Arhitectura sistemelor soft enterprise. Platforma .NET

Curs 8

Pattern-uri de legatura comportamentala obiectual-relationala

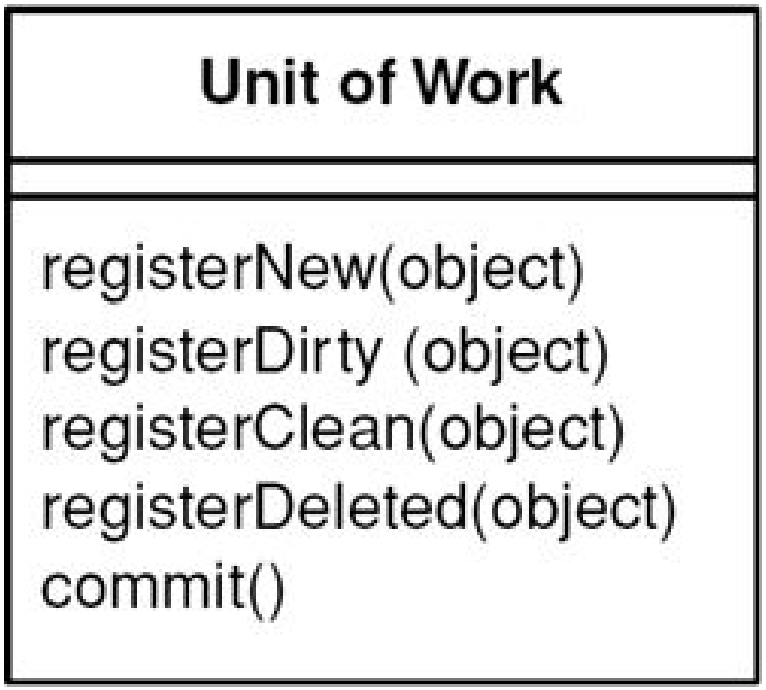
Pattern-uri de legatura obiectual-relationala

- Unit of Work
- Identity map
- Lazy Loading

- Sursa: PoEAA, capitolul 11. Object-Relational Behavioral Patterns

Pattern-uri de legatura obiectual-relationala

- Unit of Work:
  - Mentine o lista de obiecte afectate de o **tranzactie de business** si coordoneaza salvarea modificarilor survenite
  - Rezolva problemele de acces concurential



Pattern-uri de legatura obiectual-relationala

- Responsabilitati:
  - Managementul tranzactiilor
  - Ordonarea operatiilor de insert, update, delete
  - Prevenirea update-urilor duplicate
- Interfata **ITransaction** din NHibernate, clasa **DbContext** (wrapper peste ObjectContext) din Entity Framework sunt exemple de implementări UoW

```
public interface IUnitOfWork {
  void MarkDirty(object entity);
  void MarkNew(object entity);
  void MarkDeleted(object entity);
  void Commit();
  void Rollback();
}
```

