PATRONES DE DISEÑO EMPRESARIALES – SEGUNDA PARTE

ELSA ESTEVEZ
UNIVERSIDAD NACIONAL DEL SUR
DEPARTAMENTO DE CIENCIAS E INGENIERIA DE LA COMPUTACION

CONTENIDO

PATRONES DE COMPORTAMIENTO OBJETO-RELACIONAL

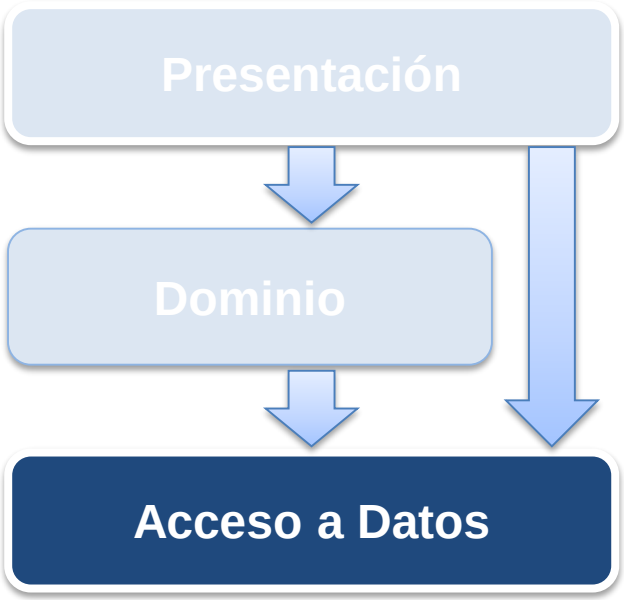
MOTIVACION –

Como hacer para cargar los objetos a memoria y salvarlos en la base de datos?

Si se cargan varios objetos en memoria y se actualizan, se debe llevar registro de cuales se actualizaron para asegurarse que son grabados en la base de datos.

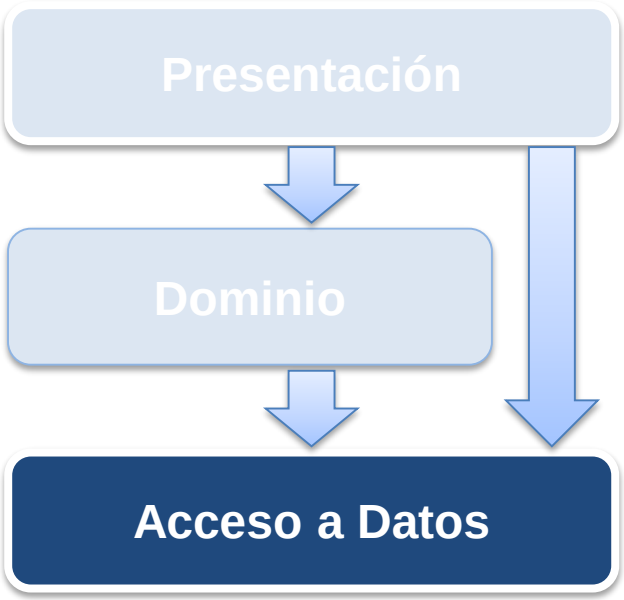
Se necesita asegurar la consitencia de la base de datos.

Si se leen objetos, es necesario saber si pueden o no actualizar simultaneamente – problema de concurrencia.



PATRONES DE COMPORTAMIENTO OBJETO-RELACIONAL

1) Unit of Work
2) Identity Map
3) Lazy Load



1 – UNIT OF WORK

DESCRIPCION - Mantiene una lista de los objetos afectados por una transacción de negocio y coordina la escritura de los cambios y la resolución de problemas de concurrencia.

Con cada cambio en el modelo de objetos, se podría actualizar la base de datos.

Sin embargo esto sería poco eficiente ya que requeriría muchas llamadas a la base de datos.

Además, se necesitaría una transacción abierta para la interacción completa, lo cual no es práctico para transacciones de negocio que implican múltiples pedidos.

UNIT OF WORK – COMO TRABAJA?

- Cuando se comienza a trabajar con la base de datos, se crea un Unit of Work
- Unit of Work mantiene registro de todos los objetos creados / modificados / eliminados durante la transacción de negocio.
- Al finalizar la transacción de negocio, Unit of Work decide qué hacer:
  - abrir una transacción de base de datos
  - realizar chequeos de concurrencia (optimista)
  - escribir cambios a la base de datos en el orden apropiado
  - cerrar la transacción de base de datos

UNIT OF WORK – USO

- Ya viene implementado en muchos ORMs:
  - Hibernate → Session
  - Entity Framework → Context
  - etc.

- Construirlo desde cero es factible, aunque requeriría bastante esfuerzo.

- Muchas veces se encapsula (wrappea) el Unit of Work provisto por el ORM
  - Para evitar tener referencias al ORM en la capa de negocios/aplicación
  - Para extenderlo con el fin de agregarle features
    - ✓ manejo de transacciones distribuidas (múltiples bases de datos)
    - ✓ logging
    - ✓ realizar acciones luego del cierre de la transacción de base de datos

2 – IDENTITY MAP

DESCRIPCION - Asegura que cada objeto sea cargado solamente una vez, manteniendo cada objeto cargado en un map. Cuando se referencia a un objeto, lo busca a través del map.

Si no se tiene cuidado, podríamos cargar los datos del mismo registro de base de datos en dos objetos diferentes.

En ese caso podríamos tener problemas de inconsistencia de información y de performance.

IDENTITY MAP – COMO TRABAJA?

- Mantiene registro de todos los objetos que han sido leídos desde la base de datos en una única transacción de negocio.
- Cuando se quiere obtener un objeto, primero se busca en el map por si ya está. Si no está, se busca en la base de datos y se lo incorpora al map.
- Si se utiliza el concepto de Unit of Work, entonces el Unit of Work es el mejor lugar para el Identity Map.
- También sirve como caché de los objetos obtenidos en una transacción de negocio.

3 – LAZY LOAD

DESCRIPCION - Un objeto que no contiene todos los datos que se necesitan, pero sabe cómo obtenerlos.

