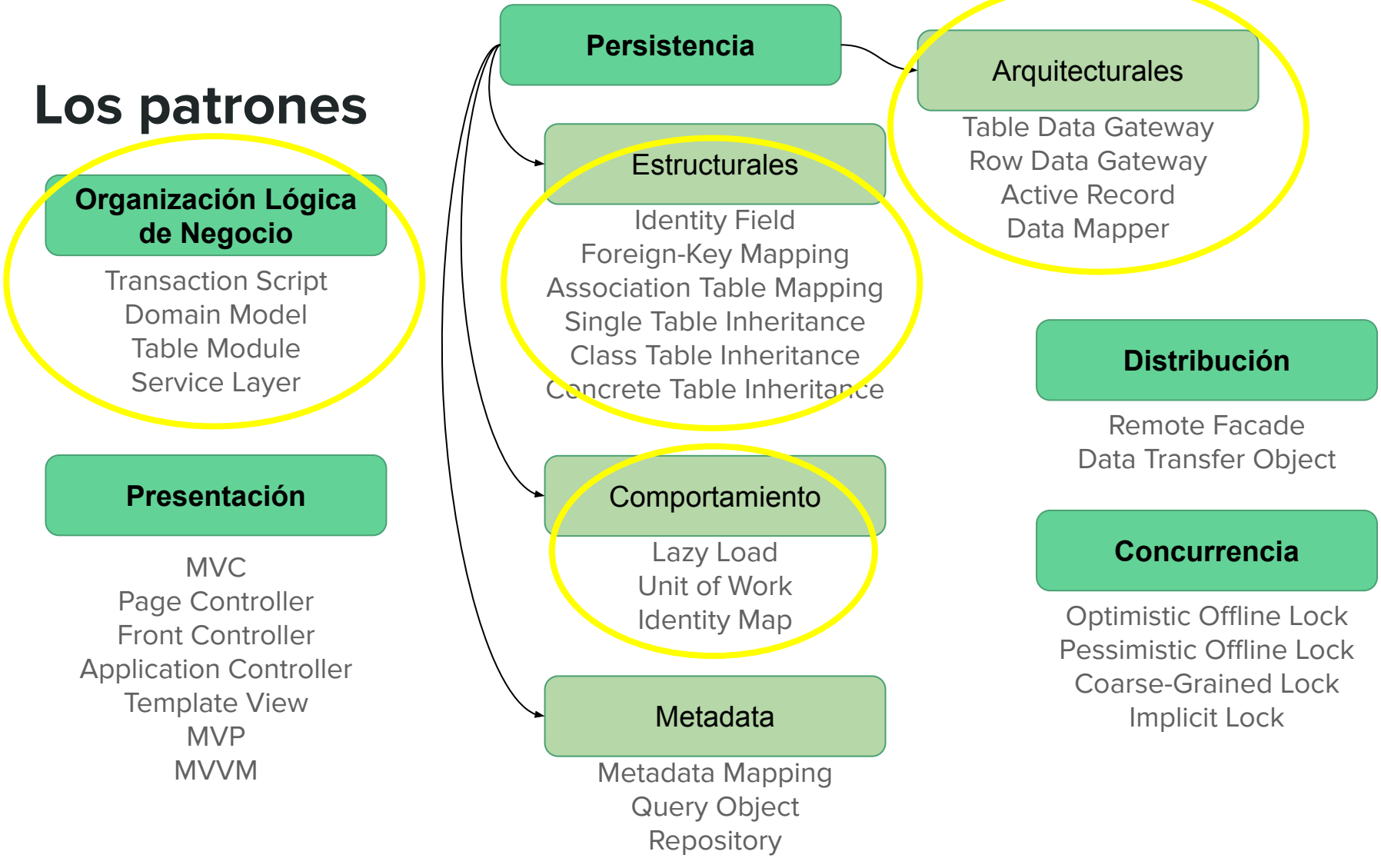
Arquitectura lógica: Layers - más variantes







Los patrones



Organización de la Lógica de Negocio

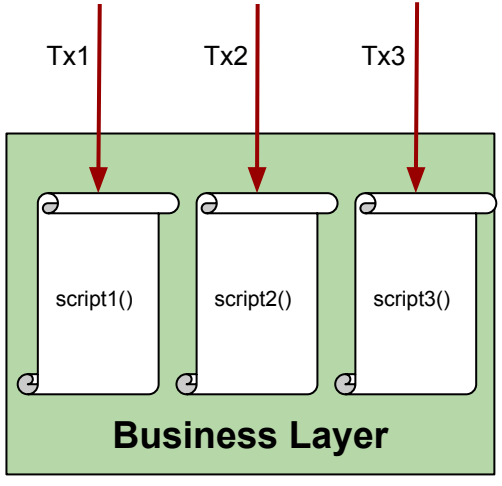
Organización Lógica de Negocio

Transaction Script
Domain Model
Table Module
Service Layer

Los tres patrones principales

Transaction Script

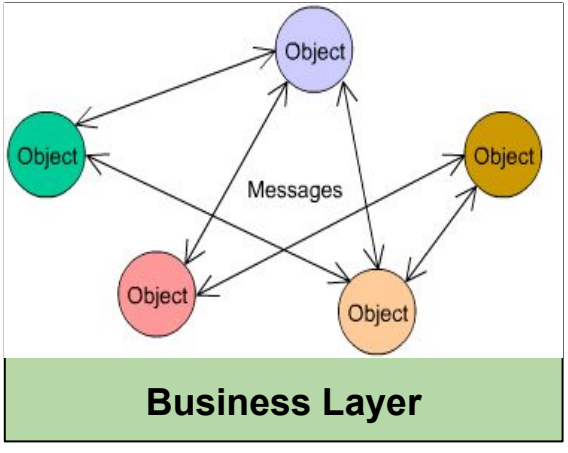
Programación Estructurada



1 procedimiento (script) por cada acción (transacción)

Domain Model

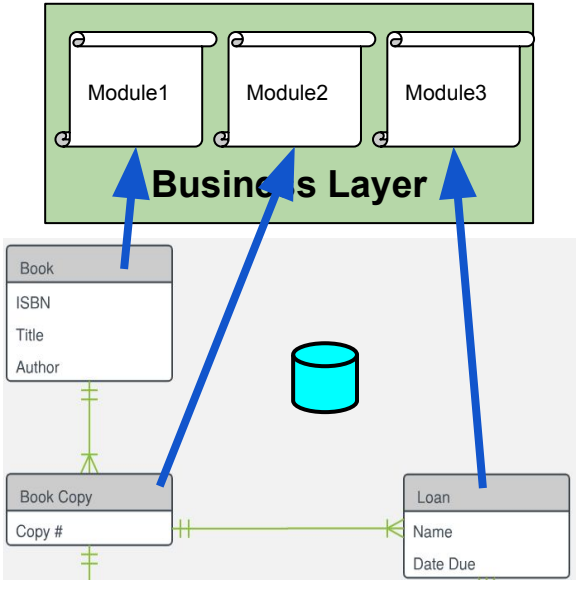
Diseño orientado a Objetos



Objetos y mensajes

Table Module

Modelo centrado en los datos



1 módulo/clase por cada entidad de datos (ó tabla)

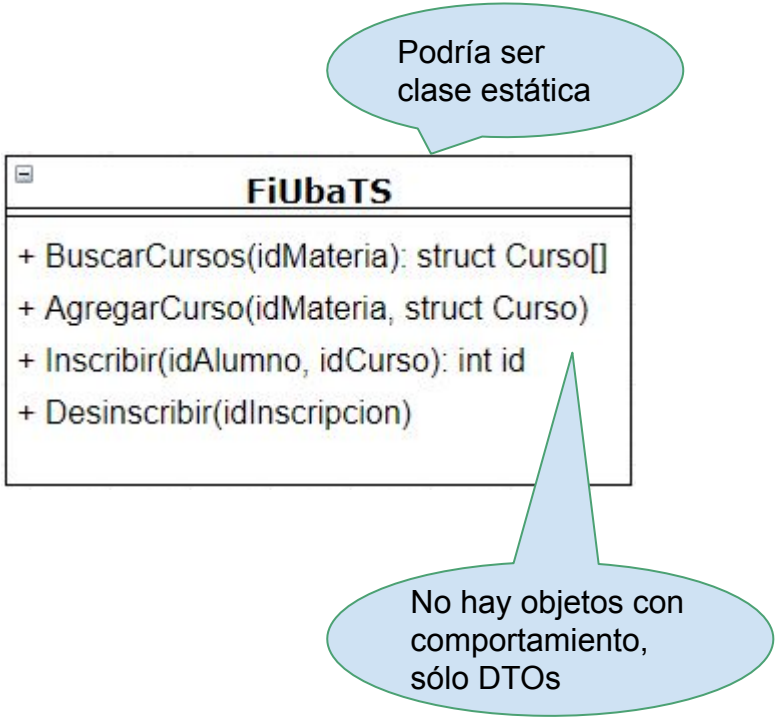
Transaction Script (TS)

1 procedimiento (script) por cada acción (transacción)

Procesa input, accede a BD, ejecuta lógica dominio, formatea resultado para UI

Puede subdividirse en subprocedimientos o llamar a otros TS

Implementable en lenguajes procedurales y orientados a objetos



\+ Simple
\+ Se adapta bien a simple Data Layer
\+ Poco overhead: buena performance
\- Duplicación de código entre scripts difícil de evitar siempre
\- Más reglas de negocio, más lógica condicional en scripts
\- Puede terminar en muchas rutinas sin estructura

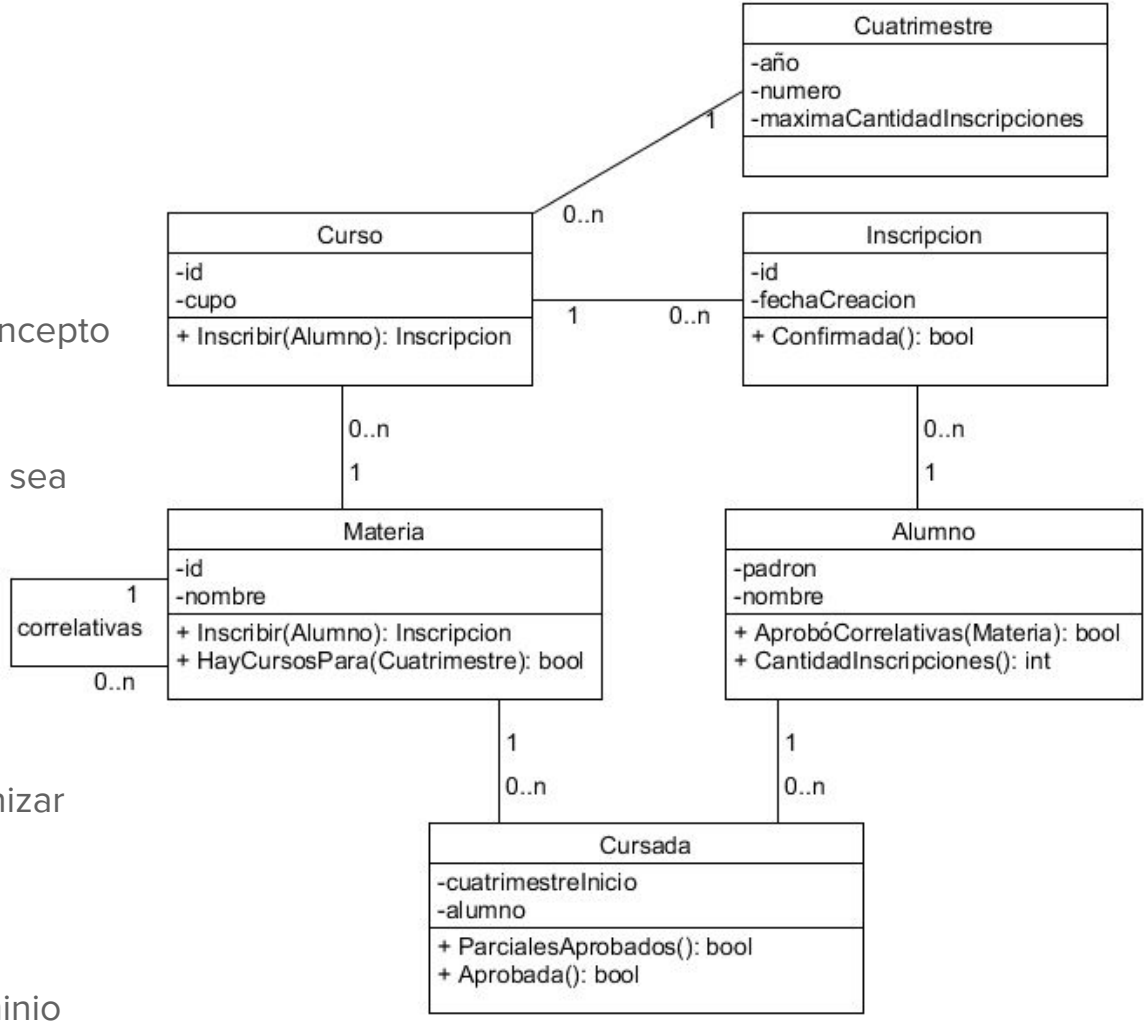
Domain Model (DM)

