Diseño de Sistemas Software

PEEA

Acceso a Bases de Datos

Introducción

- Muchas aplicaciones empresariales necesitan una **capa de acceso a datos** para almacenar información de forma persistente
- La mayoría de los sistemas almacenan los datos en una base de datos
  - **Las bases de datos orientadas a objetos** son una forma natural de persistir objetos, pero no son muy comunes
  - **Las bases de datos relacionales** son mucho más comunes, pero tienen una estructura distinta a los modelos orientados a objetos, y usan SQL para realizar las consultas

Introducción

- Tratar con bases de datos relacionales desde un systema orientado a objetos es complejo
- Distintos tipos de patrones
  - **Arquitecturales:** estructuran los objetos en la capa de datos
  - **De comportamiento:** definen cómo los objetos son almacenados y recuperados en la BBDD
  - **Etructurales:** definen cómo mapear la estructura de la lógica de negocio en la BBDD

PEEA

Patrones arquitecturales de acceso a datos

Patrones arquitecturales de acceso a datos

- La forma en que la BBDD se estructura y se consulta puede tener serias consecuencias sobre el rendimiento
- Es una buena práctica separar la lógica de negocio del acceso mediante SQL, de manera que
  - Diseñadores y desarrolladores se pueden centrar en escribir una buena lógica de negocio
  - Los expertos en BBDD pueden trabajar en el código SQL y la estructura de la BBDD para evitar cuellos de botella e interbloqueos

Patrones arquitecturales de acceso a datos

- Patrones arquitecturales
  - **Gateways:** objetos intermedios que se relacionan con la BBDD
    - **Table Data Gateway:** representa una tabla completa
    - **Row Data Gateway:** un objeto por cada registro
  - **Active Record:** los objetos de dominio son responsables de comunicarse con la BBDD
  - **Data Mapper:** capa de objetos que mapean el modelo de dominio a la estructura de la BBDD
- La elección depende principalmente de cómo se estructura la lógica de negocio

Patrones arquitecturales de acceso a datos

- Regla general (puede haber excepciones)

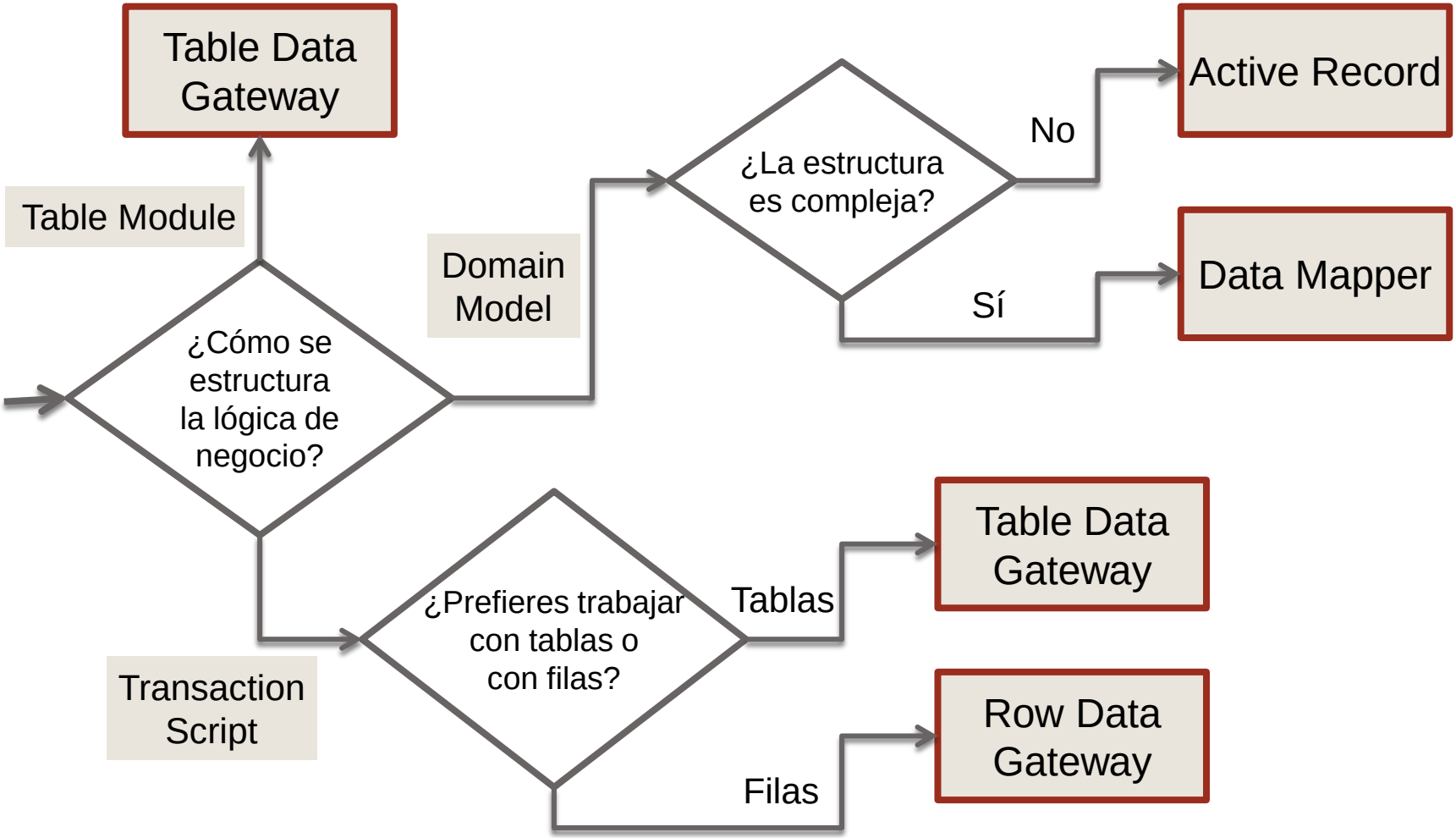


Table Data Gateway

- Definición:

  "Un objeto que actúa como portal para una tabla de la BBDD. Una única instancia gestiona todas las filas en la tabla"

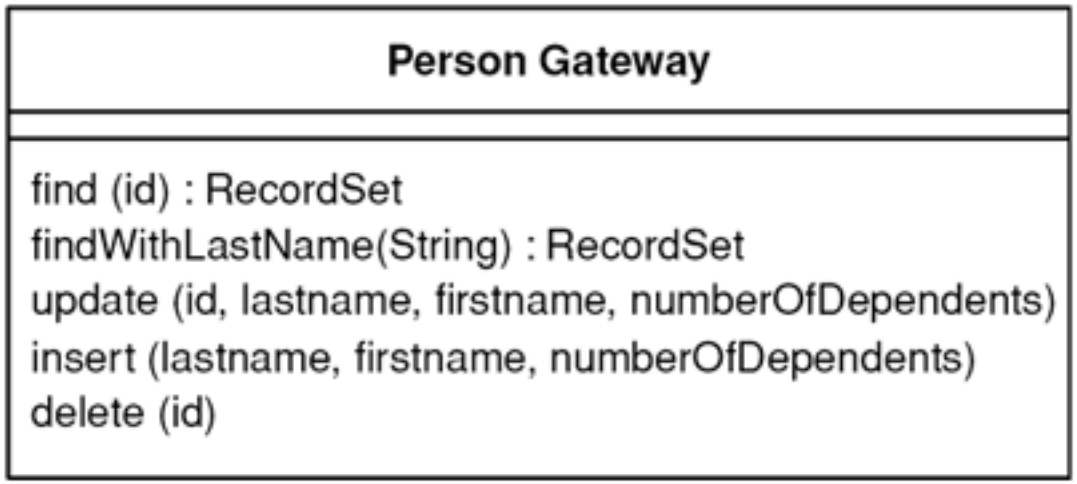


Table Data Gateway

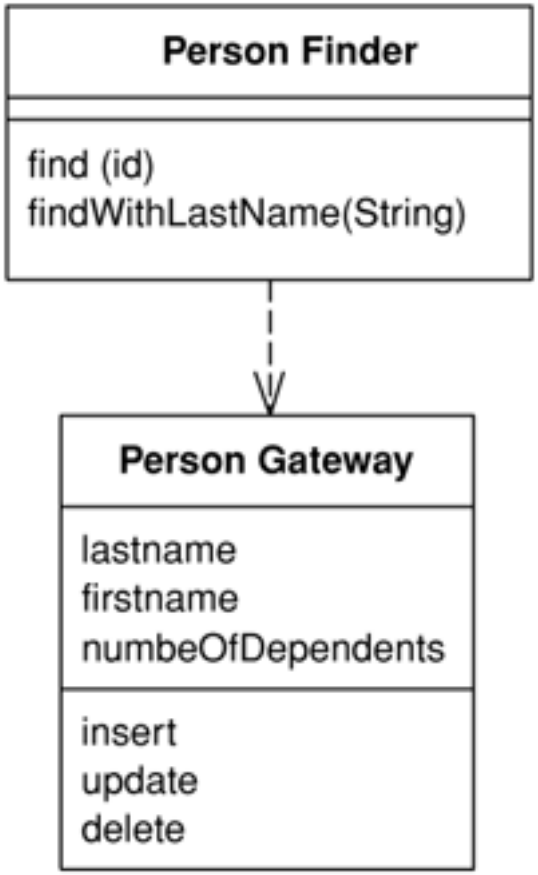
- Provee métodos CRUD para acceder a una tabla o vista (sin métodos de actualización para las vistas)
- Distintas estrategias para devolver los datos
  - Mapas de pares 'clave => valor'
  - *Data Transfer Object*
  - *Record Set*, útil cuando se usa *Table Module*

← Atrás

Row Data Gateway

- Definición:

  "Un objeto que actúa como portal para un único registro en la fuente de datos. Se crea una instancia por cada fila."



Row Data Gateway

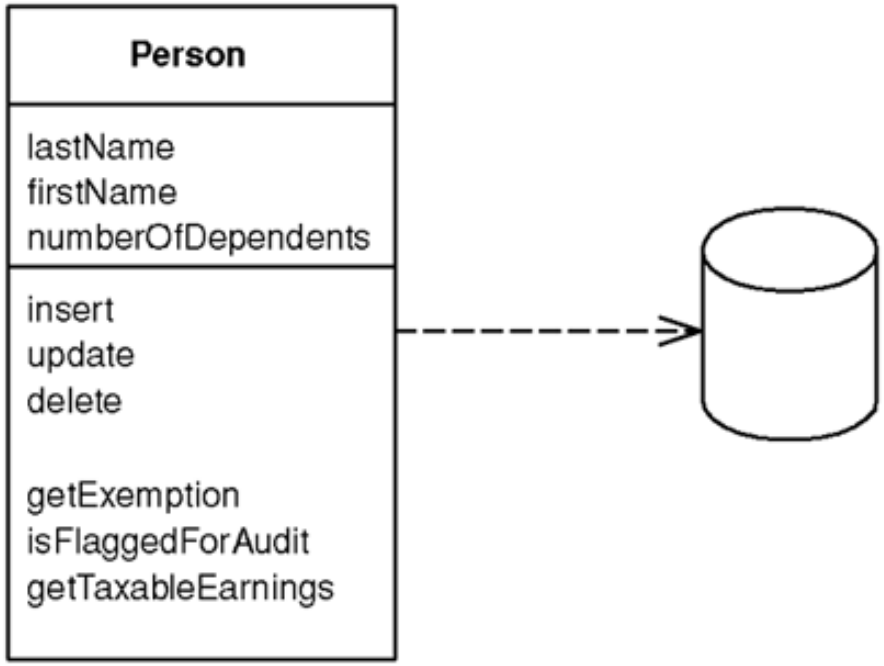
- Cada objeto *Row Data Gateway* representa un registro en la BBDD, con un campo por columna
- Permite acceder a los registros mediante una perspectiva OO, ocultando los detalles de la BBDD
- Dos estrategias para recuperar los objetos
  - Usando métodos *find* estáticos
  - Usando objetos especializados para las búsquedas

← Atrás

Active Record

- Definición:

  "Un objeto que contiene una fila en una tabla o vista, encapsula el código de acceso a BBDD y le añade lógica de negocio."