Para cargar datos de un base de datos es recomendable diseñar las cosas tal que cuando se carga un objeto de interés, se carguen también los objetos relacionados a él. Esto simplificaría las tareas de un desarrollador, ya que no debe cargar todos los objetos explícitamente.

Sin embargo, también se podría tener el caso de que la carga de un objeto provoque cargar una gran cantidad de objetos relacionados, dañando la performance cuando sólo se necesitan unos pocos objetos.

LAZY LOAD – COMO TRABAJA?

- Lazy Load interrumpe el proceso de carga de objetos relacionados al objeto que se quiere recuperar, dejando una marca en la estructura de objetos tal que si el dato es necesario, pueda ser cargado solamente cuando sea usado.

- Alternativas de implementación:
  - lazy initialization
  - virtual proxy
  - value holder
  - ghost

LAZY LOAD – USO

- actualmente lo proveen por defecto muchos ORMs
- se debe balancear la información que se recupera en un acceso versus la cantidad de accesos que se requerirán para recuperar la información necesaria
- principalmente se utiliza para colecciones de objetos relacionados.
- cuidado con el problema del N+1!

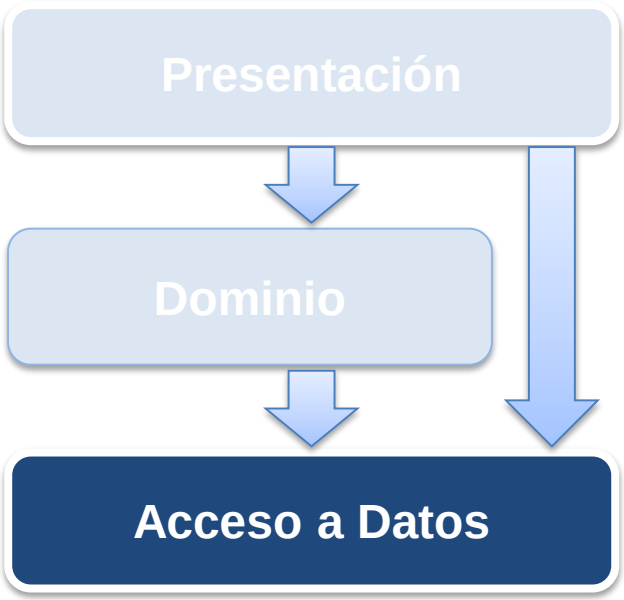
CONTENIDO

PATRONES DE MAPEO ESTRUCTURAL OBJETO-RELACIONAL

MOTIVACION –

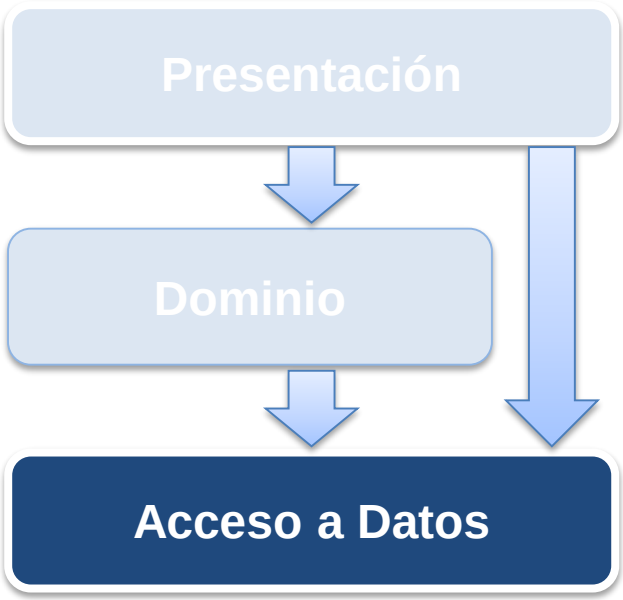
Cuando se habla de mapeo objeto relacional (ORM) se trata de como mapear objetos en memoria con las tablas de una base de datos relacional.

El tema central es el enfoque diferente para mapear links entre la orientación a objetos y las bases de datos relacionales.



PATRONES DE MAPEO ESTRUCTURAL OBJETO-RELACIONAL

1) Identity Field
2) Foreign Key Mapping
3) Association Table Mapping
4) Dependent Mapping
5) Embedded Value
6) Serialized LOB (Large Object)
7) Single Table Inheritance
8) Class Table Inheritance
9) Concrete Table Inheritance



1 – IDENTITY FIELD

DESCRIPCION - Guarda el campo ID de la base de datos en el objeto para mantener la identidad entre un objeto en memoria y la fila de la base de datos.

Para que funcionen, las claves deben ser únicas… pero para que funcionen bien, también deben ser inmutables.

IDENTITY FIELD – USOS 1

- Claves con o sin sentido
  - Meaningful key – Por ejemplo, el número de documento de una persona
  - Meaningless key – Número aleatorio que no sería utilizado por humanos
- Claves simples o compuestas
  - Clave simple - solamente usa un campo de la base de datos
  - Clave compuesta - usa más de un campo de la base de datos
- Alcance de unicidad de las claves
  - Únicas por tabla
  - Únicas por base de datos
- Tipo y tamaño
  - las operaciones más frecuentes son: igualdad y siguiente clave
  - pueden tener efecto en la performance e índices de las bases de datos
  - generalmente se utilizan: long, GUID (Globally Unique Id) e int