Diseño orientado a objetos

1 instancia por cada ocurrencia de un concepto del dominio

Cada objeto participa en la lógica que le sea relevante

El lenguaje debe soportar OOP



+ Varias técnicas para manejar y organizar lógica compleja (polimorfismo, composición, encapsulamiento, etc.)
- Mapeo complejo a BD
- Requiere skills de modelado de dominio

Table Module (TM)

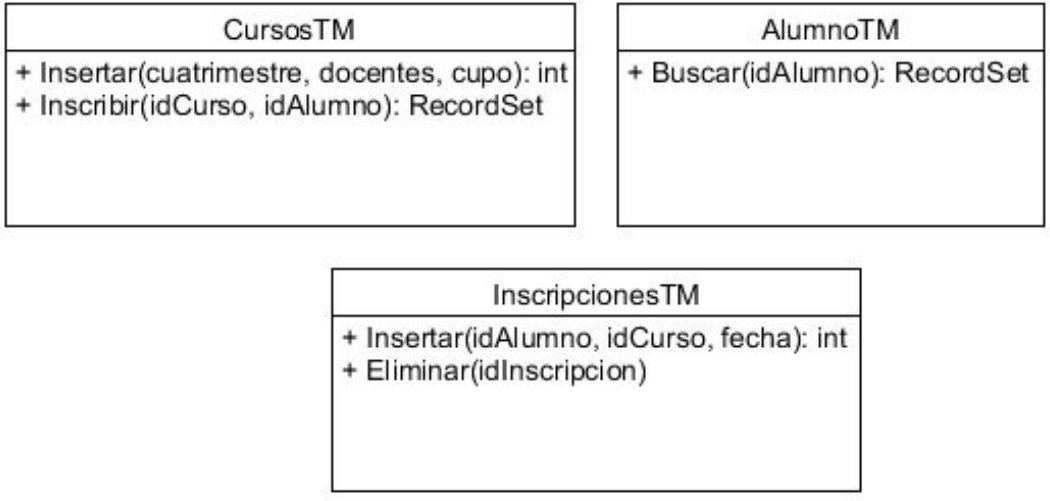
1 módulo por cada entidad de datos

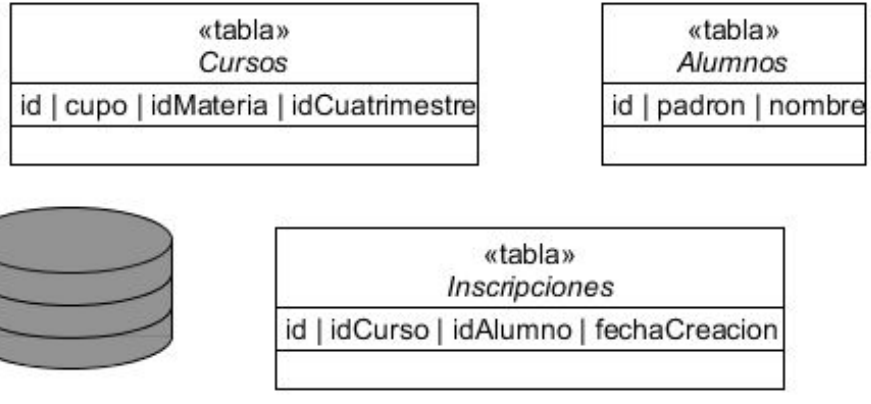
La lógica de dominio es organizada según la estructura de tablas

En lenguajes OO: 1 instancia para todas las filas de una “tabla virtual” percibida por la app

Diseñado para trabajar con RecordSets

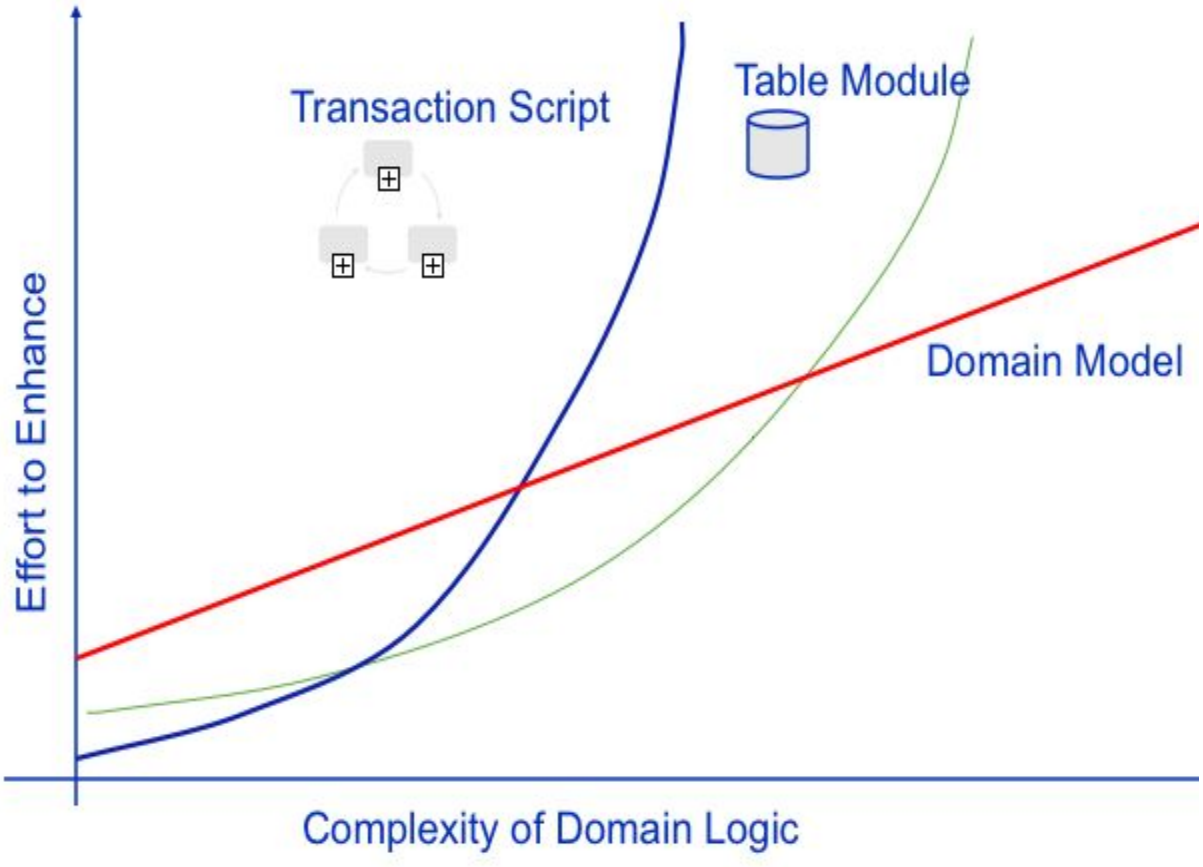
Pueden invocarse entre sí





\+ Muchas plataformas ofrecen potentes herramientas para RecordSets
\+ Mejor que TS para organizar lógica
\- No aprovecha herramientas de OOP

¿Cuál elegir?



Otras variables

- ¿BD ya definida?
- Expertise dev team
- Soporte para RecordSets

Y... podrían armarse híbridos

Service Layer (SL)

Define la frontera (API) de la aplicación

¿Más de 1 tipo de cliente?

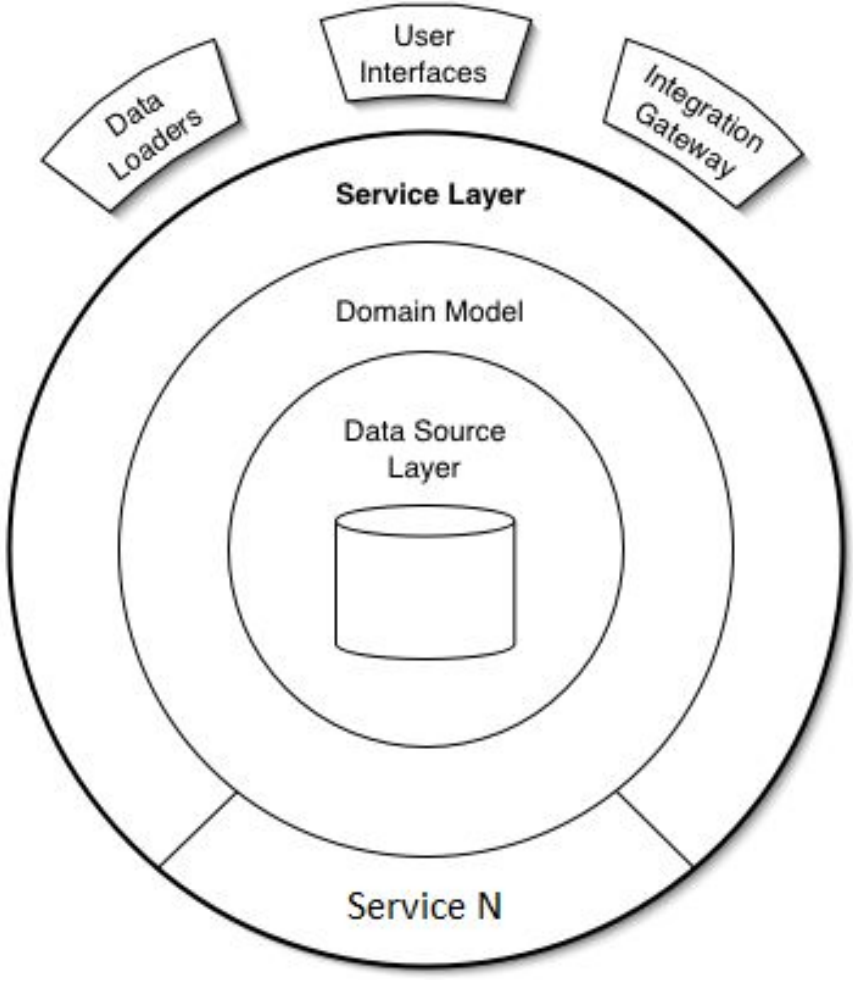
Business logic = Domain logic + App logic

SL define API clara, coordina respuestas, controla recursos transaccionales

Opciones implementación:

- Domain Facade
- Operation Script: con app logic

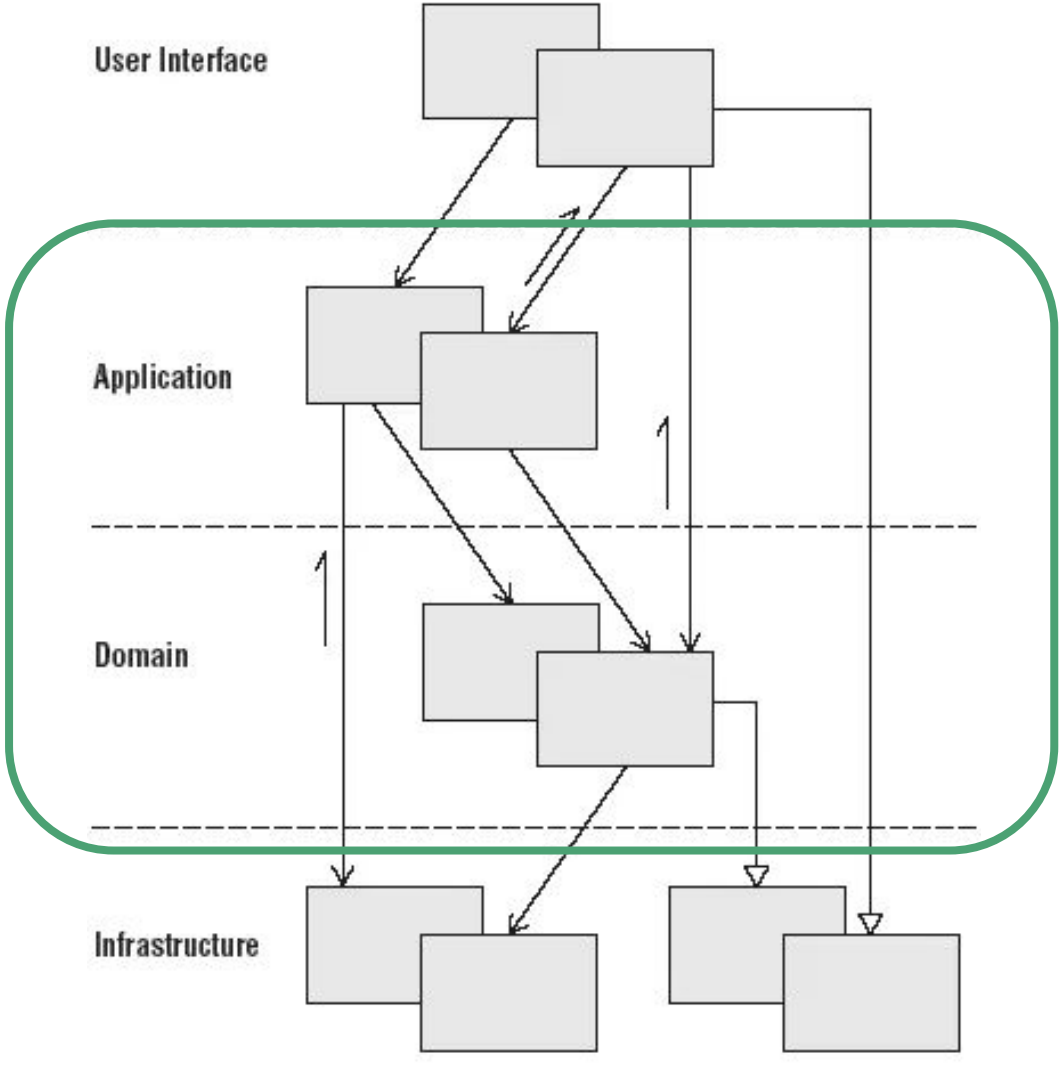
Opcional: Remote Facade, Layer Supertype