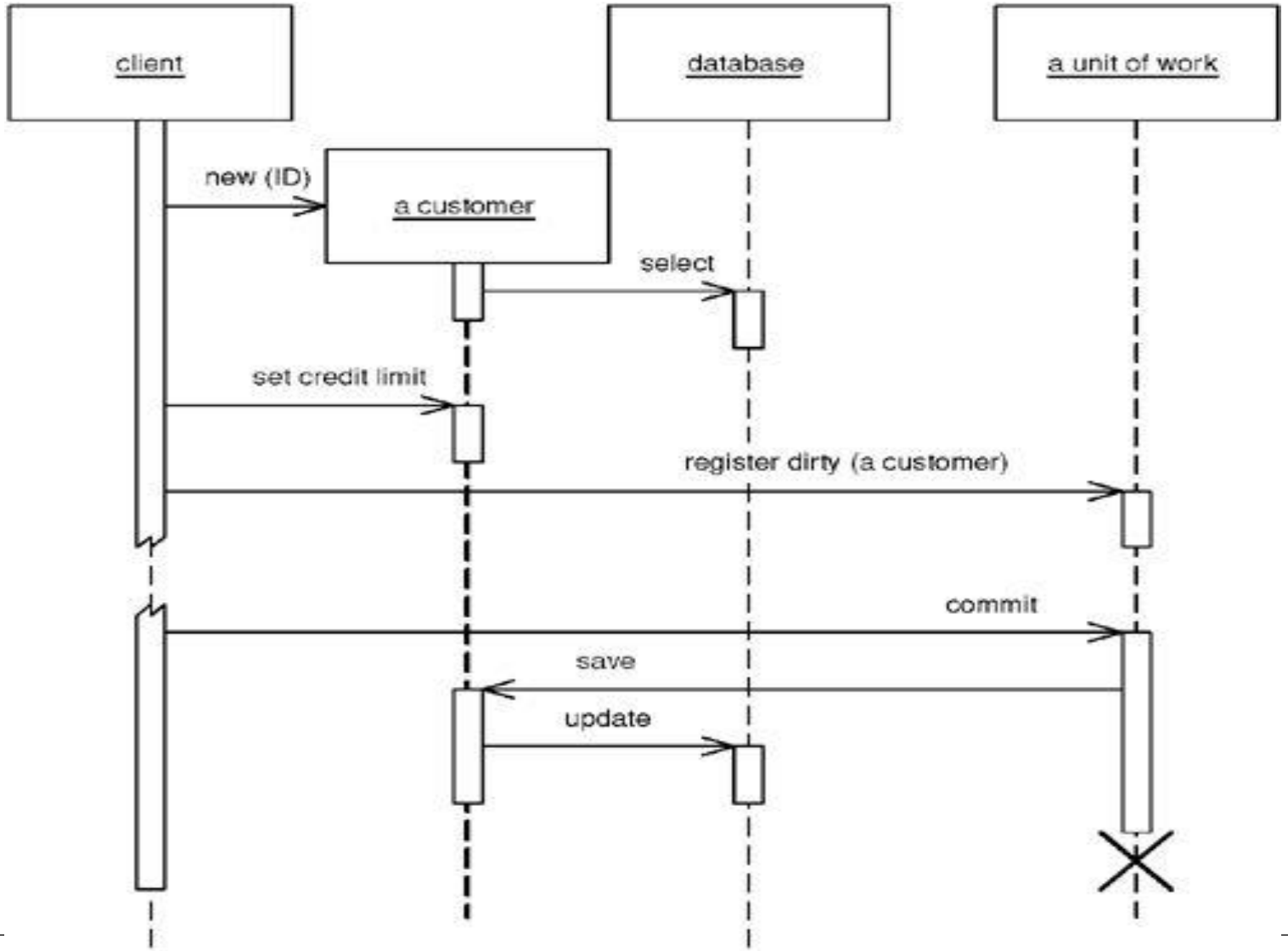
Unit of work

- Cand sunt extrase date din baza de date, este important sa ne asiguram ca pastram o evidenta a ceea ce s-a modificat; pe langa asta, ce s-a creat si se doreste a fi inserat in BD
- Similar: trebuie efectuata scrierea obiectelor noi in BD, stergerea *inregistrarilor* pentru care se cere stergerea *obiectelor* in aplicatie
- Varianta posibila, dar naiva: se actioneaza si pe tabele de indata ce se modifica starea unui obiect incarcat din BD sau se creeaza un nou obiect
- Impotriva acestei idei:
  - prea multe apeluri "marunte" catre BD => acces lent; trebuie minimizate apelurile la distanta, deoarece au impact negativ asupra performantei aplicatiei client
  - pentru fiecare operatie ar trebui deschisa/inchisa o conexiune si o tranzactie sistem, ceea ce nu trateaza cazul in care mai multe operatii fac parte dintr-o tranzactie de business (tranzactie cu multiple cereri de operare asupra datelor)
  - A se vedea de exemplu suportul ADO.NET 2.0 pentru asemenea operatii + MARS: Multiple Active Results Sets, https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/enabling-multiple-active-result-sets

Unit of work

- Cum functioneaza:
  - De indata ce se incepe operatie care poate sa schimbe ceva pe baza de date, se instantiaza un obiect Unit of Work
  - La actualizarea modificarilor pe BD: UoW se ocupa cu verificarea de acces concurent
  - Plus: cand vine momentul de comitere, UoW decide ce operatii trebuie sa solicite SGBD-ului si in ce ordine
    - Exemplu: se insereaza un nou produs dintr-o noua categorie; pentru a se respecta integritatea referentiala, prima data trebui inserata categoria (si se obtine o cheie) si apoi produsul, pentru a putea seta valoarea cheii straine in produs