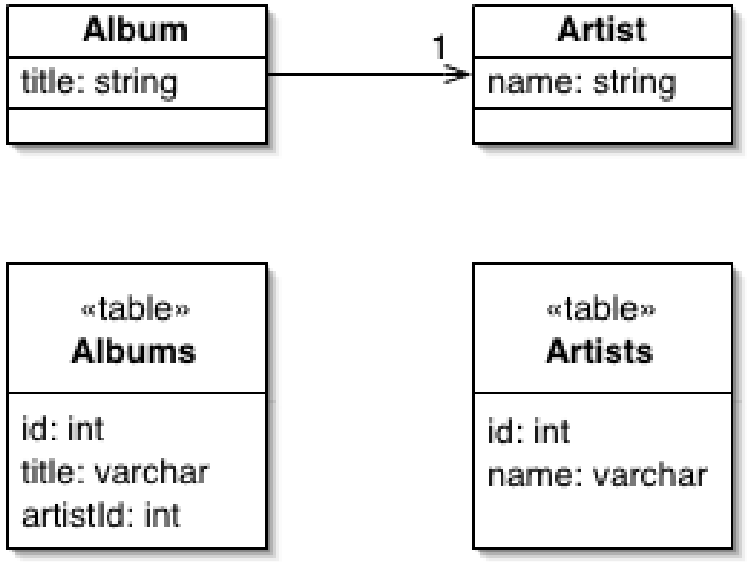
IDENTITY FIELD – USOS 2

- Representar el Identity Field en un objeto
  - la forma más simple de Identity Field es un campo que concuerde con el tipo de la clave en la base de datos
  - para las claves compuestas, generalmente se utiliza una clase que las representa
  - muchas veces se factoriza la definición de la clave (junto con la operación de igualdad) en una clase de entidad base y ese comportamiento es heredado por todas las entidades del sistema

- Obtener una nueva clave
  - delegar en la base de datos y que la auto-genere
  - usar un GUID
  - generar propias (max function, tabla de claves separada)

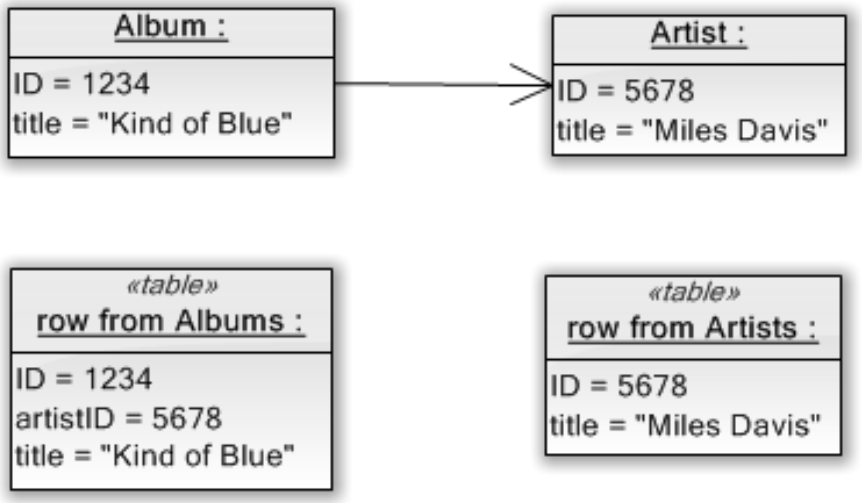
2 – FOREIGN KEY MAPPING

DESCRIPCION - Mapea una asociación entre objetos a una clave foránea entre tablas.

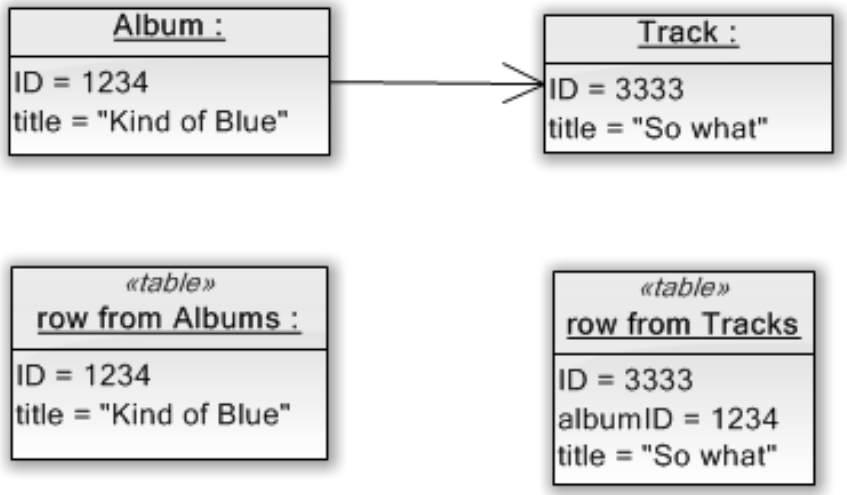


FOREIGN KEY MAPPING – COMO TRABAJA?

Mapeando una referencia



Mapeando una colección

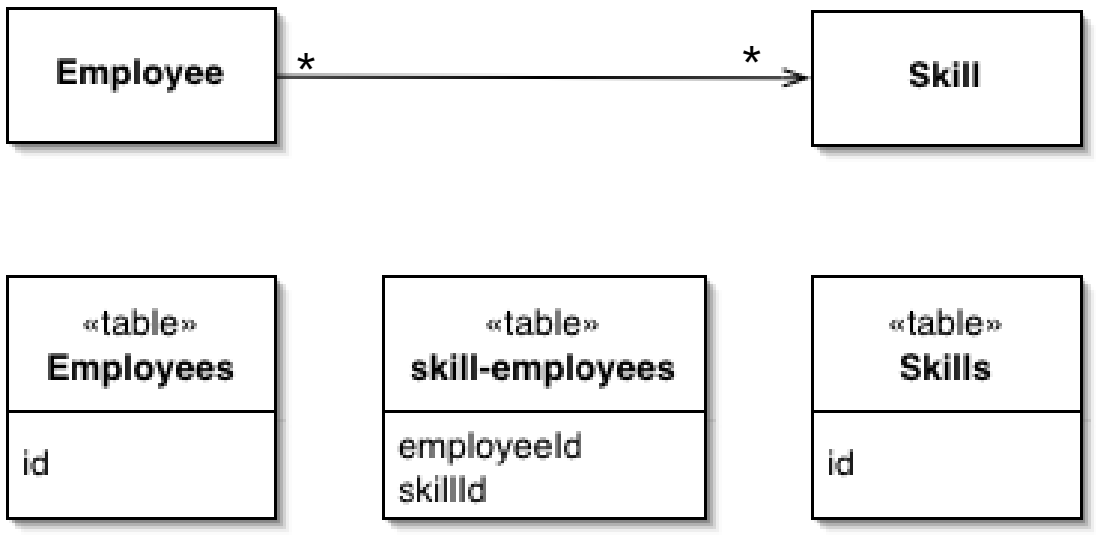


3 – ASSOCIATION TABLE MAPPING

DESCRIPCION - Guarda una asociación en una tabla con claves foráneas a las tablas que están relacionadas por la asociación.