Recapitulando

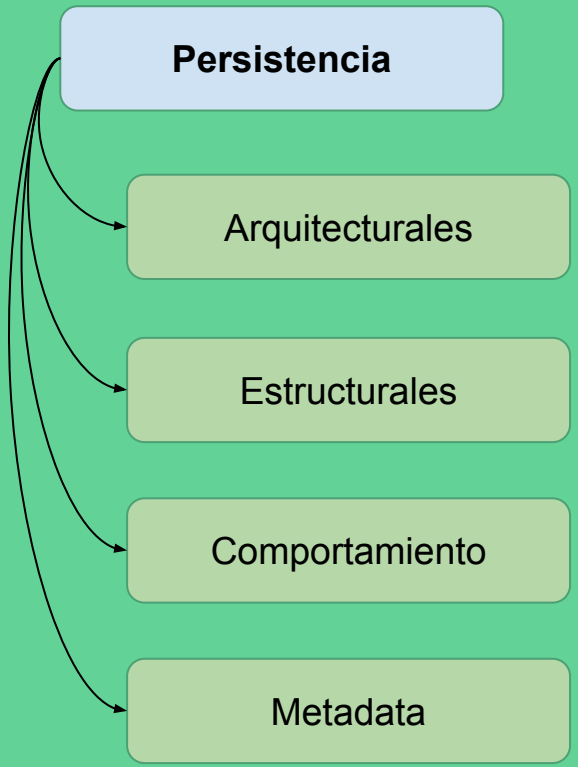
Service Layer

Transaction Script
Domain Model
Table Module



Persistencia

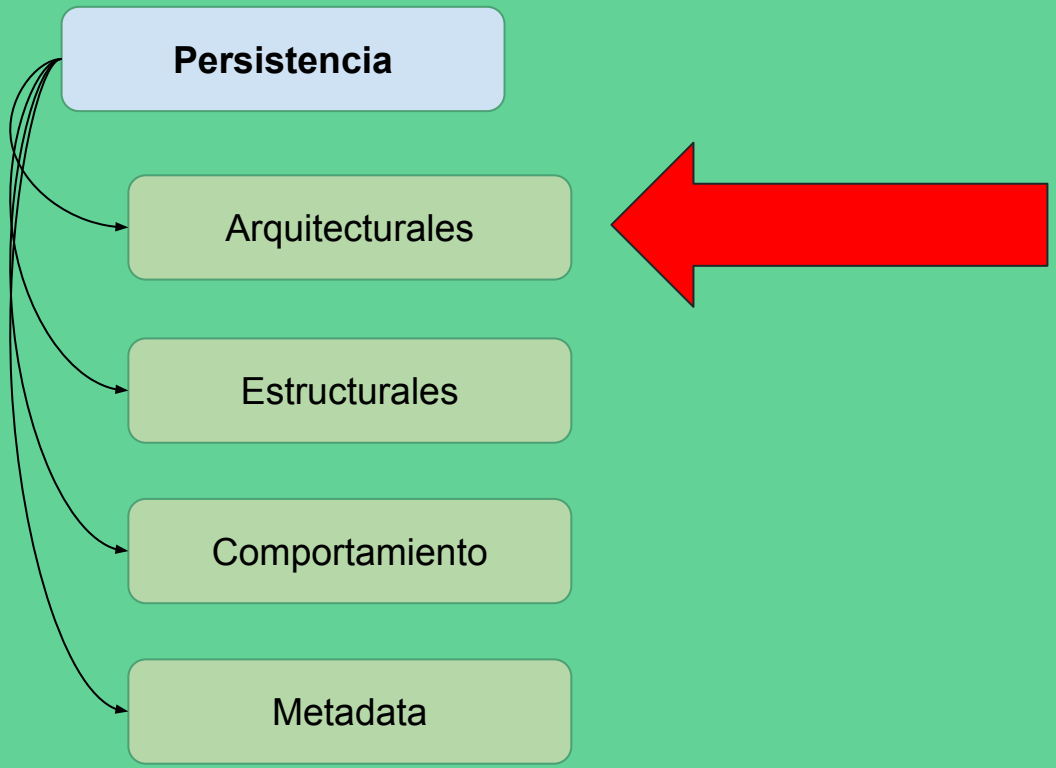
Data Access Layer



Antes de empezar...

Mapeo Objeto-Relacional

- BD relacionales se siguen usando
- Mapeo manual requiere mucho esfuerzo
- Usar herramientas (ORM) siempre que sea posible, a menos que el problema sea trivial
- ¿Para qué estudiarlo?


Persistencia
Arquitecturales
Estructurales
Comportamiento
Metadata

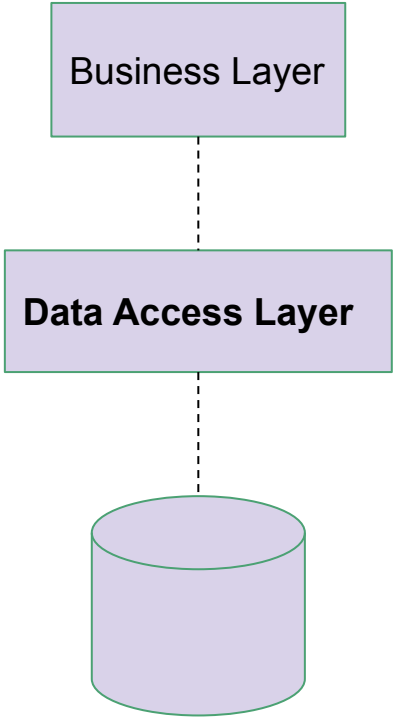
Persistencia - Arquitecturales

¿Cómo organizar la capa de acceso a datos?

El diseño de alto nivel de la DAL

Arquitecturales

Table Data Gateway
Row Data Gateway
Data Mapper
Active Record



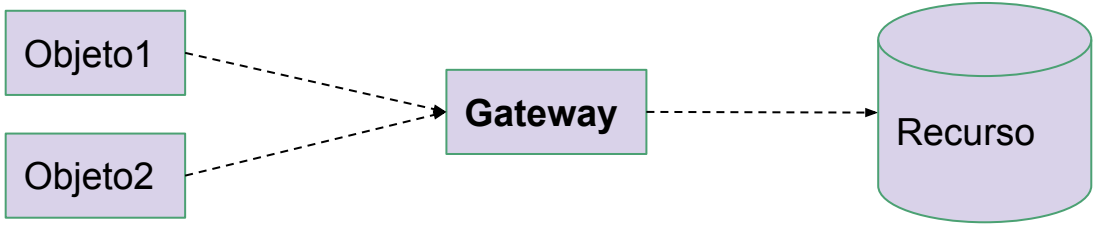
Persistencia - Arquitecturales - Patrones Base

Arquitecturales

Table Data Gateway
Row Data Gateway
Data Mapper
Active Record

Problema: Acceso desde la Business Layer a un recurso

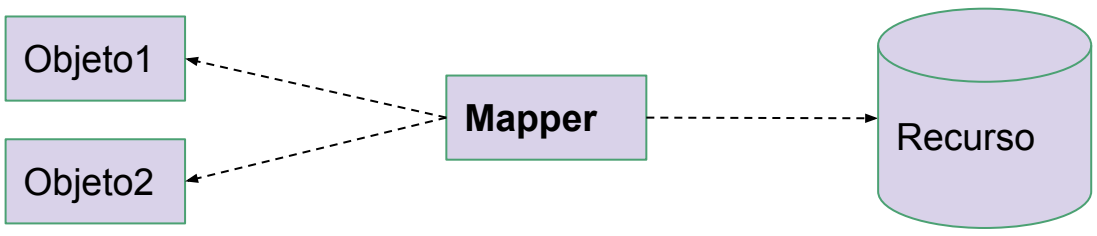
Patrón base: **Gateway** (aka Wrapper)



¿Cómo invocar al Mapper?

3er subsistema ó hacerlo Observer

Patrón base: **Mapper**

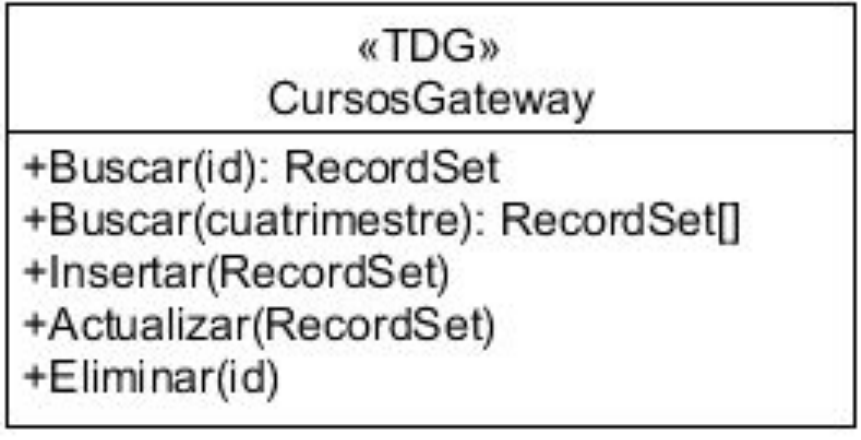


Independencia mutua entre los objetos y el recurso

Table Data Gateway (TDG)

Gateway a todos los registros de 1 tabla

1 instancia de un TDG por cada tabla



Stateless

Tipo de dato de las respuestas

- datos escalares (int, string, date, etc.)
- key-value pairs (mapas)
- DTOs
- RecordSets

Funciona bien con Table Module

Row Data Gateway (RDG)

Gateway a un registro (fila)

1 instancia de un RDG por cada registro

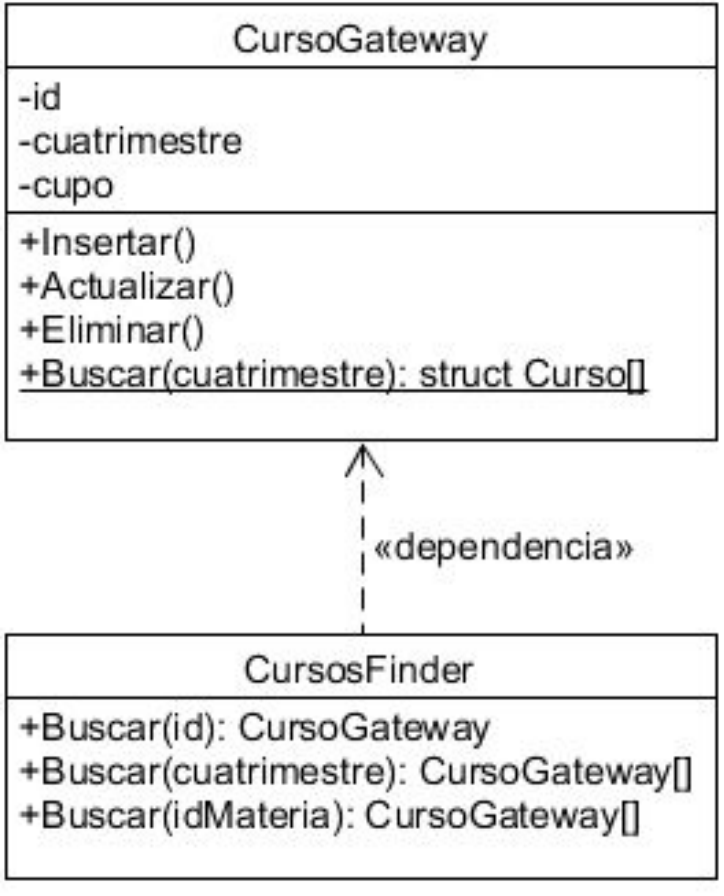
Parece un registro de la tabla correspondiente

Todo el código de mapeo va en el Gateway

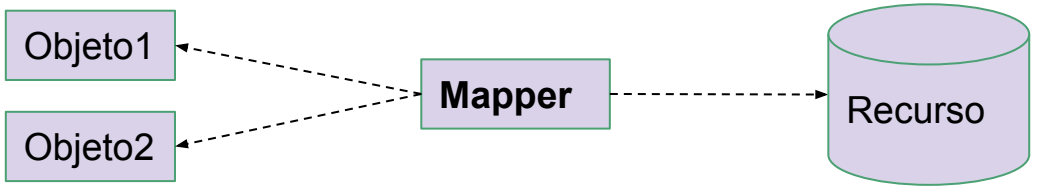
NO hay lógica de dominio en estos objetos

Consultas: métodos estáticos ó finders separados

Funciona bien con Transaction Script



Data Mapper (DMap)