Active Record

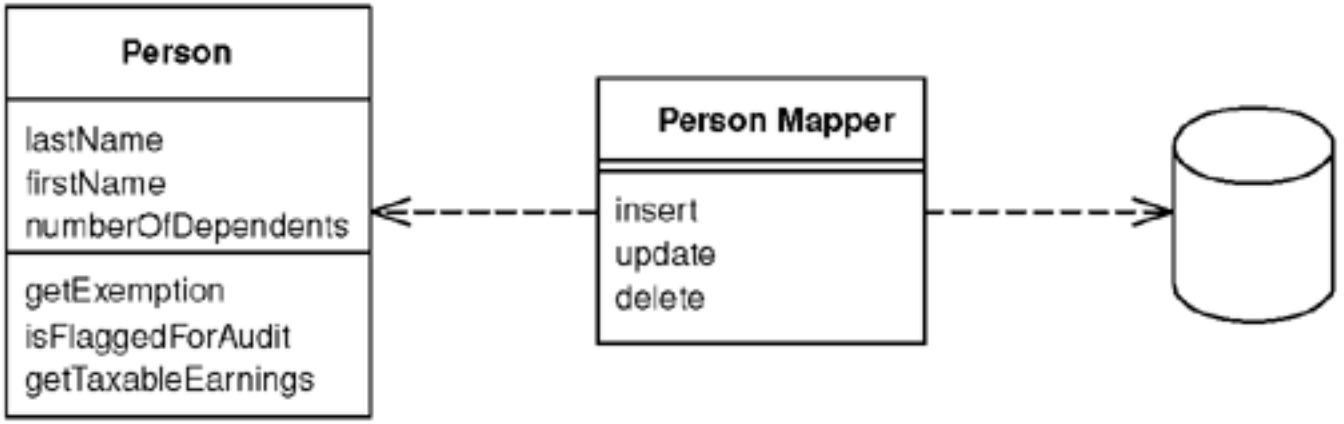
- Aproximación sencilla que mezcla lógica de negocio y de acceso a datos
- Útil cuando el modelo de dominio tiene una estructura muy parecida a la BBDD, con lógica sencilla (principalmente operaciones CRUD)
- Este patrón incrementa el acoplamiento con la BBDD y dificulta la refactorización

← Atrás

Data Mapper

- Definición:

  "Una capa de objetos (*mappers)* que transfieren los datos entre los objetos de dominio y la base de datos, manteniendo su independencia."



Data Mapper

- Los objetos *Data Mapper* transforman los datos desde el modelo de dominio a la estructura de la BBDD, y vice versa
- Necesario cuando el modelo de dominio es complejo (muchas relaciones, herencia, …)
- Facilita la navegación usando las relaciones de los objetos, cargando los datos de los objetos relacionados cuando es necesario
- Los patrones estructurales permiten implementar estas relaciones de forma efectiva

Data Mapper

- ¡No intentes implementar un *Data Mapper* desde cero! (a no ser que te guste el riesgo)
  - Necesita mucho tiempo y esfuerzo. Comprar una herramienta de mapeado objeto-relacional sería una mejor inversión
  - Pero… necesitas comprender estos patrones para hacer un uso correcto de estas herramientas

PEEA

Patrones Objeto-Relational de comportamiento

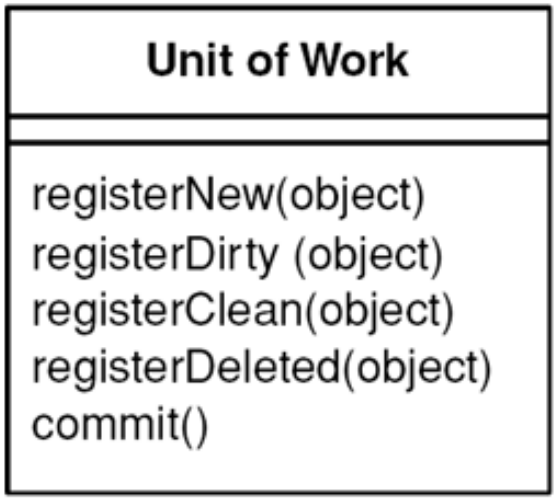
Patrones Objeto-Relational de comportamiento

- Los patrones de comportamiento gestionan cómo se almacenan y recuperan los objetos de una BBDD relacional
- Problemas a resolver
  - Se debe mantener la integridad referencial cuando se crean y modifican múltiples objetos (*Unit of Work*)
  - No debe haber múltiples copias del mismo objeto en memoria (*Identity Map*)
  - Cargar los objetos relacionados en un modelo de dominio puede acabar recuperando la BBDD completa (*Lazy Load*)

Unit of Work

- Definición:

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



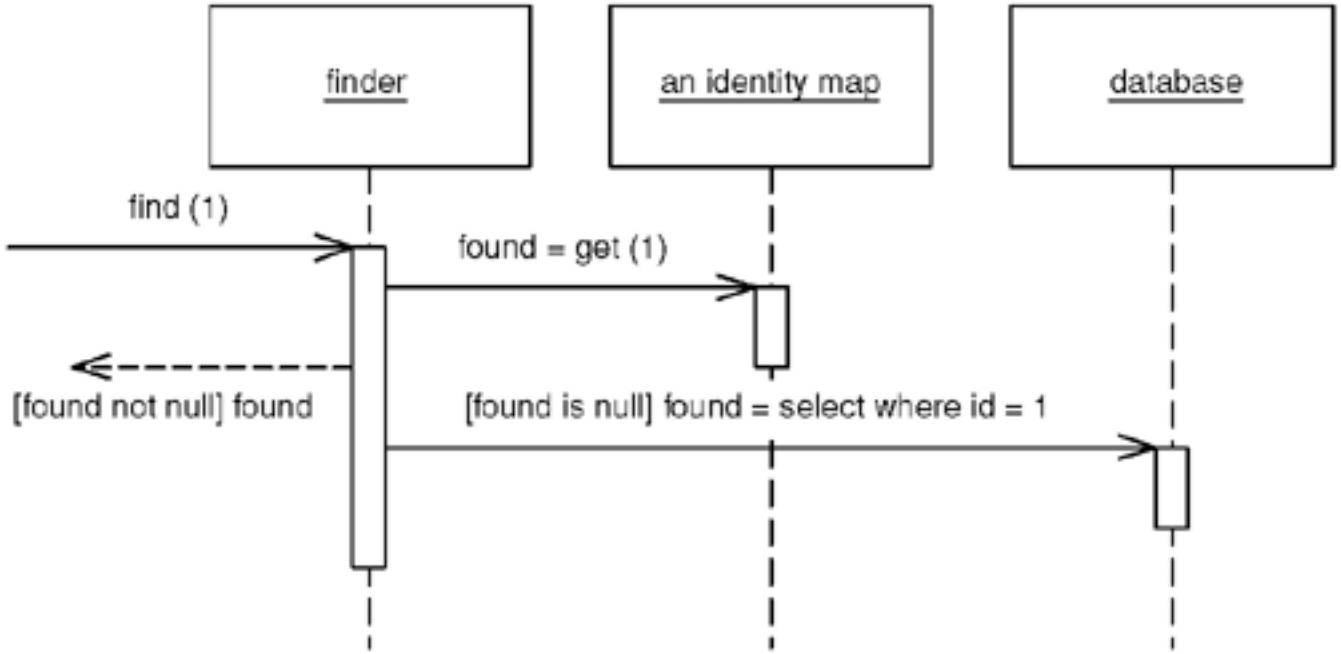
Unit of Work

- Llamar a la BBDD para cada cambio en el modelo de dominio puede afectar el rendimiento
- En lugar de llamar directamente a la BBDD, el sistema notifica al objeto *Unit of Work*, de manera que pueda llevar un registro de los objetos nuevos, recuperados de la BBDD, modificados y borrados
- Cuando es necesario, abre una transacción con la BBDD y realiza todos los cambios en orden

Identity Map

- Definición:

  "Asegura que cada objeto se carga una única vez llevando un registro de cada objeto cargado en un mapa. Busca los objetos en el mapa antes de recuperarlos de la BBDD."



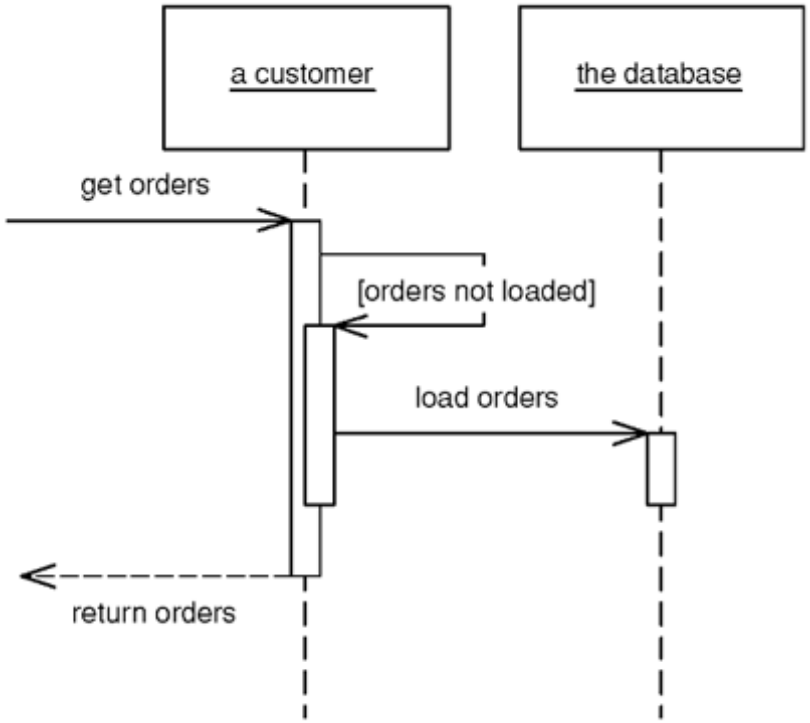
Identity Map

- Cargar un registro varias veces de la BBDD puede dar lugar a múltiples objetos con los mismos datos
- Problemas
  - Gasto excesivo de recursos
  - Inconsistencia cuando uno de ellos se modifica
- El objeto *Identity Map* lleva un registro de los objetos cargados y devuelve referencias para los que ya existe una instancia en memoria
- También actúa de caché para la BBDD

Lazy Load

- Definición:

  "Un objeto que no contiene los datos, pero sabe cómo obtenerlos."