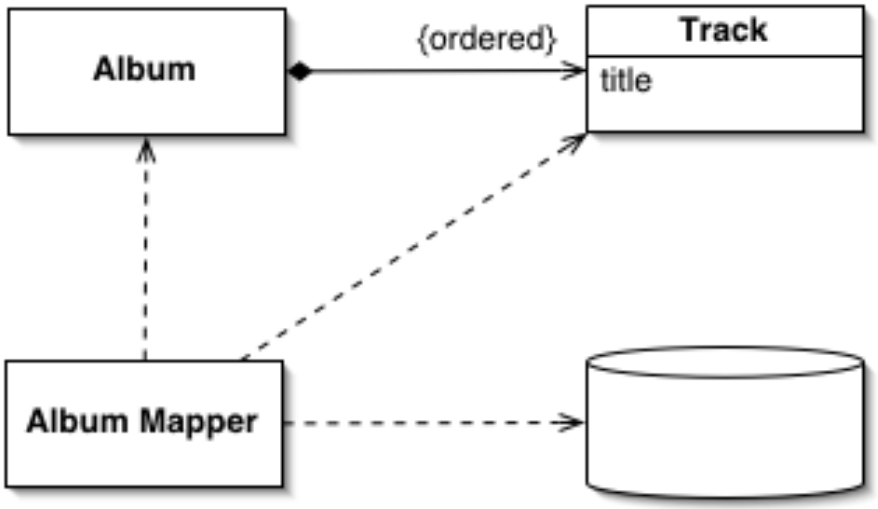
El caso de las relaciones muchos a muchos no puede solucionarse con un patrón Foreign Key Mapping, en este caso, se recomienda usar el Association Table Mapping.

ASSOCIATION TABLE MAPPING – COMO TRABAJA?



4 – DEPENDENT MAPPING

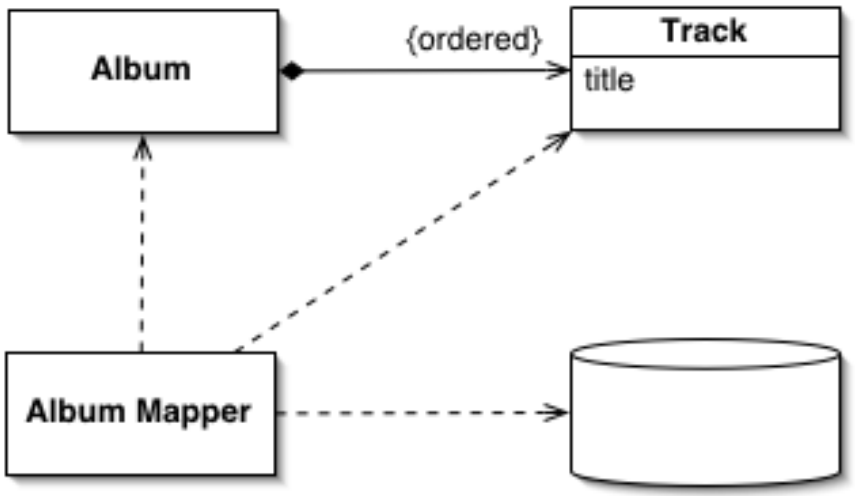
DESCRIPCION - Una clase tiene que realizar el mapeo de base de datos para una clase hija.



DEPENDENT MAPPING – COMO TRABAJA?

El objeto dependiente:

- Sólo tiene un owner
- No puede haber referencias a él desde otro objeto que no sea su owner
- No tiene un Identity Field
- No es almacedo en el Identity Map
- No puede ser buscado directamente
- Sólo es accedido desde su owner
- Generalmente es un Value Object



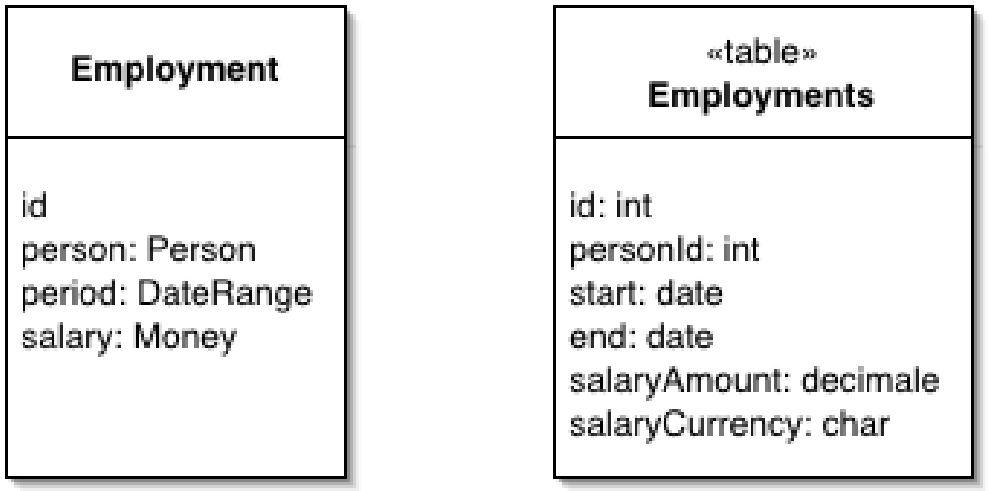
5 – EMBEDDED VALUE

DESCRIPCION - Mapea un objeto en varios campos de la tabla de otro objeto.

Muchos pequeños objetos tienen sentido en un sistema orientado a objetos. Sin embargo, no tienen sentido como tablas en una base de datos. Por ejemplo, objetos relacionados con tipo de moneda y rango de fechas.