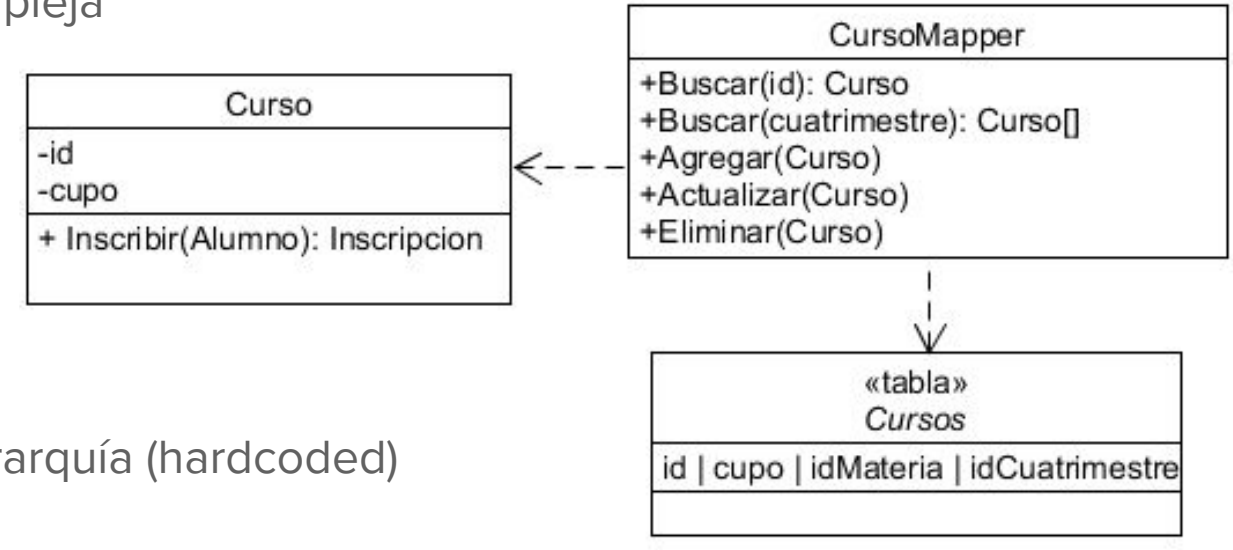
Mapper entre objetos del Domain Model y registros de la BD

Separa objetos de la BD: ni saben que hay una Data Layer

Útil para lógica dominio compleja

Sustituíble por completo

(testability)



Cantidad

- 1 mapper por clase o jerarquía (hardcoded)
- Metadata Mapping

Uso de Unit of Work, Lazy Load, Identity Map

Data Mapper

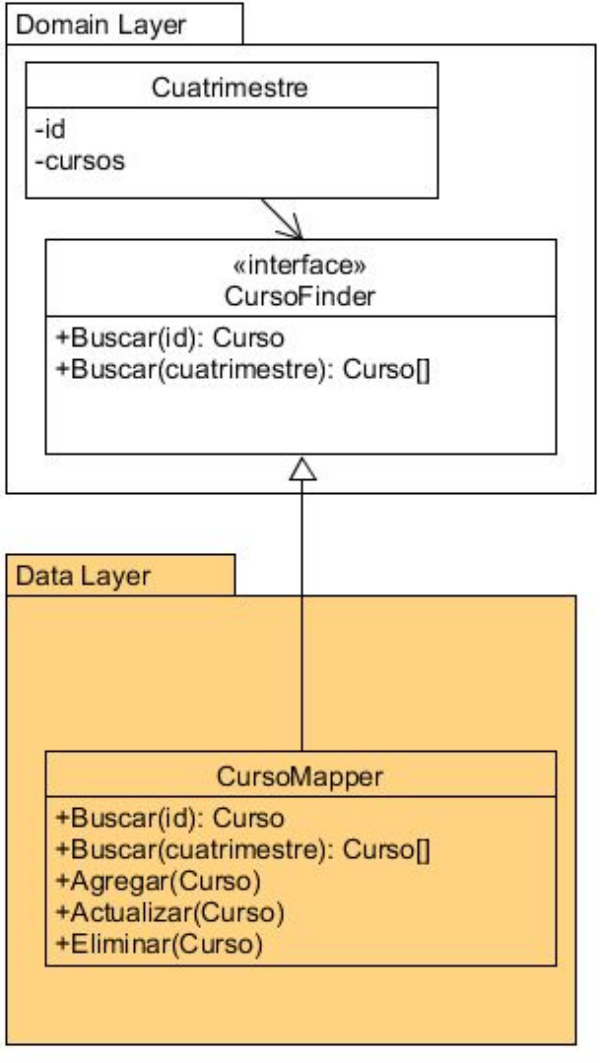
Objetos de dominio pueden necesitar cargar otros objetos de dominio

Usar **Separated Interface**

¿Cómo hacen los mappers para leer los fields de los domain objects?

Properties públicas

Reflection



Active Record (AR)

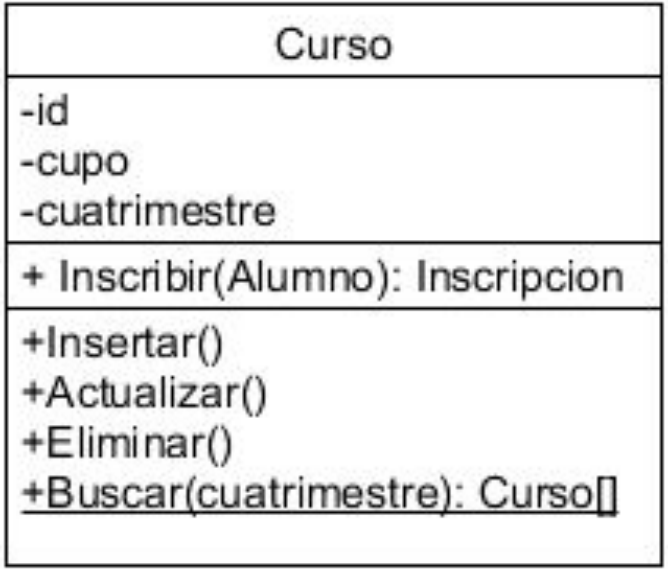
Ubicar la lógica de acceso a datos en el objeto de dominio

Domain Model

Debe haber semejanza entre las clases y la estructura de tablas

Lógica de dominio simple

Se pueden usar finders separados

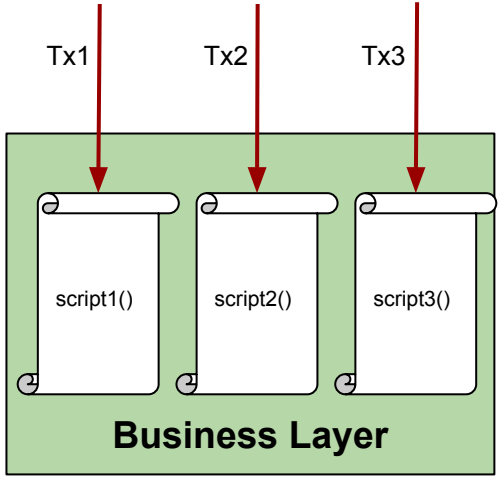


Recap

Organizando la Business Layer

Transaction Script

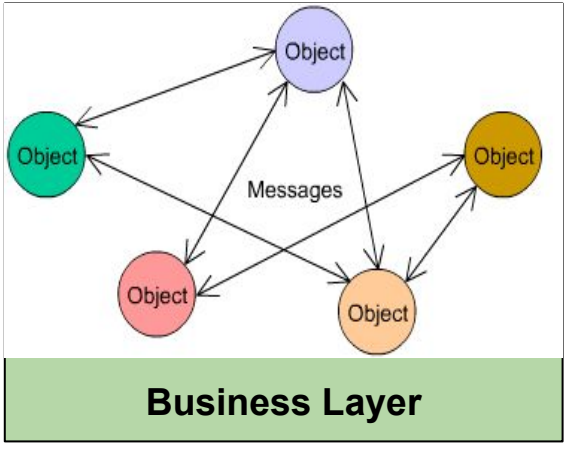
Modelo procedural



1 procedimiento (script) por cada acción (transacción)