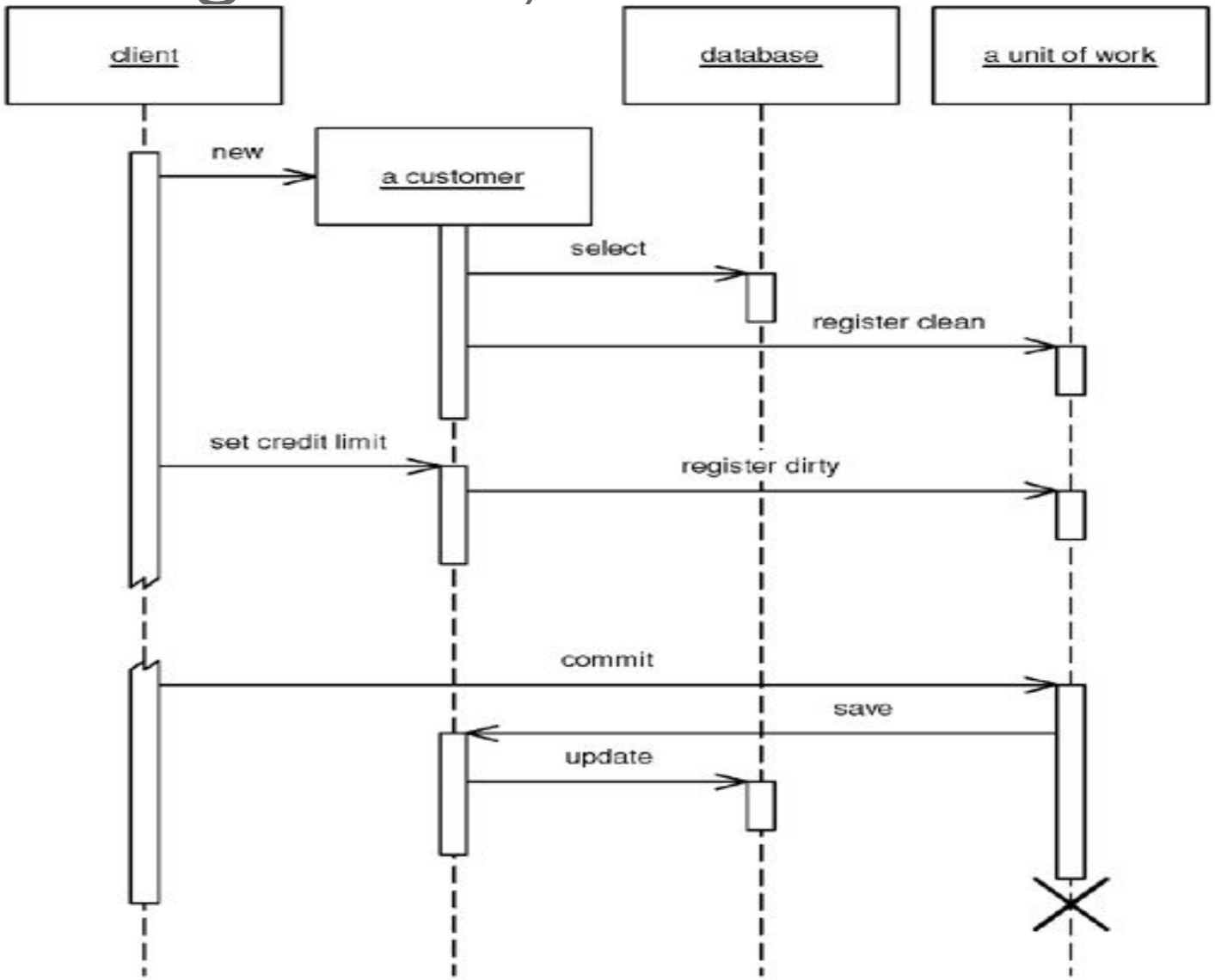
Unit of work: inregistrarea de catre apelator (caller registration)



Unit of work: inregistrarea de catre apelator (caller registration)