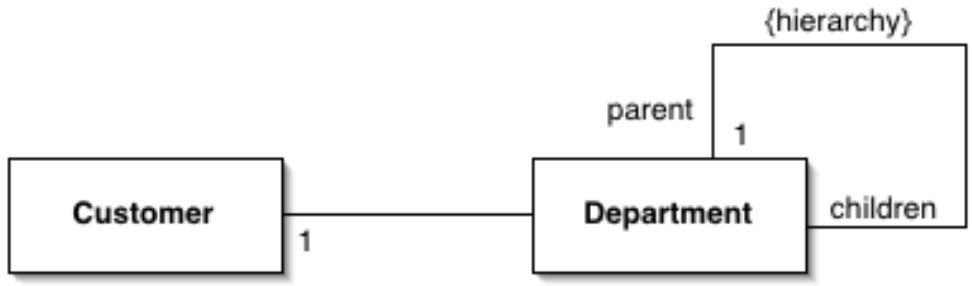
EMBEDDED VALUE – COMO TRABAJA?

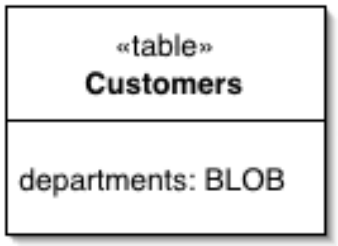
Se mapean los valores de un objeto (e.g. period) a campos (e.g. start, end) en el registro del propietario del objeto.



6 – SERIALIZED LOB

DESCRIPCION - Guarda un grafo de objetos serializándolos en un único objeto grande (LOB), el cual se almacena en un campo de la BD.





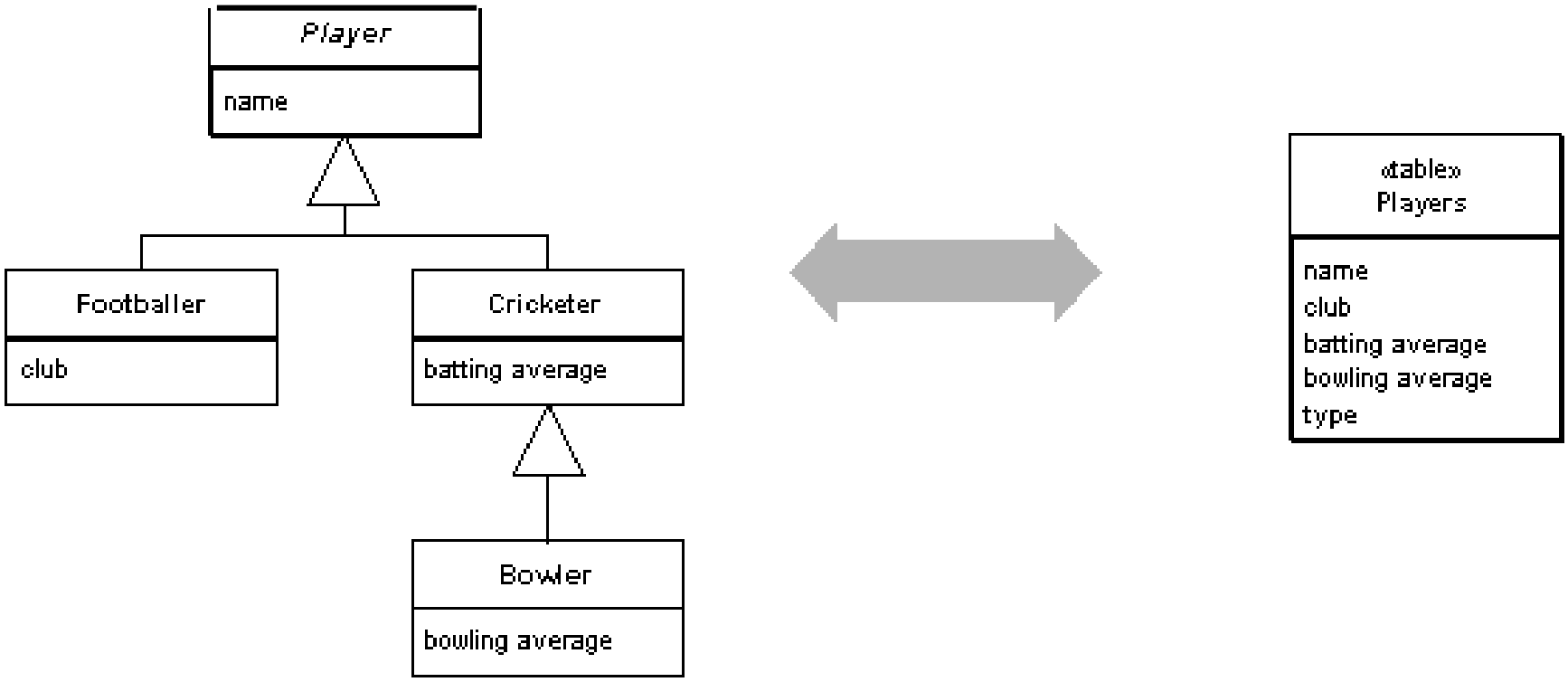
BLOB – Binary LOB

SERIALIZED LOB – COMO TRABAJA?

- Tipos de serialización:
  - BLOB: Binary LOB
  - CLOB: Textual characters LOB
- Se diferencia con Embedded Value en que está orientado estructuras de objetos más complejas.

7 – SINGLE TABLE INHERITANCE

DESCRIPCION - Representa una jerarquía de herencia de clases como una única tabla que tiene columnas para todos los campos de las clases de la jerarquía



SINGLE TABLE INHERITANCE – COMO TRABAJA?