Lazy Load

- Cuando se carga un objeto del modelo de dominio puede ser útil cargar sus objetos relacionados
- Sin embargo, esto puede degradar el rendimiento si el número de objetos relacionados es elevado, o incluso acabar cargando la BBDD completa en una reacción en cadena
- *Lazy Load* sólo carga un objeto cuando se usa

Lazy Load

- Distintas alternativas
  - **Lazy initialization:** usa un valor nulo para los objetos hasta que se cargan
  - **Virtual proxy:** los objetos se sustituyen por un objeto vacío que carga los datos cuando es necesario
  - **Value holder:** los objetos relacionados son cargados por otro objeto
  - **Ghosts:** los objetos se crean vacíos, y actúan como su propio proxy

PEAA

Patrones Objeto-Relacional estructurales

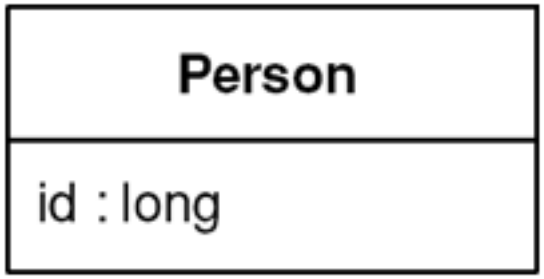
Patrones Objeto-Relacional estructurales

- Las relaciones se representan de manera diferente en la programación OO y en las BBDD
  - Los objetos relacionados se almacenan como atributos
  - Las BBDD usan una clave a otra tabla
  - Los objetos usan colecciones para contener múltiples referencias
  - En las BBDD, se da la vuelta a la relación y los objetos contenidos hacen referencia al contenedor
- Para facilitar el trabajo a un *Data Mapper*, las relaciones se pueden implementar tal como son en una BBDD

Identity Field

- Definición:

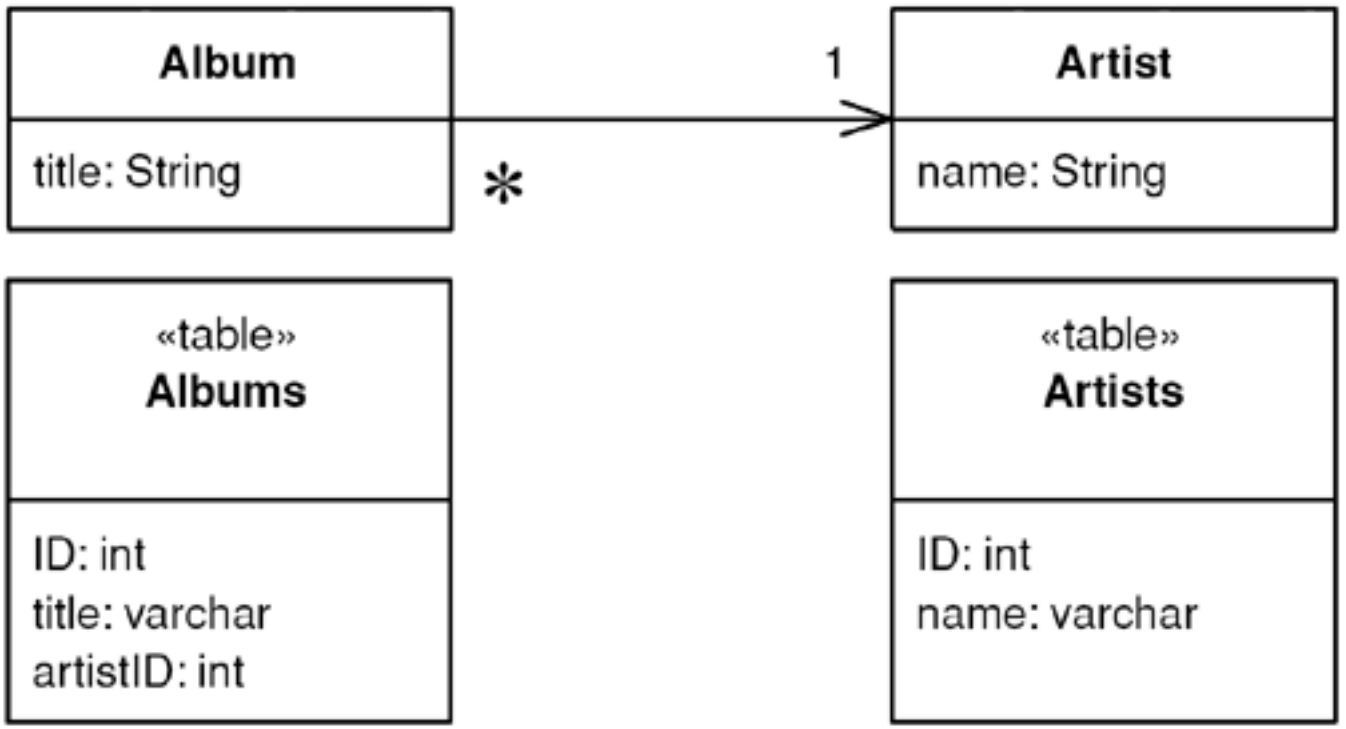
  "Almacena el ID de BBDD en los objetos para mantener la identidad entre los objetos en memoria y los registros de la BBDD."