- Particularitati:
  - Codul client trebuie sa faca apeluri catre UoW pentru a face inregistrarea obiectului/obiectelor cu care se lucreaza
  - Daca nu faci apel la UoW, atunci obiectul nu va fi comis in baza de date (sters/inserat/modificat)
    - Avantaj: poate vrei sa lucrezi cu niste obiecte pe care uneori nu intentionezi sa le salvezi
    - Dezavantaj pentru scenariul anterior: sporeste confuzia, se lucreaza si cu doua familii de obiecte: unele care se urmeaza sa fie salvate in BD, altele nu

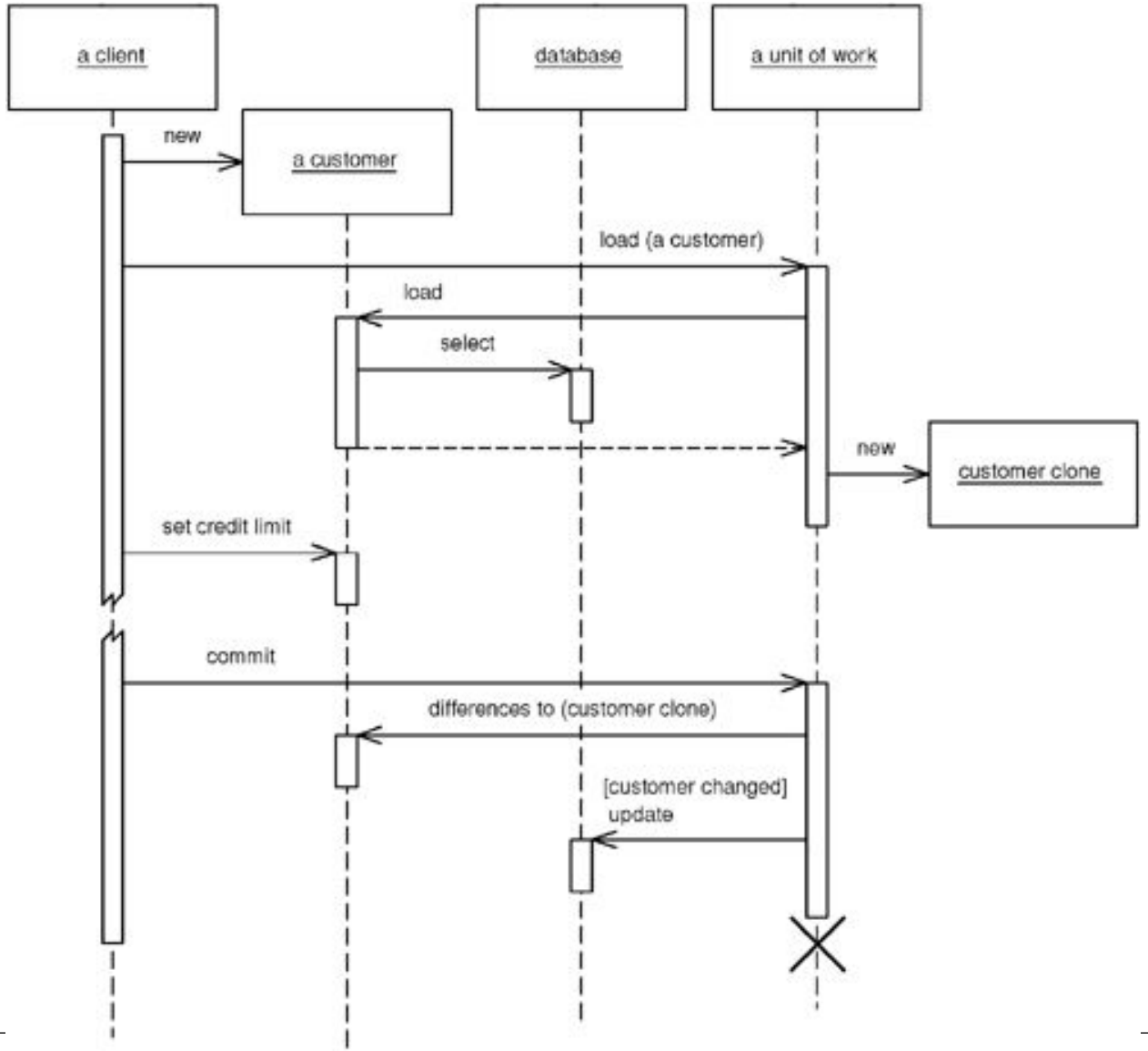
Unit of work: inregistrarea de catre obiect (object registration)