- Existe una tabla que contiene todos los datos para todas las clases de la jerarquía.
- Cuando se carga un objeto a memoria, se necesita conocer qué clase instanciar:
  - Se necesita un campo extra en la tabla que indique qué clase debiera utilizarse para esa fila.
    - ✓ Nombre de la clase
    - ✓ Código discriminador

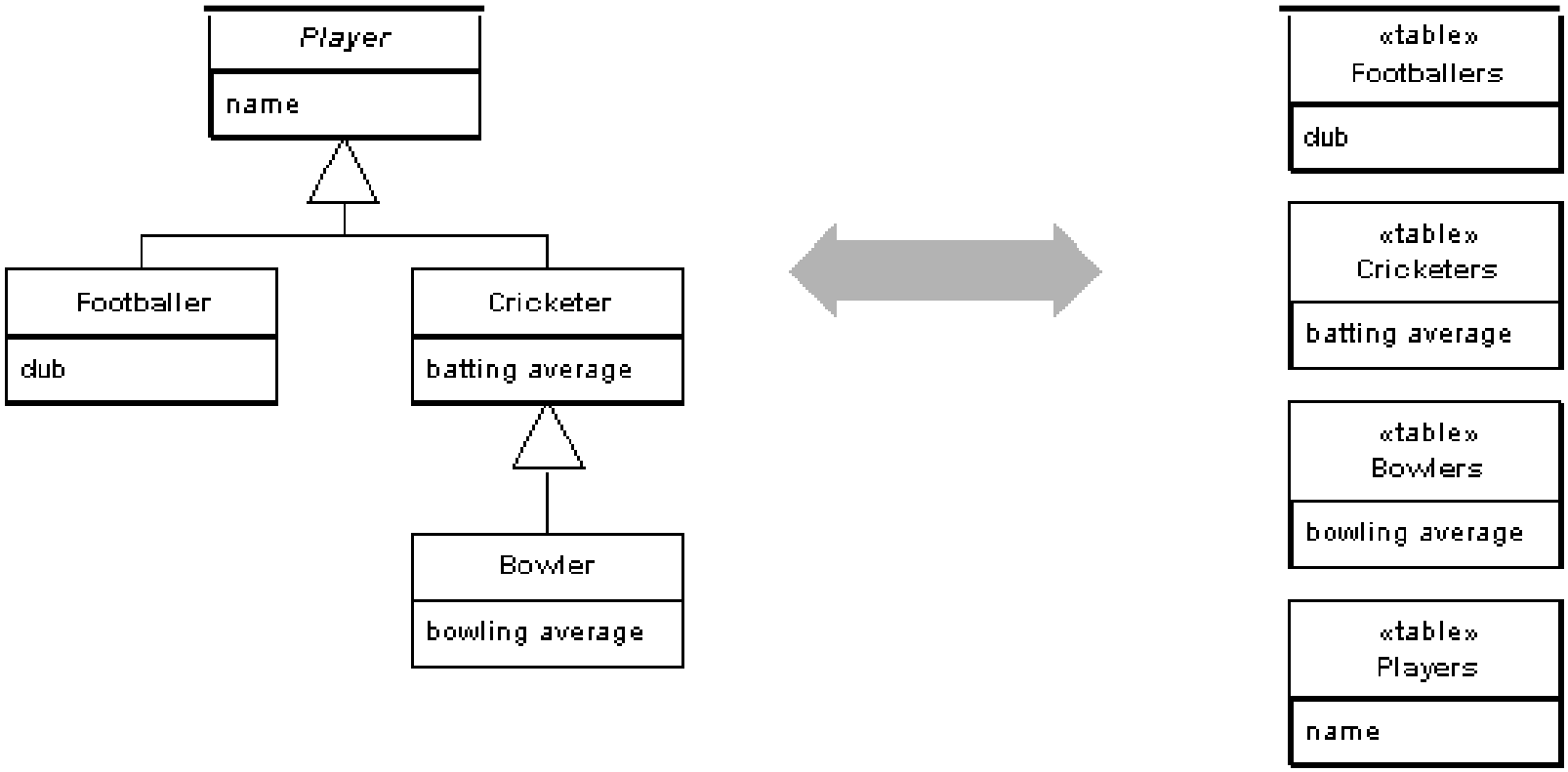
SINGLE TABLE INHERITANCE – USO

- 👍 Habrá una única tabla por la que preocuparse
- 👍 No se realizan joins para recuperar información
- 👍 Cualquier refactoring que eleve/baje campos de la jerarquía no requiere que se cambie la base de datos

- 👎 Los nombres de columnas a veces son relevantes y a veces no, lo cual puede confundir a personas que accedan directamente a la base de datos
- 👎 Las columnas que sólo son utilizadas por algunas subclases conducen a desperdiciar espacio.
- 👎 La tabla podría terminar siendo muy grande, con muchos índices y frecuentes bloqueos, lo cual afectaría la performance.
- 👎 Sólo se tiene un único namespace para los nombres de los campos, con lo cual se debe asegurar que no existan dos campos en el jerarquía con el mismo nombre… o definir una política que resuelva estas cuestiones.

8 – CLASS TABLE INHERITANCE

DESCRIPCION - Representa una jerarquía de herencia de clases con una tabla para cada clase



CLASS TABLE INHERITANCE – COMO TRABAJA?

- Existe una tabla por cada clase en la jerarquía de herencia.
- Los campos en la clase de domino se mapean directamente a los campos de la tabla correspondiente.
- ¿Cómo enlazar las filas en distintas tablas que representan a un objeto concreto de la jerarquía?
  - Usar un valor de clave primaria común a la jerarquía
    - ✓ En la superclase se asegura la unicidad de las claves
  - Cada tabla tiene sus propias claves primarias. Entonces, usar foreign keys en la tabla de la superclase para apuntar al resto de las filas.
- El problema más desafiante es cómo recuperar los datos desde múltiples tablas de manera eficiente – por ejemplo, queries separadas, joins… tienen un costo alto de performance en muchos casos.