Domain Model

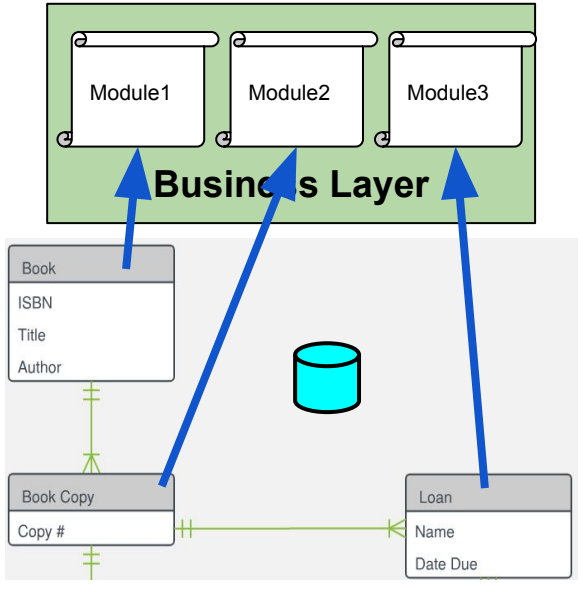
Diseño orientado a Objetos



Objetos y mensajes

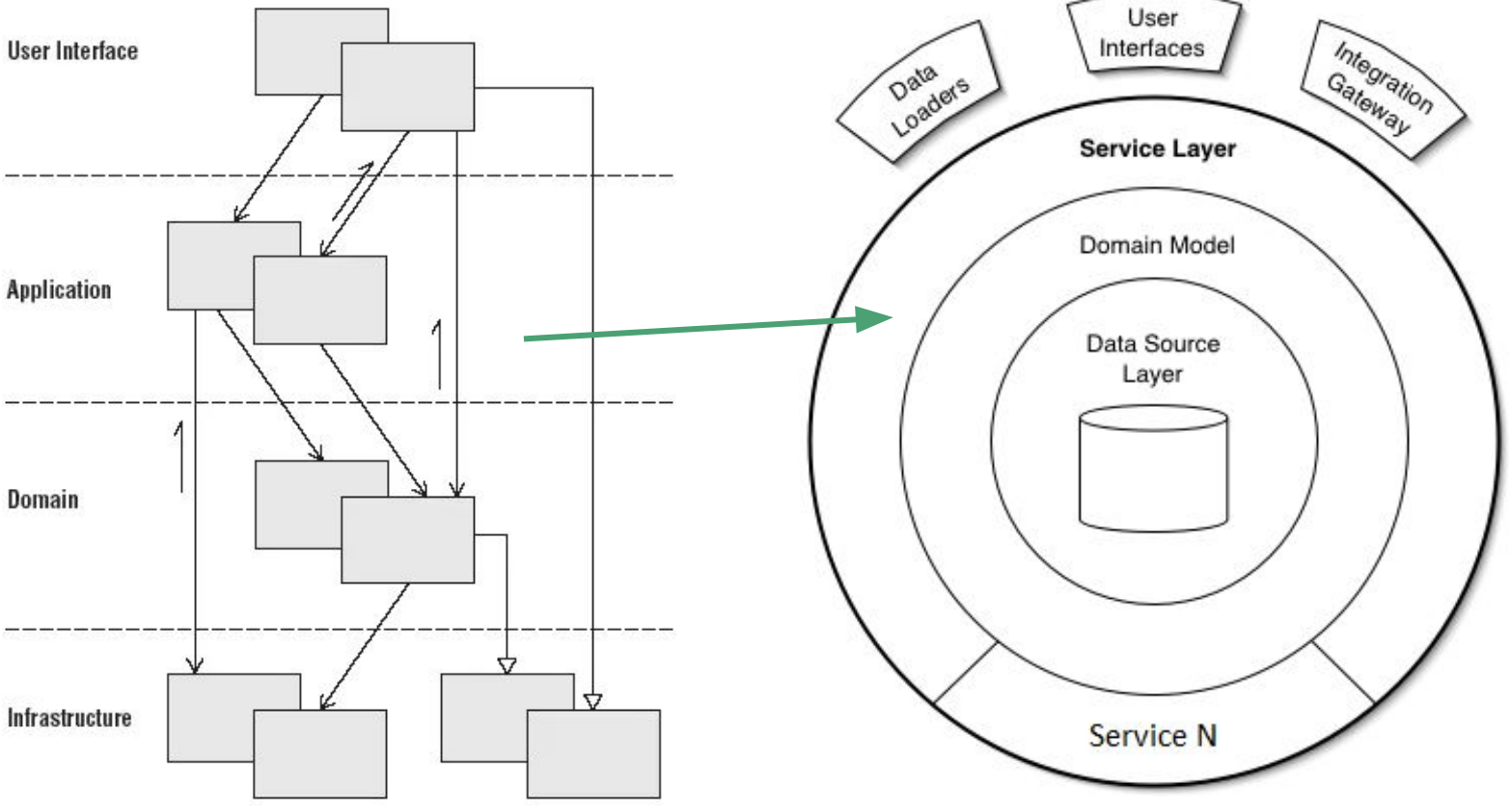
Table Module

Modelo centrado en los datos

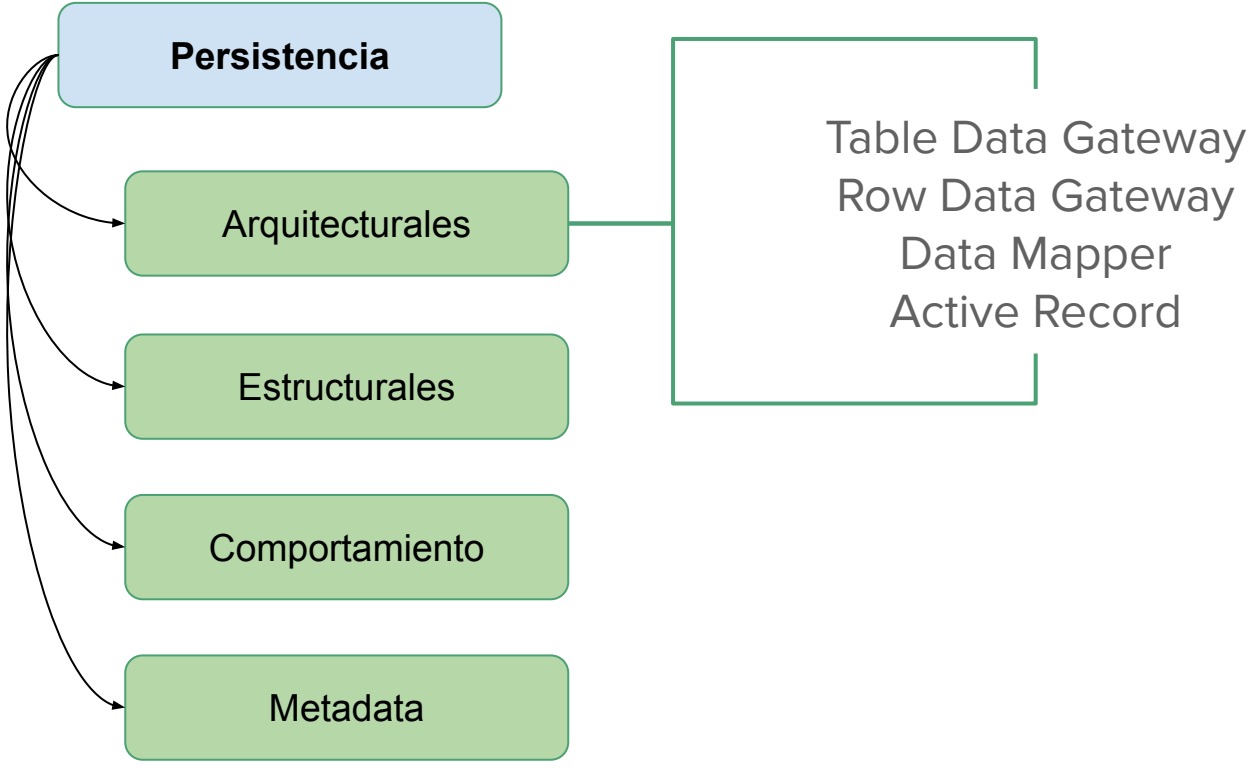


1 módulo/clase por cada entidad de datos (ó tabla)

Capa de Negocio: Service Layer



Capa de Persistencia



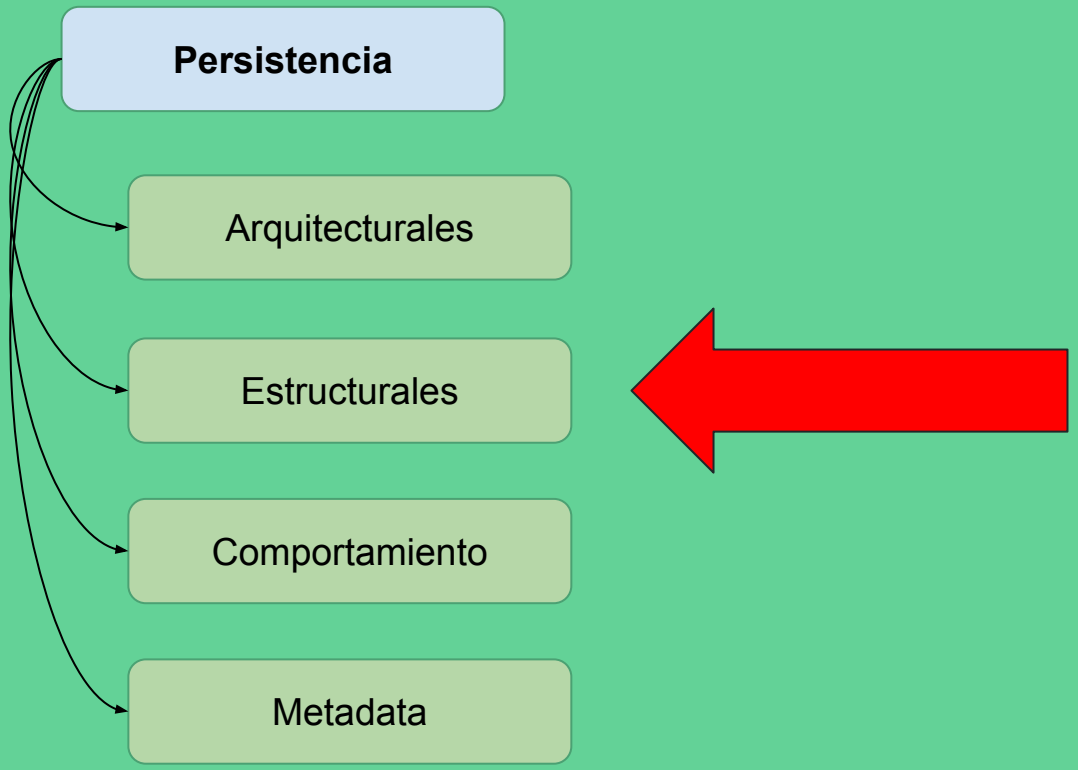
Arquitectura Business Layer + Data Layer

Si en la BL usamos ... la DL la podemos diseñar con ...

Domain Model → Data Mapper ó **Active Record**

Transaction Script → preferentemente **Row Data Gateway**, luego **Table Data Gateway**

Table Module → conviene **Table Data Gateway**


Persistencia
Arquitecturales
Estructurales
Comportamiento
Metadata

Persistencia - Estructurales

Estructurales

¿Cómo mapear una jerarquía de clases a una jerarquía de tablas?

Enfoque Domain Model ➔ DB schema

Identity Field
Foreign-Key Mapping
Association Table Mapping

Jerarquías de asociación (composición/agregación)

Single Table Inheritance
Class Table Inheritance
Concrete Table Inheritance

Jerarquías de herencia

Identity Field

¿Cómo relacionar un objeto en memoria con un registro de una tabla de la BD?

Almacenar en el objeto un campo ID de la BD

Naturaleza: deben ser únicas e inmutables

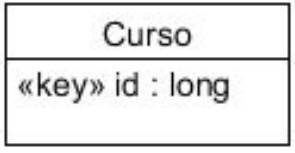
- Meaningful
- Meaningless

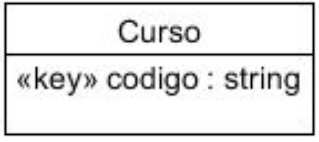
Tipo: Simple / Compuesta

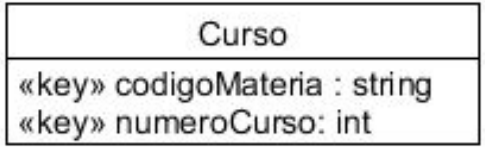
Tipo de Dato

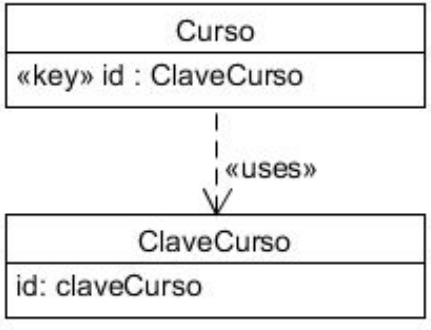
- soportar chequeos igualdad, generación nuevas claves, performance
- Ej.: long, string, date (no recomendado)

Unicidad: table-unique / db-unique keys









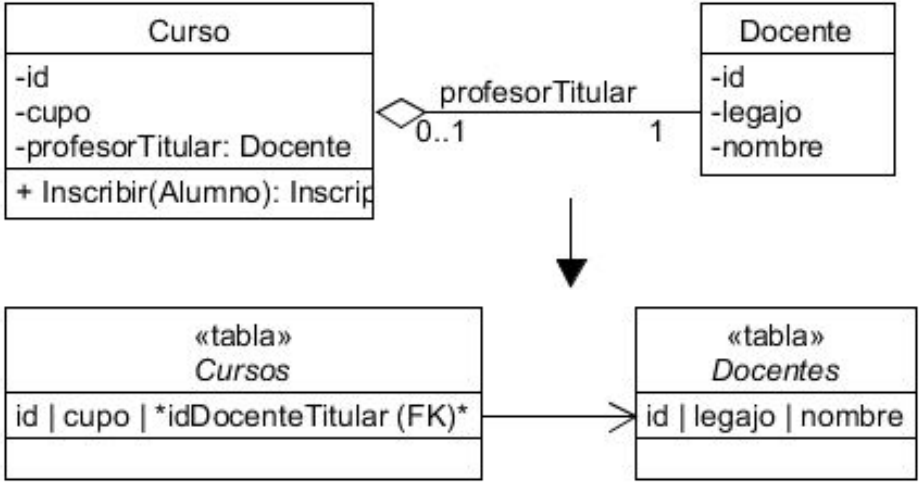
Generación claves simples meaningless

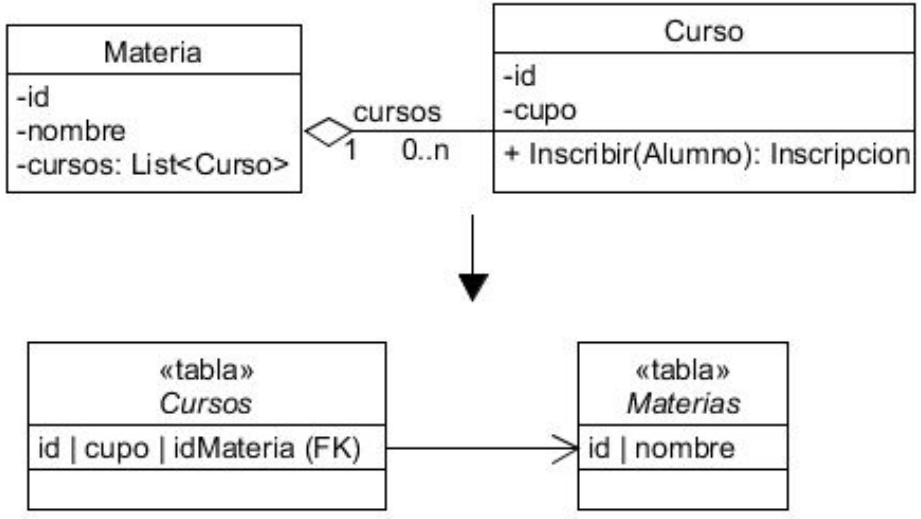
- DB auto-generated field
- DB counter (SELECT SEQUENCE)
- GUIDs
- Key Table (1 fila por tabla)

Foreign-Key Mapping

¿Cómo representar en tablas relaciones 1-1 o 1-* entre objetos?

Utilizar foreign-keys en una de las tablas de la relación

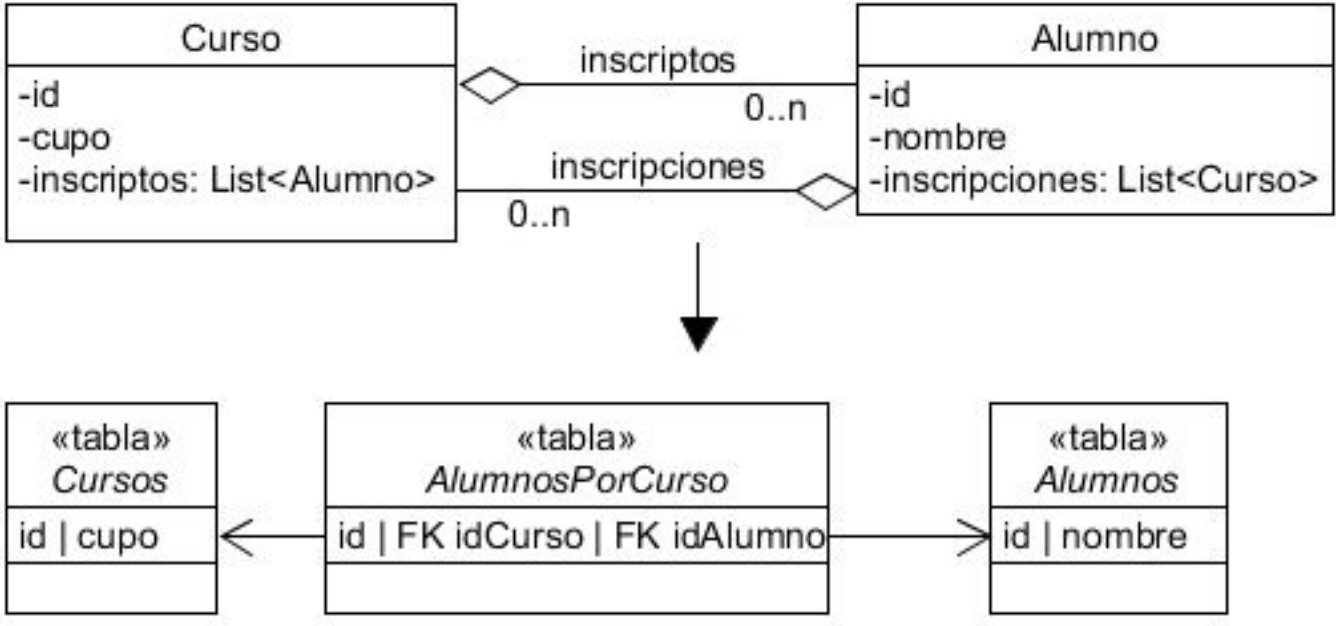




Association-Table Mapping

¿Y mapear “muchos a muchos”?