Unit of work: inregistrarea obiectului (object registration)

- Responsabilitatea inregistrarii obiectului cu UoW nu se mai face de catre client
- Implementare: in metodele obiectului se plaseaza apeluri catre UoW.
  - Incarcarea unui obiect declanseaza apel de metoda “registerClean” catre UoW
  - La modificarea starii obiectului se declanseaza apel de “registerDirty”
  - Fie se paseaza UoW ca parametru fiecarui obiect, fie se depune intr-un loc accesibil pentru obiect (e.g. singleton la nivel de thread = Registry la nivel de thread)
  - Cererea de commit pentru UoW se face de catre client
  - Posibilitati de implementare:
    - Aspect Oriented Programming = injectare de cod pre/post operatie;
    - proxy

Unit of Work ca si controller pentru baza de date



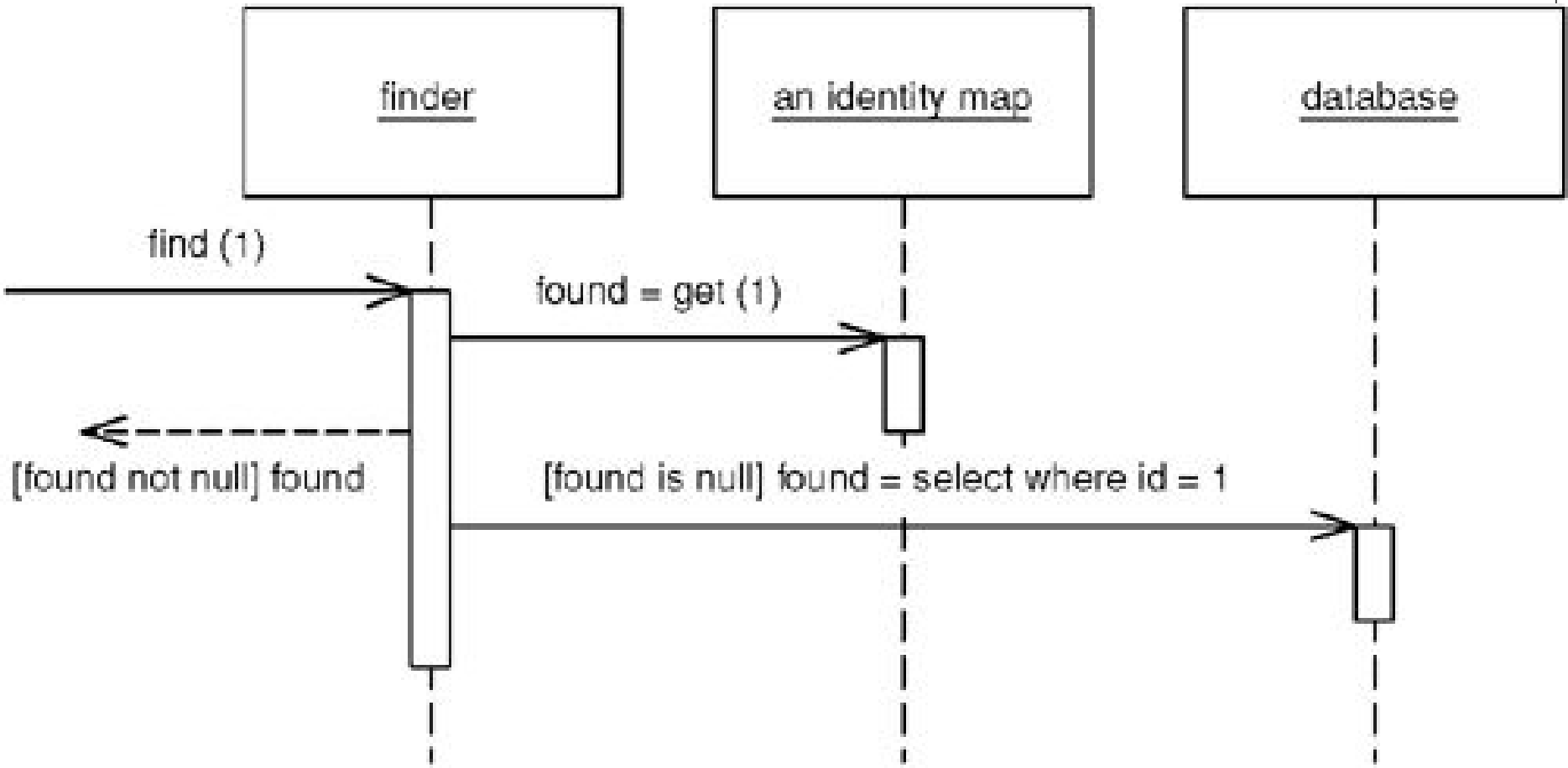
Unit of Work: beneficii

- Poate avea grija de ordinea in care se fac modificarile pe baza de date
- Reducere de blocaje mortale (deadlocks)
- Batch updates: se pot trimite mai multe comenzi SQL catre baza de date intr-o singura cerere = o singura conexiune deschisa catre baza de date

Identity map

- Se asigura de faptul ca fiecare obiect e incarcat in memorie o singura data, pastrandu-le intr-un dictionar
- Cauta obiectele in acest dictionar atunci cand se cere referinta la ele
- Scop: se evita posibilitatea de a avea 2 obiecte in memorie care corespund aceleiasi inregistrari din BD

Identity map



Identity map

- Rezultat: pentru orice inregistrare identificata prin cheie nu se va crea mai mult de un obiect