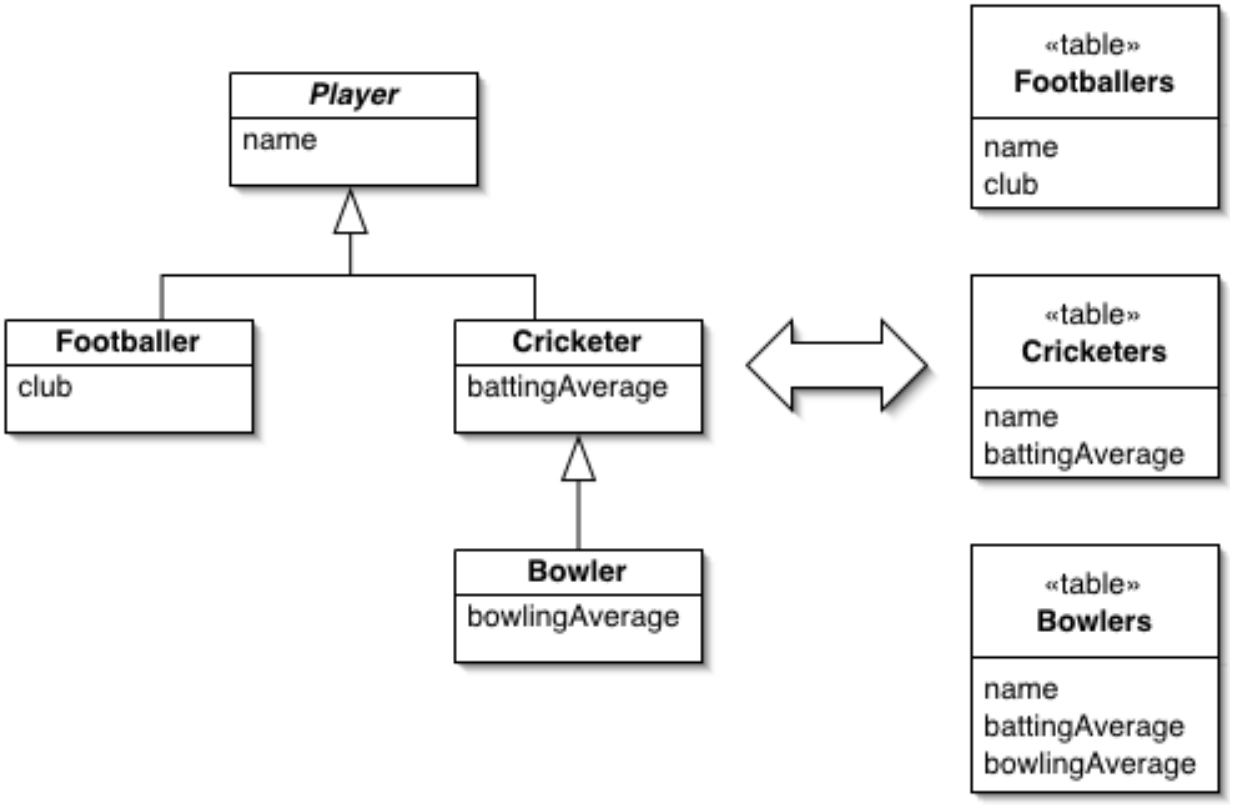
CLASS TABLE INHERITANCE – USO

- Todas las columnas son relevantes para cada fila. Por lo tanto las tablas son fáciles de entender y no desperdician espacio.
- La relación entre el modelo de dominio y la base de datos es directa.

- Se necesita acceder a múltiples tablas para cargar un objeto (un join o múltiples queries, más el armado en memoria)
- Cualquier refactoring que eleve/baje campos de la jerarquía causa cambios en la base de datos
- Las tablas de los supertipos podrían convertirse en un cuello de botella ya que serían accedidas con mucha más frecuencia que las tablas de las hojas de la jerarquía
- La alta normalización puede hacer que sea difícil entender los queries

9 – CONCRETE TABLE INHERITANCE

DESCRIPCION - Representa una jerarquía de herencia de clases con una tabla por clase concreta de la jerarquía



CONCRETE TABLE INHERITANCE – COMO TRABAJA?

- Existe una tabla para cada clase concreta de la jerarquía de herencia
- Cada tabla contiene las columnas para los campos propios de la clase concreta y para los campos de todos sus ancestros. Por lo tanto, los campos de las superclases son duplicados en las tablas de sus subclases concretas
- Se necesitan claves que sean únicas para toda la jerarquía.
- Existe un problema con la integridad referencial, ya que no existe la posibilidad que los clientes de la jerarquía fuercen una referencia a clases no-concretas.
  - Ignorar integridad referencial
  - Utilizar múltiples tablas de enlace, una para cada tabla real en la base de datos

CONCRETE TABLE INHERITANCE – USO

- 👍 Cada tabla es auto-contenida y no tiene campos no relevantes. Esto es favorable cuando las tablas serán utilizadas también por otros sistemas que no estén usando objetos.
- 👍 No se necesitan joins para leer datos.
- 👍 Cada tabla es accedida solamente cuando la clase correspondiente es accedida.

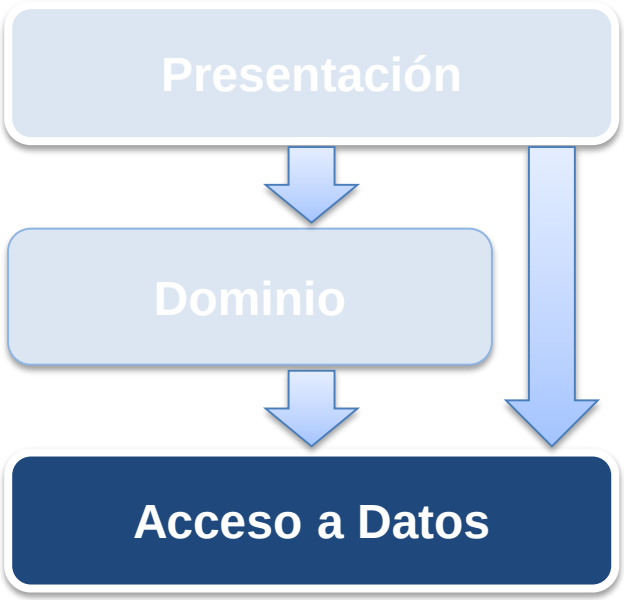
- 👎 Las claves primarias pueden ser difíciles de manejar.
- 👎 No se pueden forzar relaciones de base de datos a clases abstractas.
- 👎 Cualquier refactoring que eleve/baje campos de la jerarquía causa cambios en la base de datos, aunque no tantos como en Class Table Inheritance.
- 👎 Si cambia un campo de una superclase, se deben modificar todas tablas que contengan dicho campo.
- 👎 Una búsqueda sobre la superclase fuerza a chequear todas las tablas.