Foreign Key Mapping

- Definición:

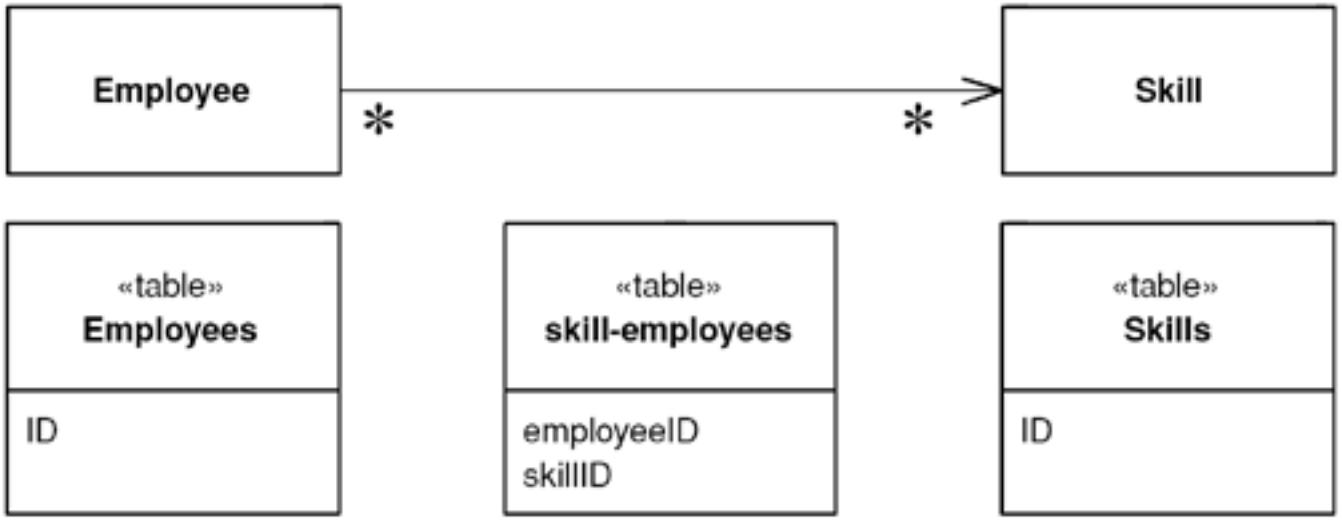
  "Mapea una asociación entre objetos a una clave ajena entre tablas."



Association Table Mapping

- Definición:

  "Almacena una asociación como una tabla con claves ajenas a las tablas enlazadas por la asociación."



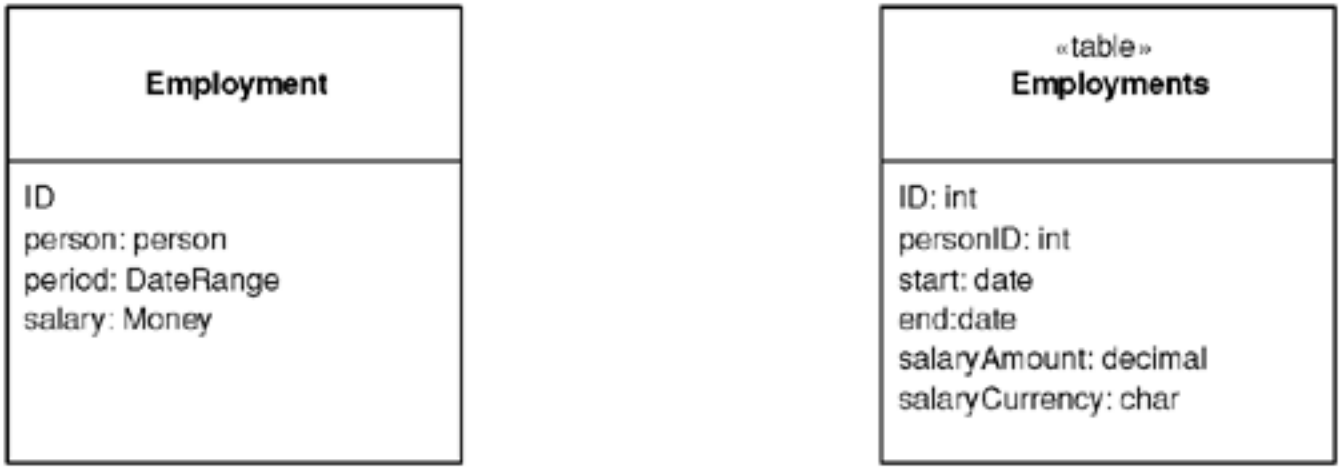
Patrones Objeto-Relacional estructurales

- A veces no es necesario mapear todos los objetos de dominio a tablas en la BBDD
  - Los objetos pequeños se pueden guardar junto a su contendor
  - Las colecciones o jerarquías de pequeños objetos se pueden guardar juntas, de manera que se pueda acceder a ellas en una sola operación

Embedded Value

- Definición:

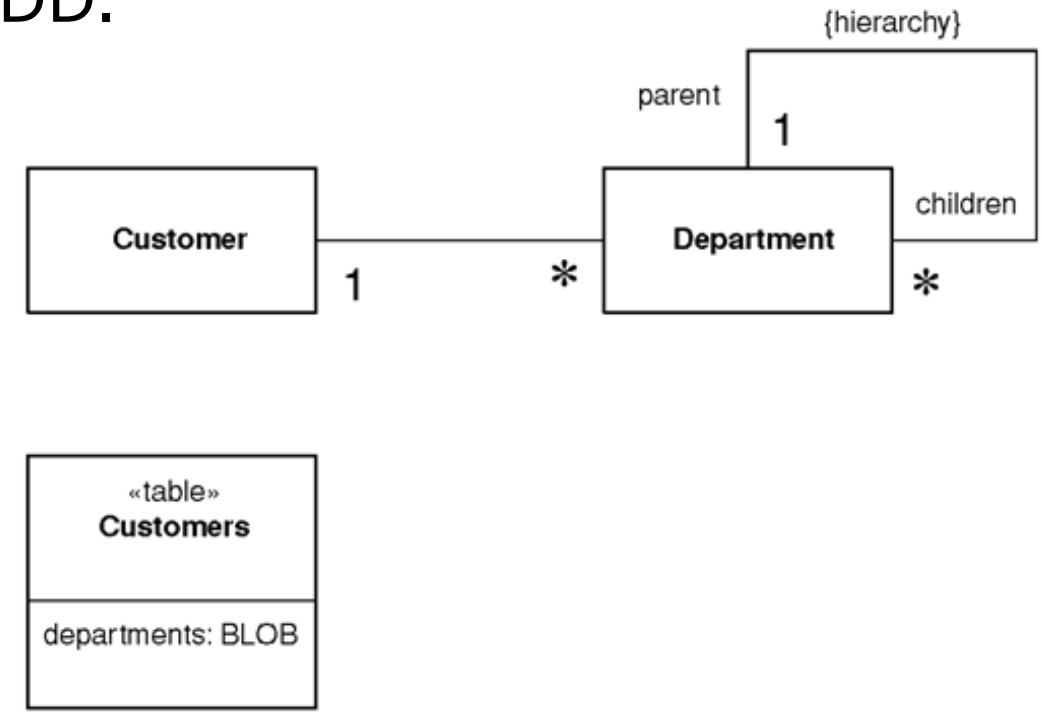
  “Mapea un objeto en varios campos de otra tabla.”