- Pe langa faptul ca se asigura consistenta datelor pastrate in memorie, se elimina si interogarea repetata pentru aceleasi date => reducerea de apeluri la distanta => marirea vitezei aplicatiei
- Efect de caching: datele ce se afla deja in gestiunea lui IM sunt accesate mai rapid
  - *Caching-ul este un doar un efect (benefic), nu scopul principal al lui IM*

Identity map

- Cum lucreaza:
  - O serie de dictionare care retin obiectele ce au fost extrase din baza de date
  - Varianta simpla: un dictionar pentru fiecare tabela
- Alegerea cheilor de dictionar
  - Alegere evidenta: cheia primara a tabelei
  - Lucreaza bine daca cheia este simpla si imuabila
  - Exemplu de chei preferate: cheie surogat, http://en.wikipedia.org/wiki/Surrogate_key

Identity map

- Variatiuni de realizare:
  - Explicita
  - Generica
- Varianta explicita: accesarea se face cu metode distincte pentru clase distincte: FindPerson(2), FindOrder(3)
- Varianta generica: accesarea se face printr-o singura metoda: Find(“Person”, 2)
  - Care ar putea fi problemele cu varianta generica?

Identity map

- Varianta explicita: are avantajul verificarii la compilare (orice pereche este de forma int (cheie)-Person, nu int-Object)
- De asemenea, e simplu de vazut pentru fiecare indentity map cate apeluri au primit
- Daca se foloseste varianta generica: toate cheile trebuie sa aiba acelasi tip => trebuie implementat corespunzator enterprise pattern-ul Identity Field