Crear una nueva tabla de asociación



Mapeando herencia

Single Table Inheritance

1 única tabla para toda la jerarquía

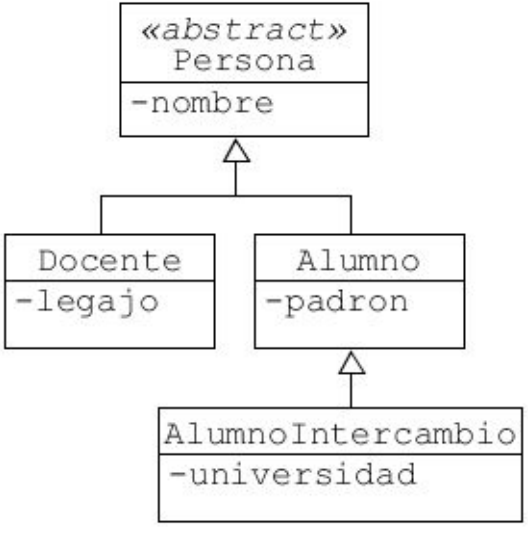
Class Table Inheritance

1 tabla por cada clase de la jerarquía

Concrete Table Inheritance

1 tabla por cada clase concreta de la jerarquía

Single Table Inheritance (STI)



Campos irrelevantes para ciertos tipos

Campo "discriminador"

«tabla»
Personas

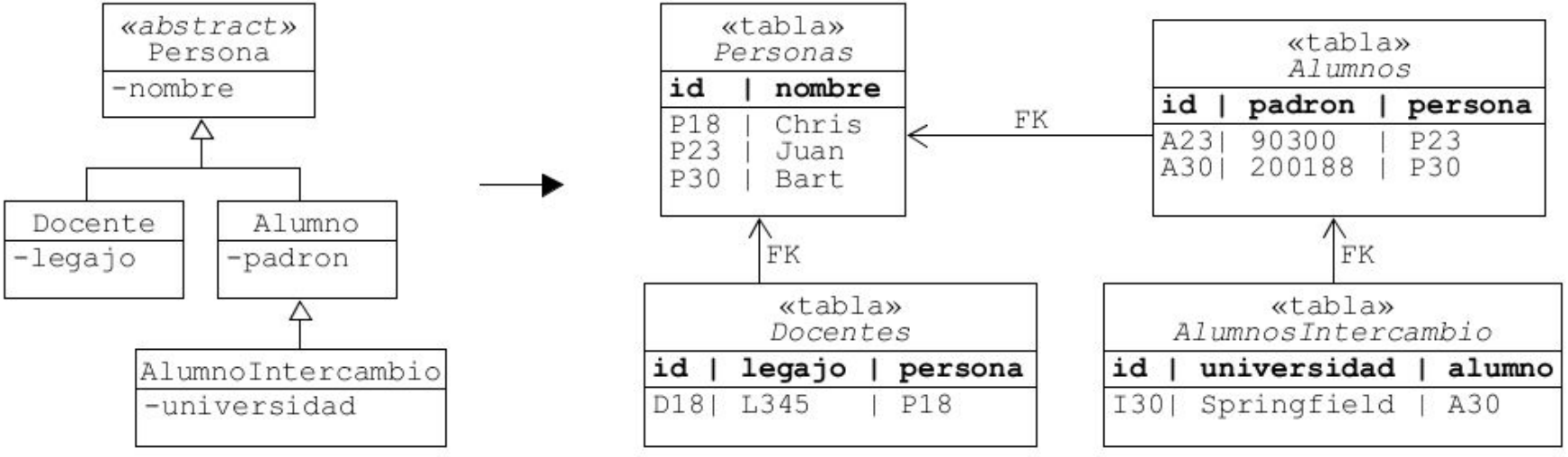
id	nombre	legajo	padron	universidad	tipo
P18	Chris	L345	NULL	NULL	Docente
P23	Juan	NULL	90300	NULL	Alumno
P30	Bart	NULL	200188	Springfield	Interc

Consultas → no hay joins

Refactoring jerarquía de clases → poco impacto en BD

DB locks → muy propenso

Class Table Inheritance (CTI)



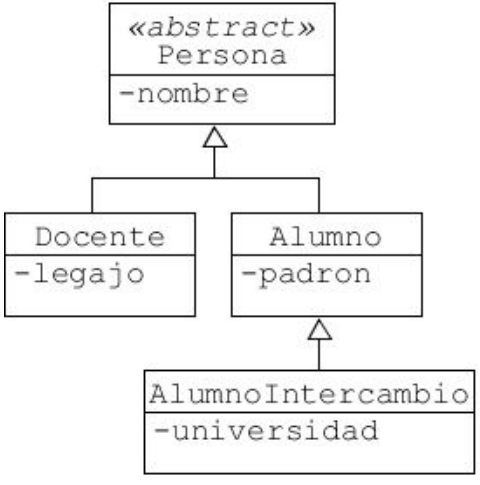
No hay campos irrelevantes

Refactoring jerarquía clases ➜ impacta DB

Consultas ➜ potencialmente varios joins

DB locks ➜ sólo en el supertipo

Concrete Table Inheritance (CoTI)



Cada objeto es mapeado a 1 único registro

«tabla» *Alumnos*

id	nombre	padron
A23	Juan	90300

«tabla» *Docentes*

id	nombre	legajo
D18	Chris	L345

«tabla» *AlumnosIntercambio*

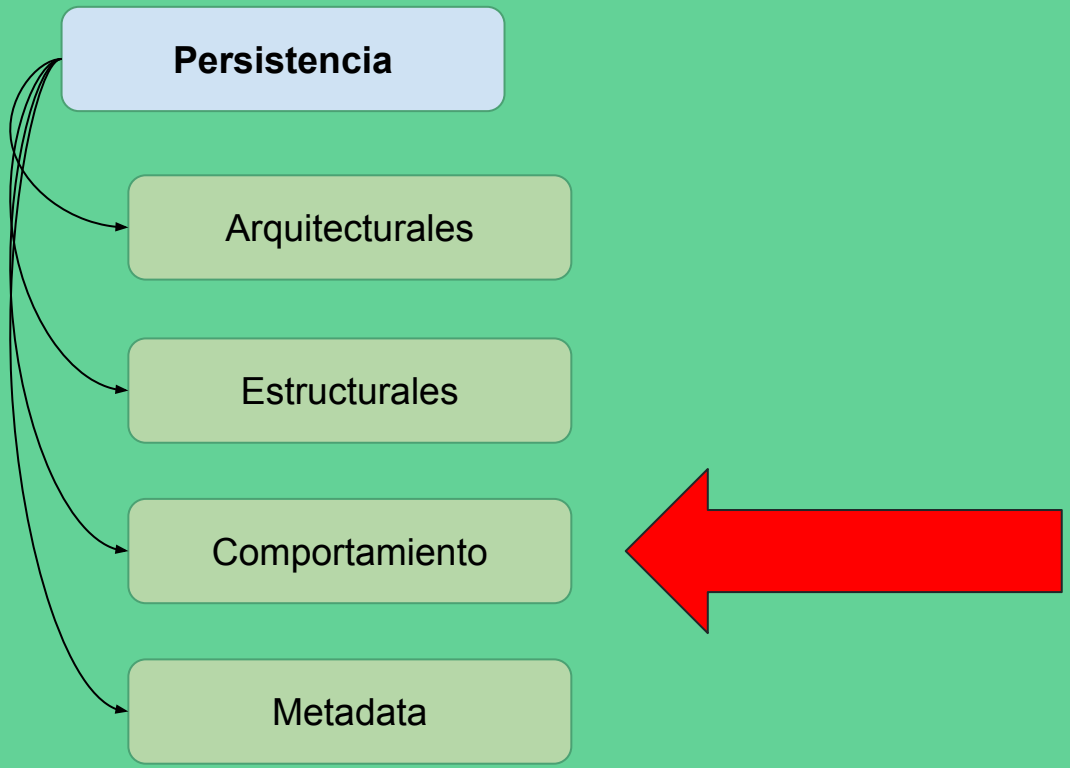
id	nombre	padron	universidad
I30	Bart	200188	Springfield

No hay campos irrelevantes

Consultas ➜ No joins

Refactoring jerarquía clases ➜ impacta DB

DB locks ➜ sólo por clase concreta


Persistencia
Arquitecturales
Estructurales
Comportamiento
Metadata

Persistencia - Comportamiento

Comportamiento

Lazy Load
Unit of Work
Identity Map

¿En qué momento accedemos a la base de datos?

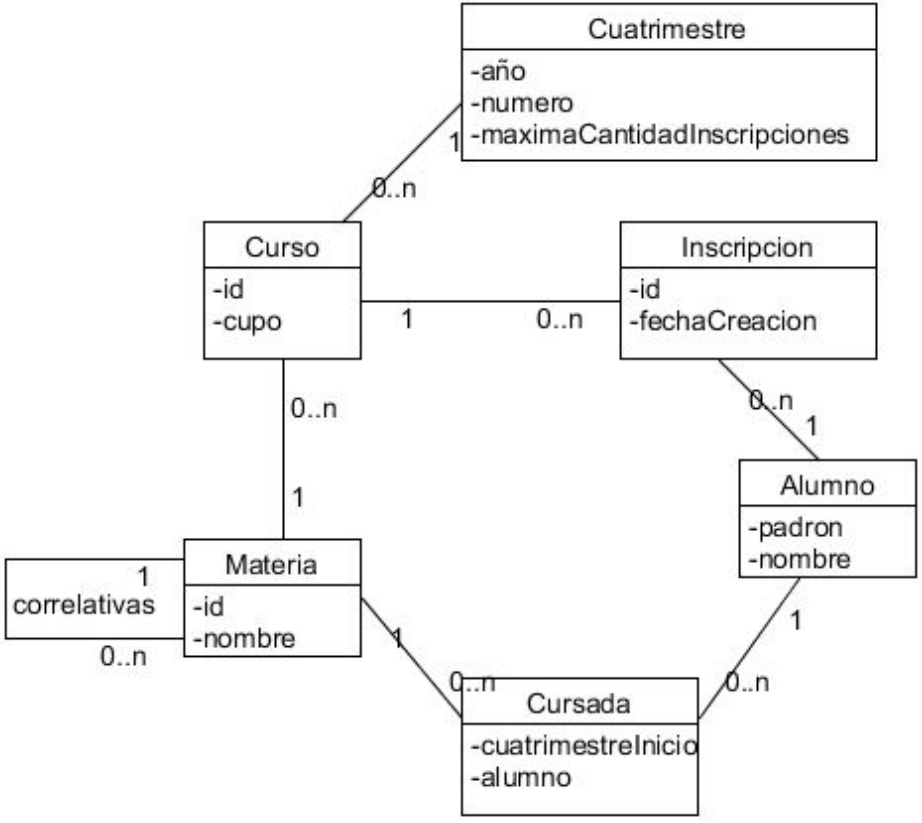
¿Cuándo cargamos en memoria un objeto?

¿Y sus relaciones?

¿Escribimos en la BD inmediatamente luego de crear/eliminar/modificar un objeto?

¿Cuántas veces accederemos a la BD para cargar un objeto dentro de una transacción?