CONTENIDO

PATRONES DE MAPEO DE METADATOS OBJETO-RELACIONAL

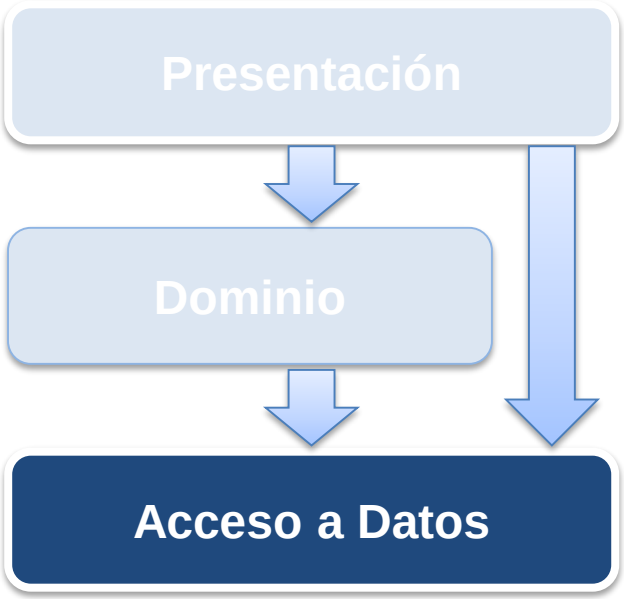
MOTIVACION –

Como facilitar/optimizar el acceso a la base de datos?



PATRONES DE MAPEO DE METADATOS OBJETO-RELACIONAL

1) Repository
2) Metadata Mapping
3) Query Object

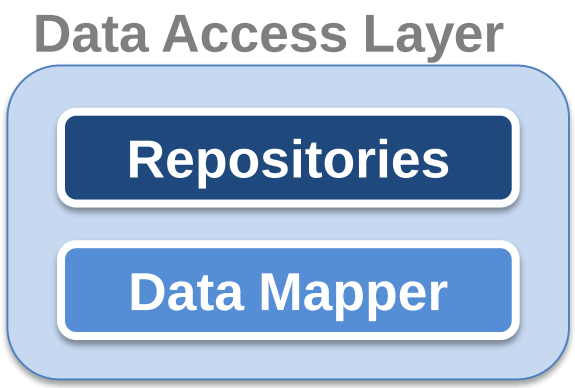


1 – REPOSITORY

DESCRIPCION – Sirve de intermediario entre la capa de dominio y la de mapeo de base de datos, usando una interfaz símil colección para acceder a los objetos del dominio.

- un Repository media entre las capas de dominio y de mapeo de datos, actuando como una colección de objetos de dominio en memoria.

- encapsula un conjunto de objetos homogéneos persistidos en un almacenamiento de datos, junto con las operaciones realizadas sobre ellos.



REPOSITORY – COMO TRABAJA?

- Para el código que usa al Repository, parece una simple colección en memoria de objetos de dominio.
- Remplaza los métodos finder de las clases Data Mapper por una aproximación basada en especificaciones para seleccionar objetos.
- No hay noción de ejecución de una query, sino más bien de satisfacción de una criterio de selección.
- El destino de un Repository podría no ser una base de datos relacional. Esto permite reemplazarlos por implementaciones alternativas que, por ejemplo, faciliten la definición de tests unitarios y agilicen su ejecución.
- Son un buen mecanismo para mejorar la legibilidad del código que usa extensivamente consultas.

REPOSITORY – USO

- 👍 Reduce la cantidad de código necesario para tratar con la interacción con la base de datos.
- 👍 Promueve el patrón Specification (por ej., en forma de objetos que definen el criterio) que encapsula el query a realizarse de una manera orientada a objetos.
- 👍 Los clientes nunca tienen que pensar en términos de SQL, sino mas bien en términos de objetos.
- 👍 Permite tener distintas alternativas de implementación a muy bajo costo (unit tests).

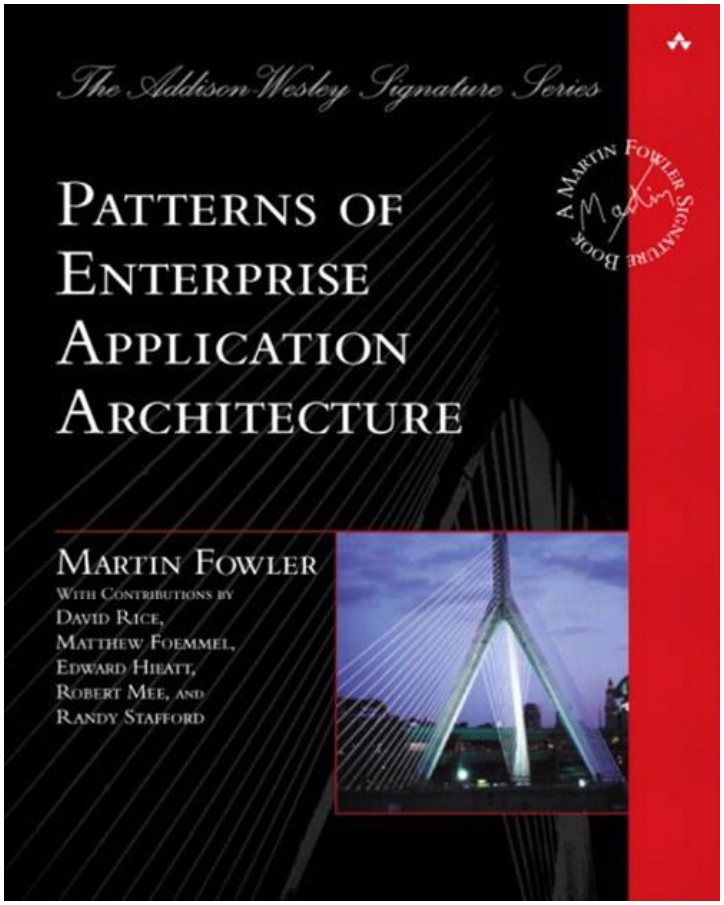
- 👎 En sistemas muy simples que no requieran de mucha complejidad.

CONTENIDO

Making Architecture Matter – Martin Fowler

https://www.youtube.com/watch?v=DngAZyWMGR0

BIBLIOGRAFIA



Patterns of Enterprise Application Architecture (2002)

Martin Fowler
www.martinfowler.com

Elsa Estevez
ece@cs.uns.edu.ar