Identity map

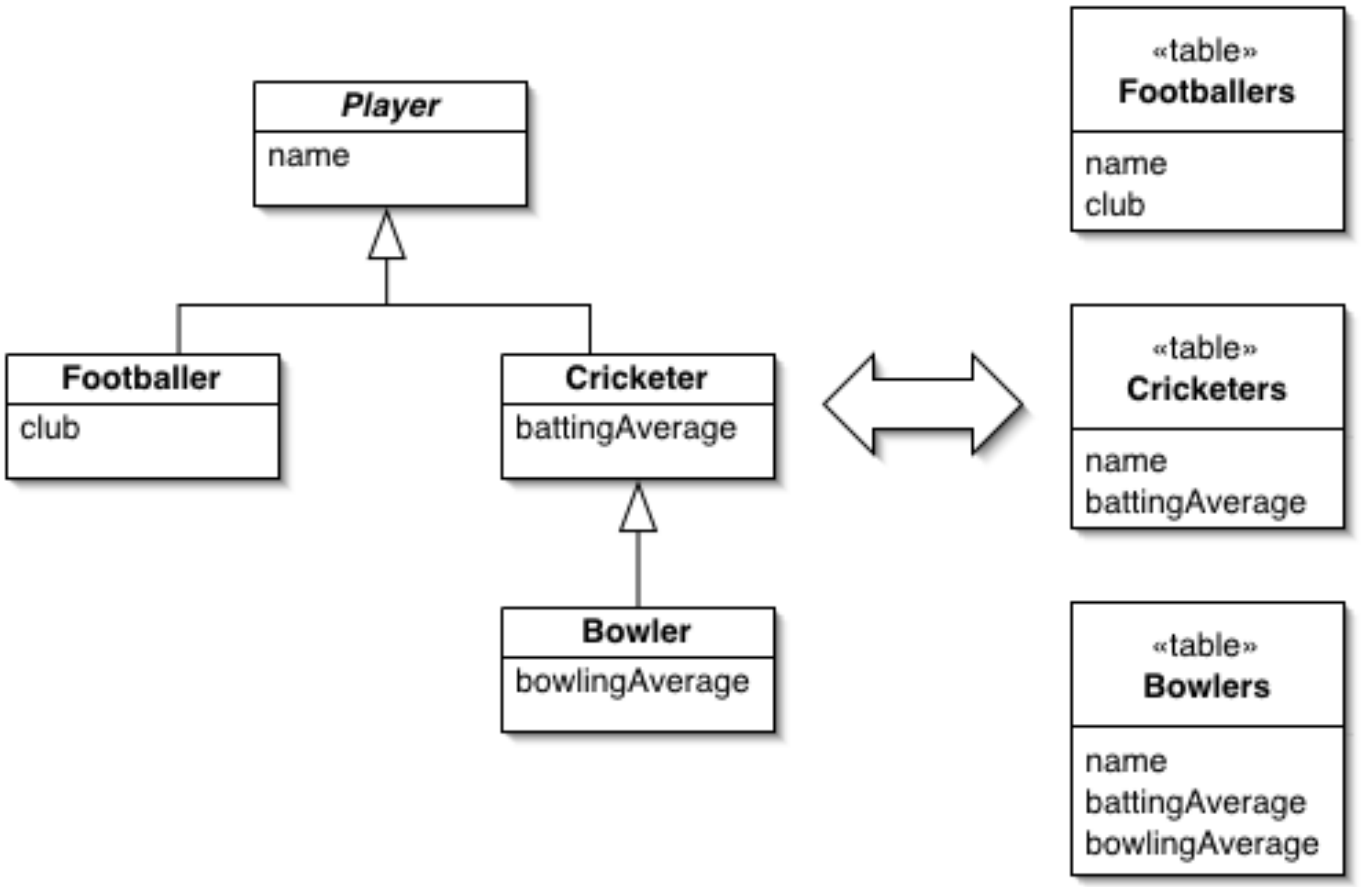
- Cate map-uri?
  - Cate un map pe clasa vs. un map pentru toate clasele
  - Un singur map pentru toate clasele => chei unice in baza de date (cum se pot obtine?)
  - Mai multe map-uri: cate un map pentru fiecare clasa sau tabela; aceasta varianta lucreaza bine daca tabele si clasele sunt asociate 1:1
  - Daca tabelele si clasele nu sunt 1:1, atunci se sugereaza ca map-urile sa se refere la clase

Identity map

- La mai multe map-uri apare problema in caz de relatii de mostenire intre clase
- Masina, subclasa de Vehicul => un singur map sau mai multe, legate prin derivare? Daca sunt mai multe map-uri, atunci cautarea trebuie facuta in toate
- De preferat: cate un map pentru fiecare arbore de derivare, dar numai daca cheile sunt unice de-a lungul arborelui => este exclusa folosirea de concrete table inheritance

Identity map

- Concrete table inheritance:



Identity map

- Unde se pun identity maps?
  - Sa fie usor de gasit de catre o sesiune
  - Fiecare sesiune sa primeasca propria sa instanta, care sa fie izolata de orice alta sesiune de instanta
  - Daca se foloseste Unit of Work, atunci acesta este locul cel mai potrivit de a gazdui un identity map
  - Daca nu ai UoW, atunci un Registry = obiect bine cunoscut de catre toate obiectele interesate; e.g. singleton la nivel de sesiune (e.g. sesiune HTTP)
- Daca se partajeaza un Identity Map pentru mai multe sesiuni => necesar a asigura protectie la acces concurent