Lazy Load (LL)



Al cargar un objeto, es cómodo cargar también sus objetos relacionados

Problemas:

- Carga desmesurada ➜ impacto performance
- Cargas circulares

Lazy Load: *hacer que el objeto no contenga todos sus datos, pero sepa cómo cargarlos*

Al ser *perezoso* con la carga de datos extra, se gana performance en los casos que esos datos no se necesitan

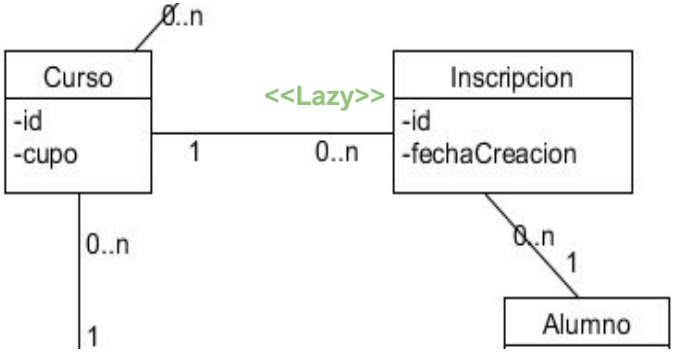
Lazy Load: implementaciones

Lazy Initialization

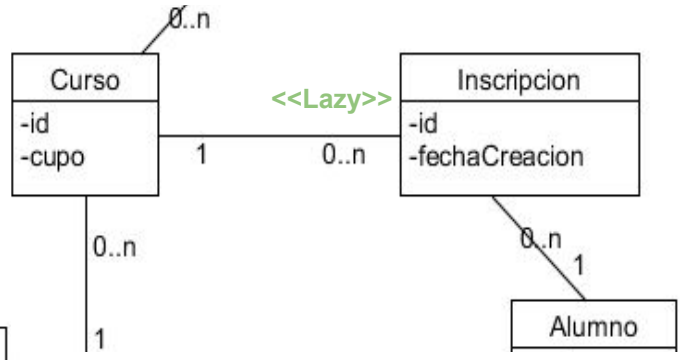
Simple

```
1   public class Curso
2   {
3       private List<Inscripcion> inscriptos;
4
5       public List<Inscripcion> GetInscriptos()
6       {
7           if (inscriptos == null)
8           {
9               inscriptos = Inscripcion.Buscar(id);
10          }
11
12          return inscriptos;
13      }
14  }
```

Acopla objeto de dominio con data layer

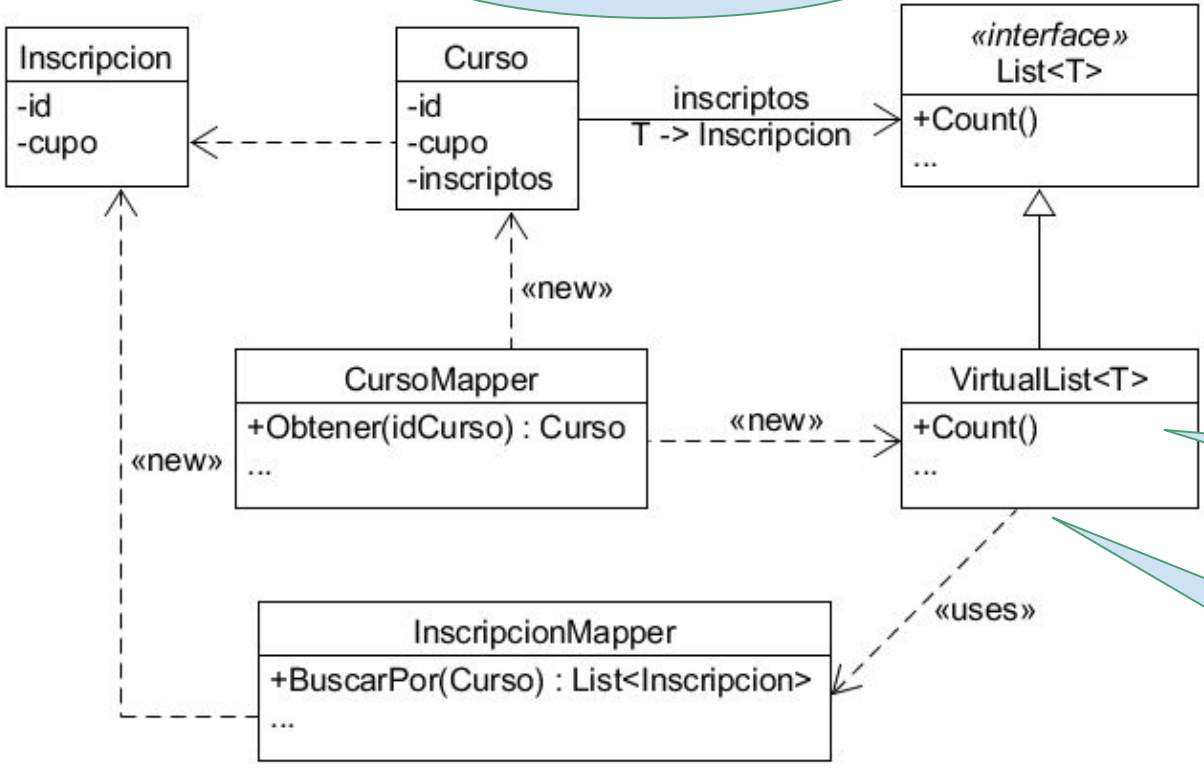


Lazy Load: implementaciones



Virtual Proxy

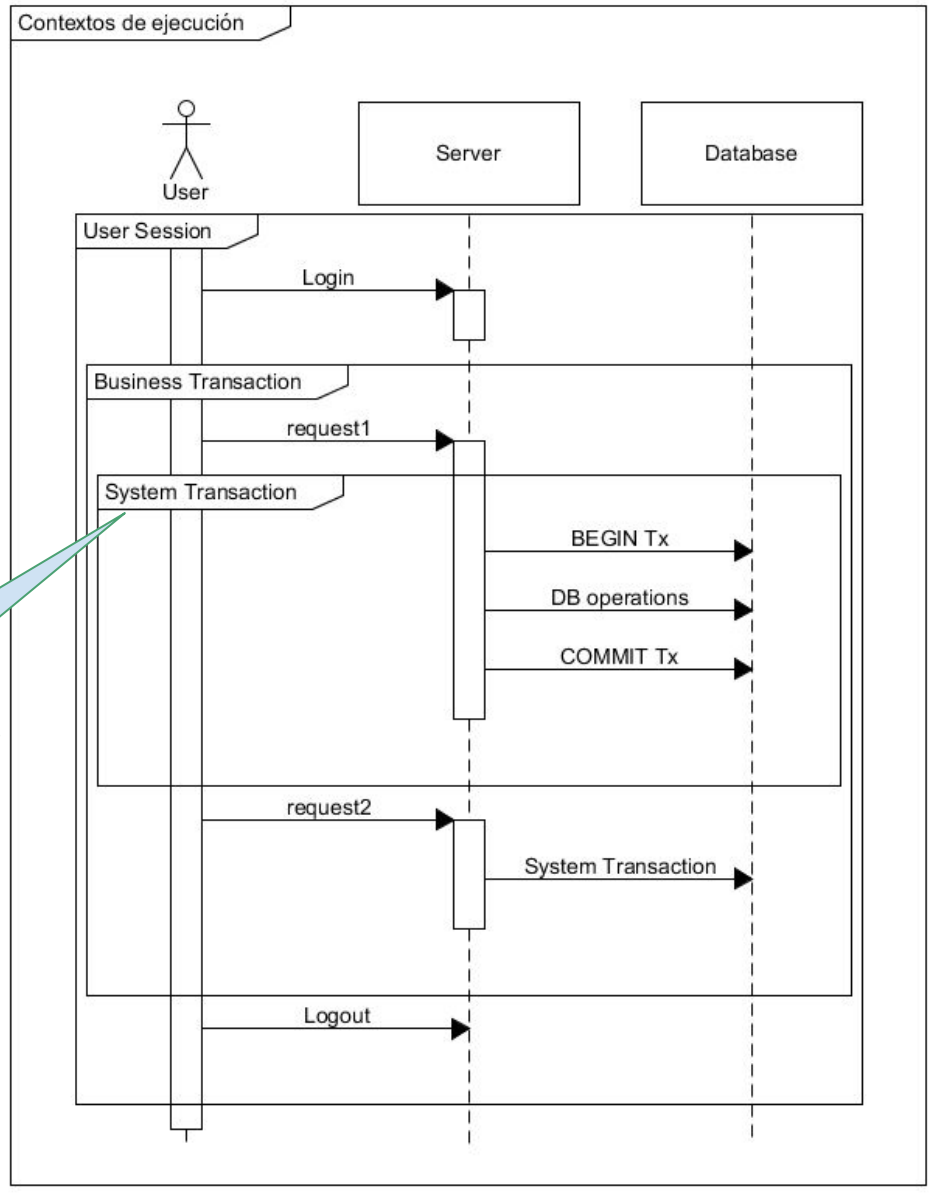
Imita al objeto real, pero no contiene nada



Contextos de Ejecución

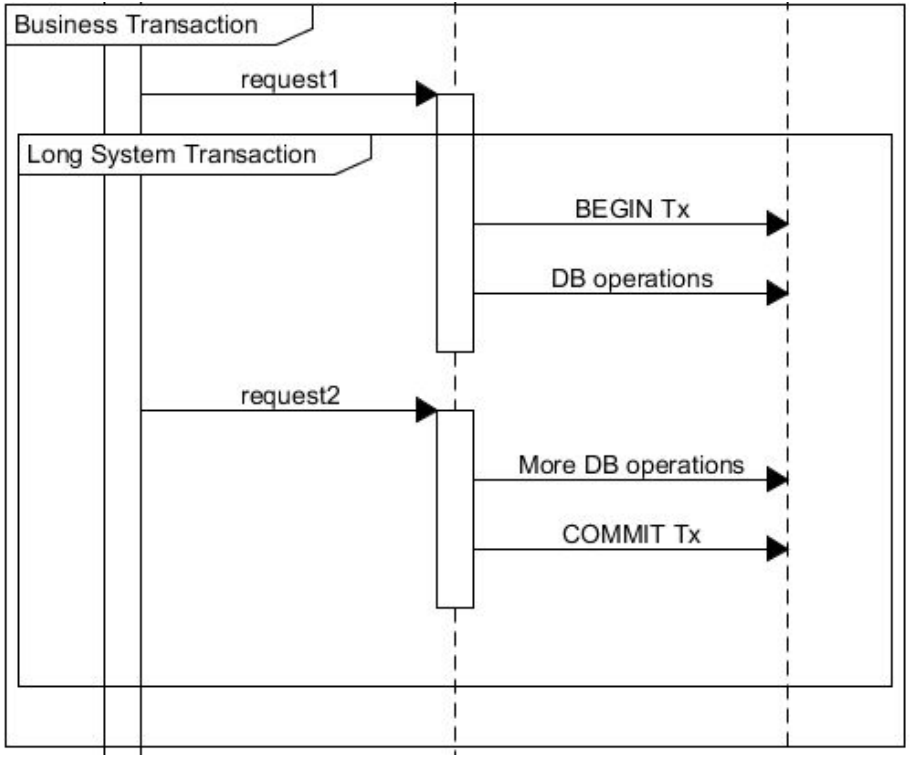
De más amplios a más específicos:

1. User Session
2. Business Transaction
3. Request
4. System Transaction
   a. Request Sys Tx
   b. Long Sys Tx
   c. Late Sys Tx
5. Processes/threads