Serialized LOB

- Definición:

  "Guarda un grafo de objetos serializándolos en un único objeto (LOB) que se almacena en un campo de la BBDD."



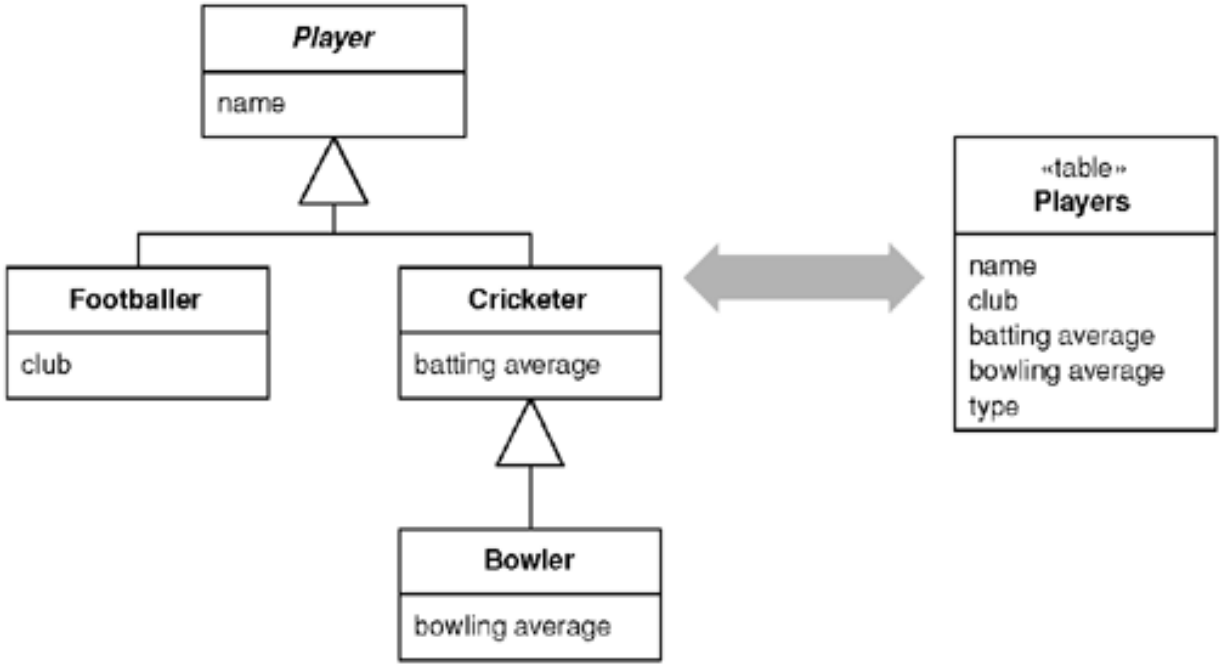
Patrones Objeto-Relacional estructurales

- Las BBDD relacionales no pueden representar jerarquías de objetos
- Tres opciones
  - **Single Table Inheritance:** mayor rendimiento, pero desperdicia espacio
  - **Concrete Table Inheritance:** si se cambia una tabla es necesario cambiar las demás
  - **Class Table Inheritance:** evita duplicación, pero disminuye el rendimiento porque necesita hacer *join* de varias tablas

Single Table Inheritance

- Definición:

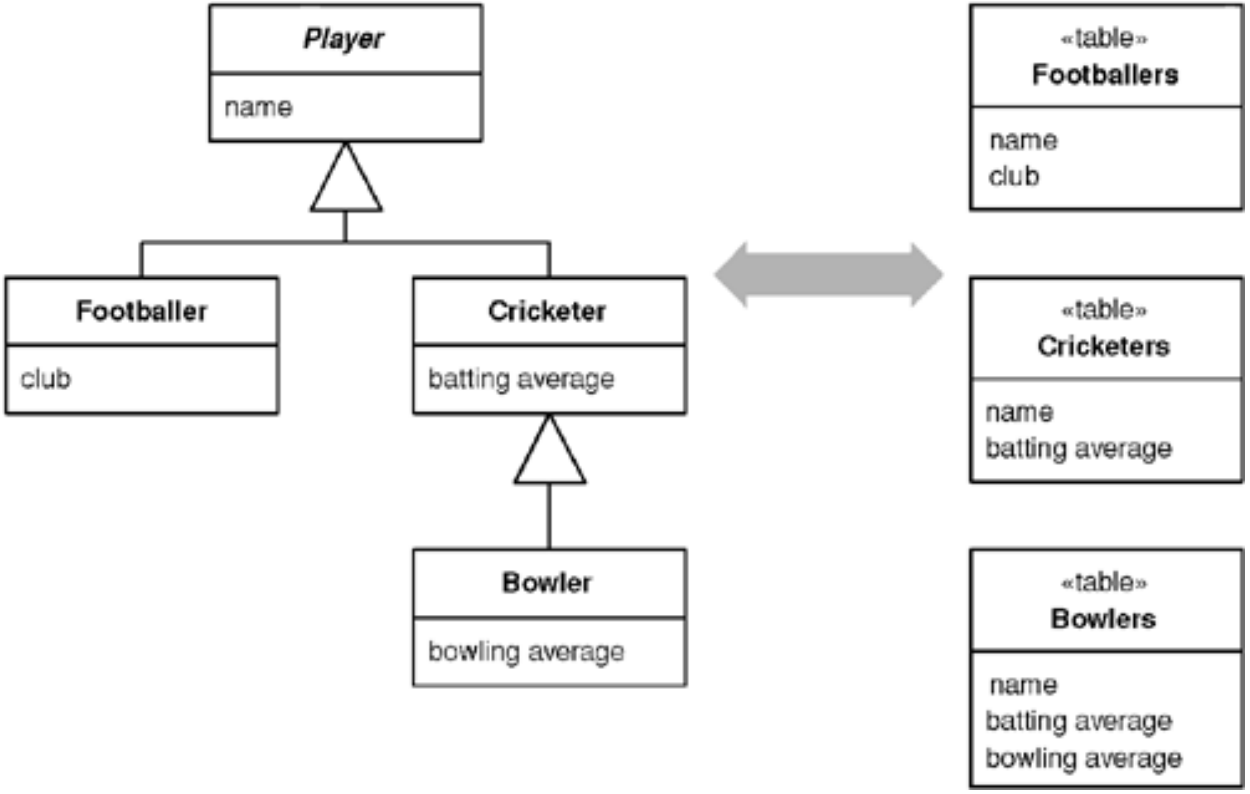
  "Representa una jerarquía de herencia como una única tabla que tiene columnas para todos los campos de todas las clases en la herencia."



Concrete Table Inheritance

- Definición:

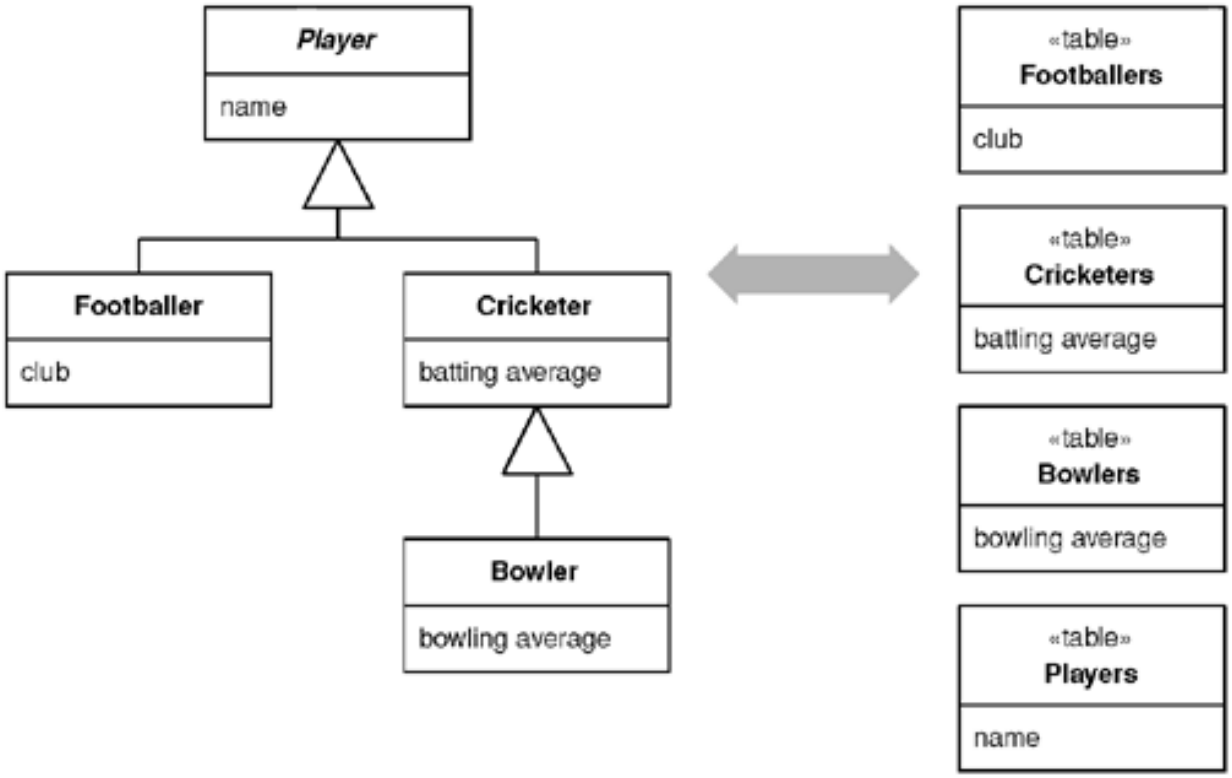
  "Representa una jerarquía de herencia con una tabla por cada clase concreta en la jerarquía."



Class Table Inheritance

- Definición:

  "Representa una jerarquía de herencia con una tabla por cada clase."



Notas finales

Notas finales

- Estos patrones se pueden combinar, p.ej. el *Data Mapper* puede usar un *Table Data Gateway* para comunicarse con la BBDD
- Los patrones simples y repetitivos pueden dar lugar a código simple y repetitivo. Esto se puede evitar usando *Metadata Mapping* y generación de código

```
<field name="customer" targetClass="customer"
       dbColumn="custID" targetTable="customers"
       lowerBound="1" upperBound="1"
       setter="loadCustomer"/>
```

Bibliografía

- Fowler, Martin

  Patterns of Enterprise Application Architecture

  Addison Wesley, 2003.