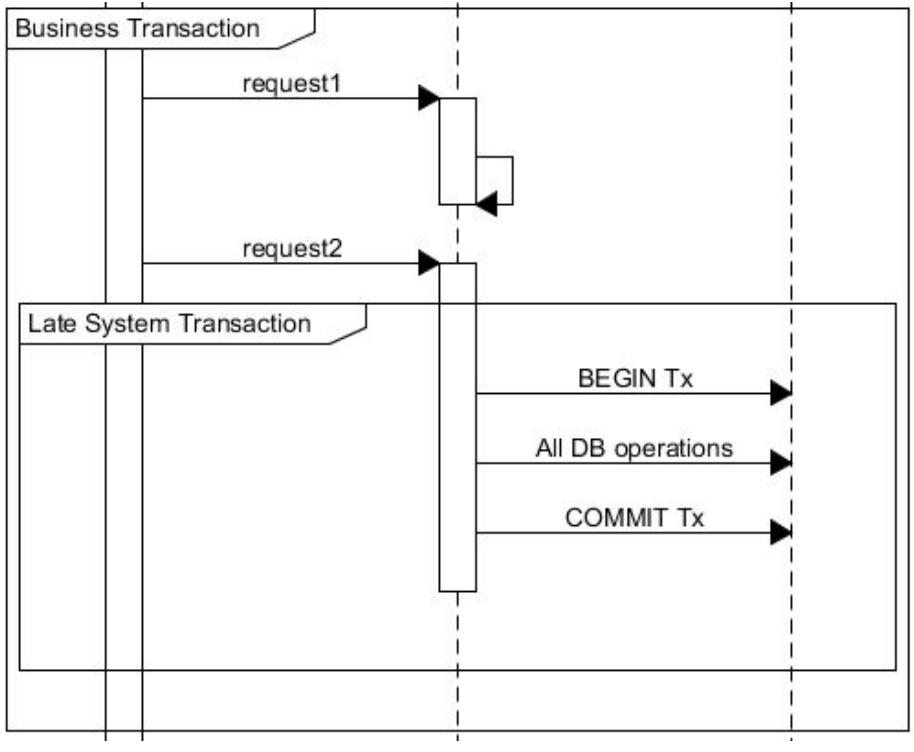


Contextos de Ejecución - Tipos de System Txs

Long System Transaction



Late System Transaction

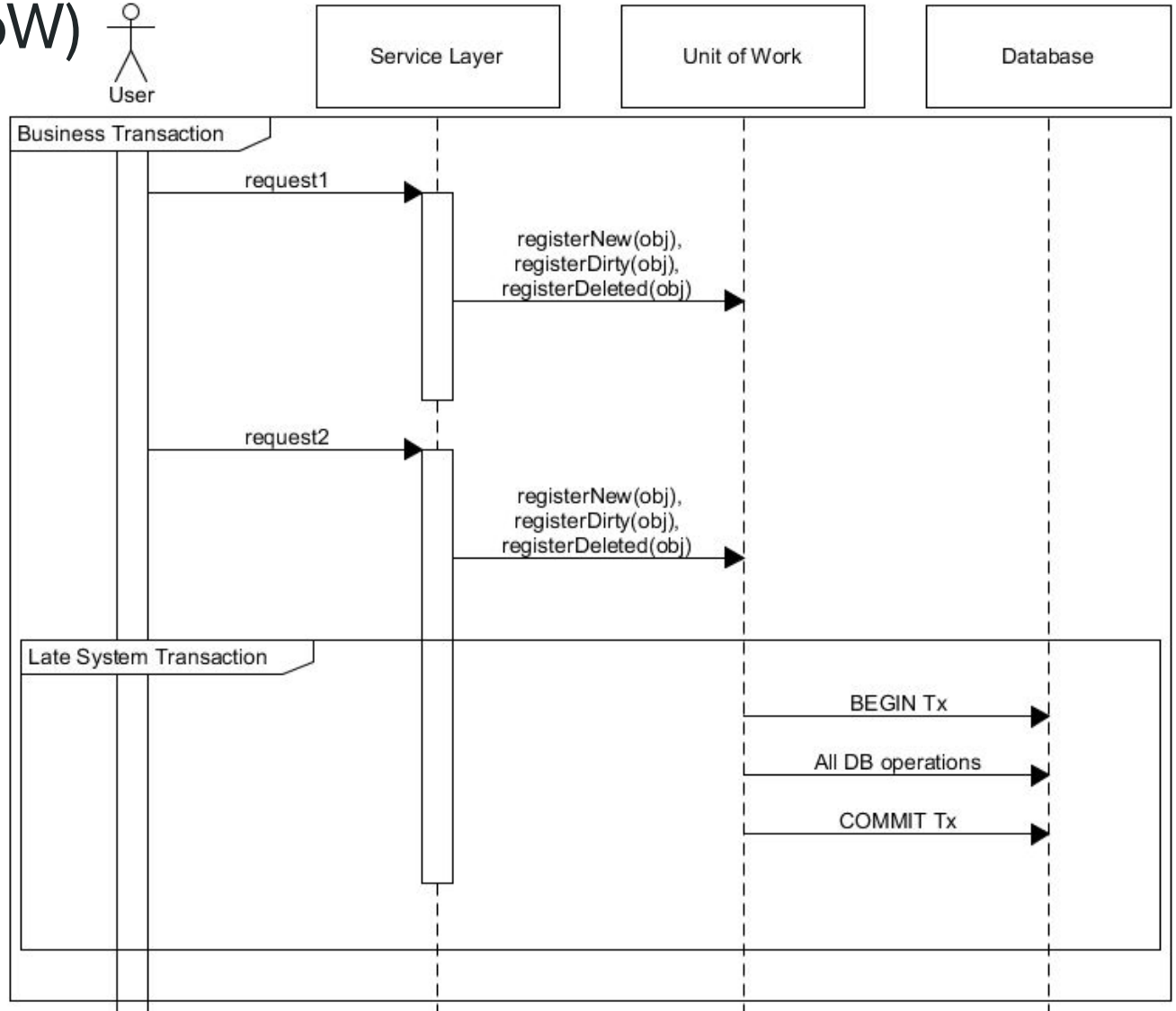


Unit of Work (UoW)

Long System Transactions disminuyen el throughput

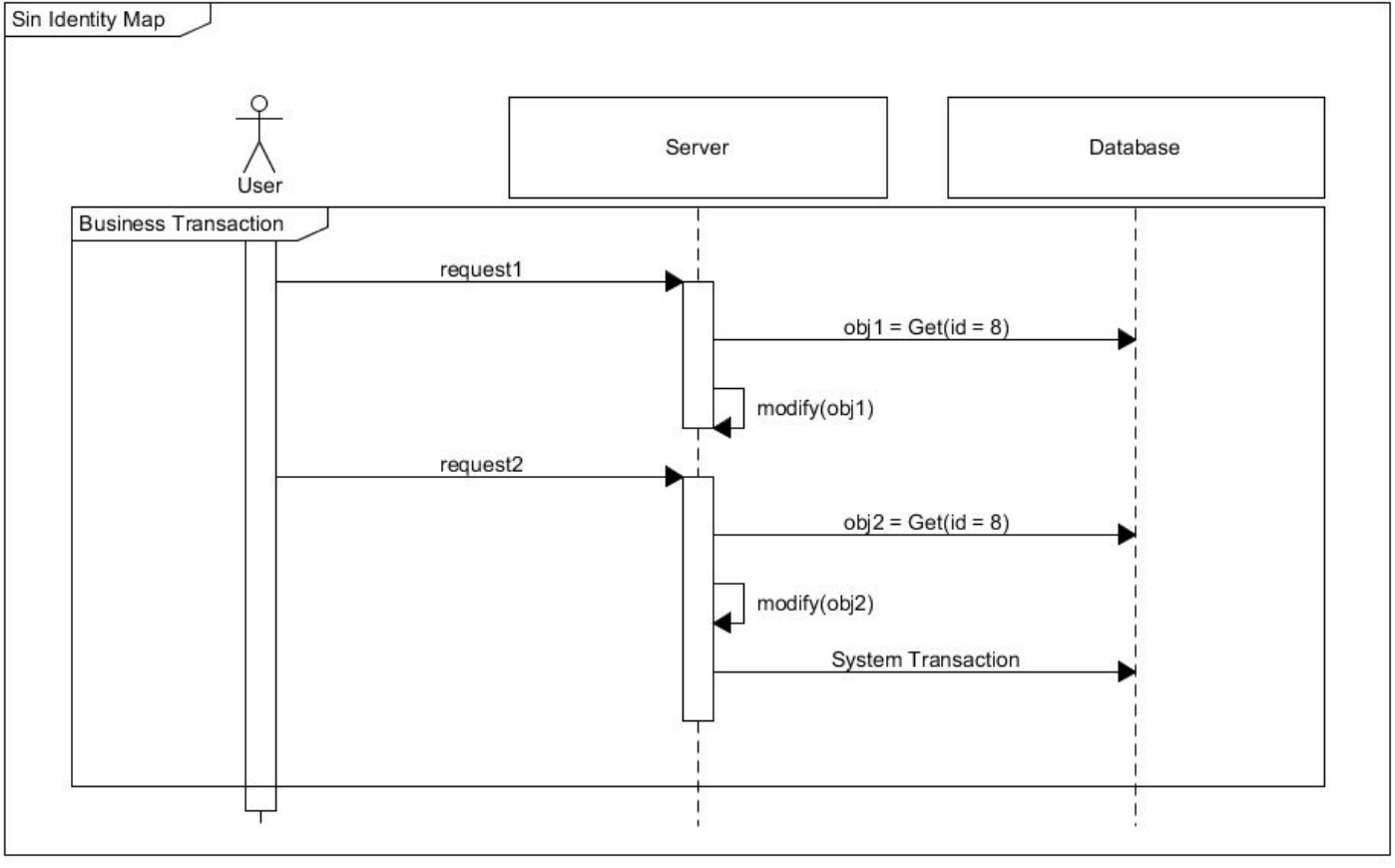
➜ ¿Cómo implementar Late System Transactions?

Unit of Work: *mantiene una lista de objetos afectados dentro de una Business Transaction, y coordina la escritura de cambios en la BD*



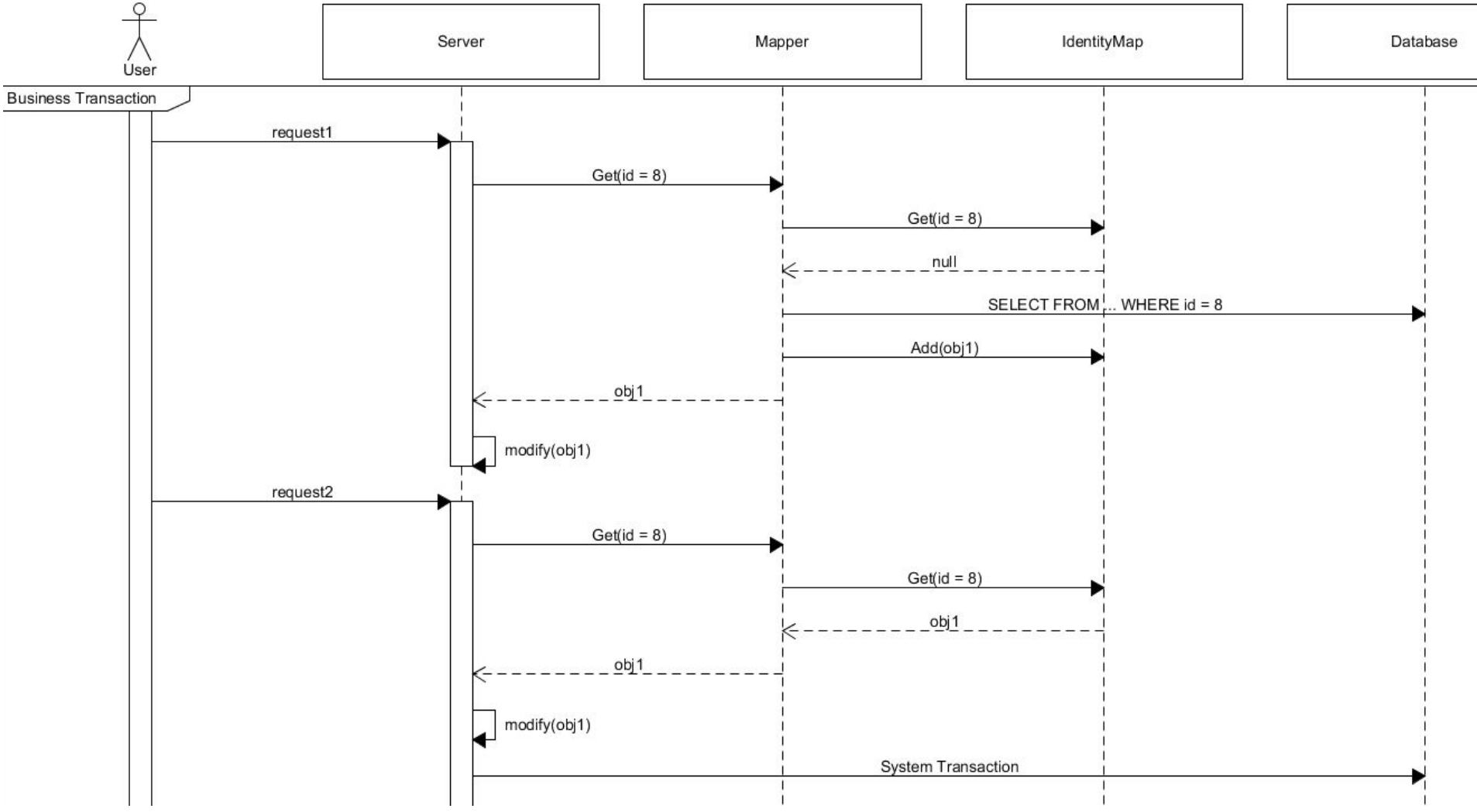
Identity Map (IdMap)

¿Cómo evitar que haya más de 1 objeto en memoria representando 1 único registro de BD?



Identity Map

*IDMap asegura que, **dentro de una Business Transaction**, cada objeto se cargue una sola vez*



En suma...

Hemos visto:

Organización Lógica de Negocio

Transaction Script
Domain Model
Table Module
Service Layer

Persistencia

Arquitecturales

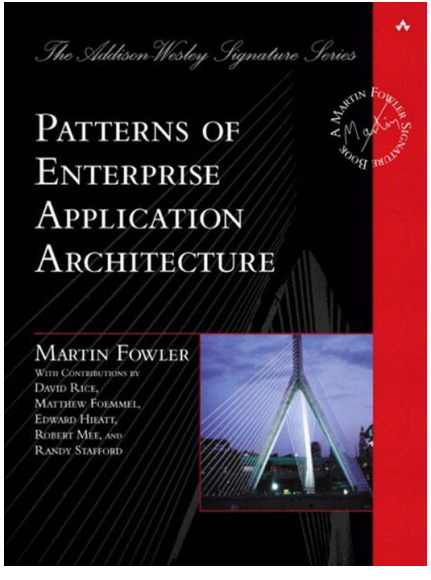
Table Data Gateway
Row Data Gateway
Active Record
Data Mapper

Estructurales

Identity Field
Foreign-Key Mapping
Association Table Mapping
Single Table Inheritance
Class Table Inheritance
Concrete Table Inheritance

Comportamiento

Lazy Load
Unit of Work
Identity Map



¿Consultas?

Feedback

https://goo.gl/forms/NvrORS12kuuBitpE3

Guillermo Rugilo

grugilo@fi.uba.ar

¡Gracias!