Identity map

- Cand se foloseste:
  - Cand se doreste managementul obiectelor aduse din baza de date: vrei o singura instanta pentru aceleasi date de provenienta
  - Pe langa avantajul asigurarii unicitatii inregistrarilor incarcate in memorie: caching
  - Nu e obligatoriu a se folosi Identity Map cand se lucreaza cu obiecte imuabile (eng: immutable), dar IM ramane cu avantajul de caching;

Identity map

- Ramane inca problema controlului de acces concurent la inregistrari pentru mai multe sesiuni
- Problema tratata mai tarziu prin Optimistic Offline Lock sau Pessimistic Offline Lock

Identity map

- **Exemplu de implementare in .NET**

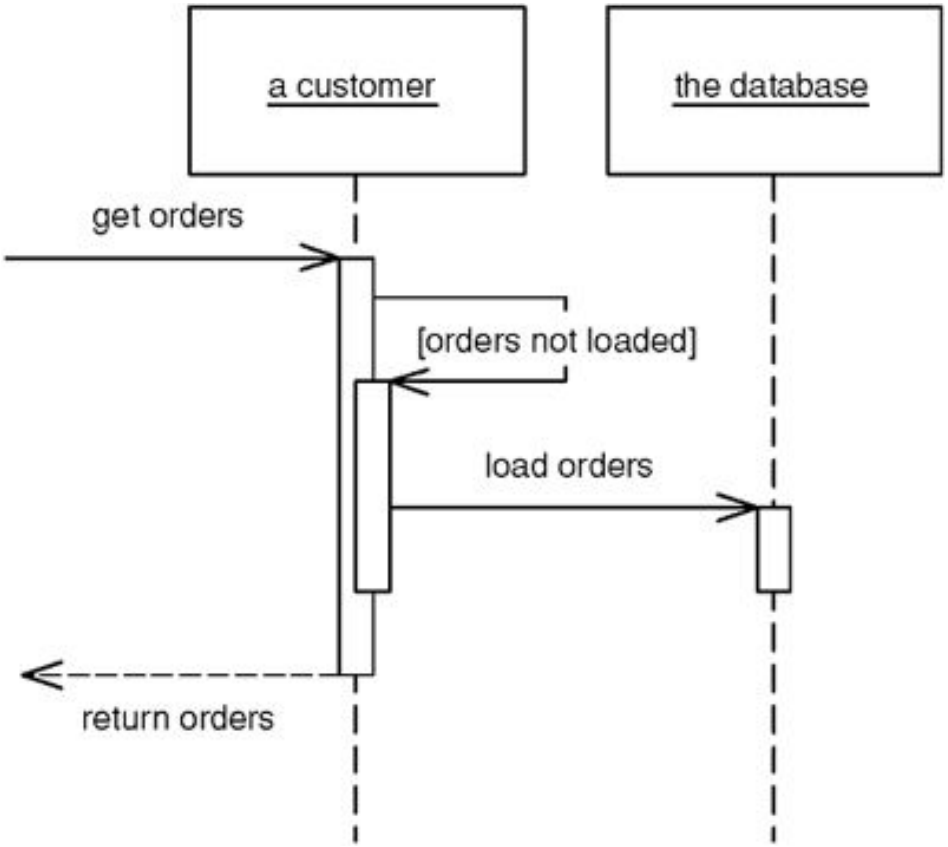
```
Dictionary<int, Person> personsMap;

public void AddPerson(Person p)
{
    personsMap.Add(p.Id, p); //atentie, aici se poate arunca exceptie de
                             //catre clasa Dictionary
}

public Person GetPerson(int id)
{
        if (personsMap.Keys.Contains(id))
                return personsMap[id];
        else
                return null;
}
```

Lazy Load

- Folosit in cazul unui obiect care nu contine toate datele, dar stie cum sa le obtina



Lazy Load

- Incarcarea datelor din start = varianta confortabila: ori de cate ori ai nevoie de un obiect, il incarci cu toate dependintele sale
- Avantaj: acces imediat la date
- Dezavantaje:
  - durata mare de incarcare, trafic pe retea
  - incarcare masiva de date =: presiune inutila asupra memoriei RAM
    - La numar mare de sesiuni concurente, presiunea creste

Lazy Load

- Un Lazy Load intrerupe acest proces de incarcare pentru moment
- Se lasa un marker in structura obiectului prin care se arata ca datele nu sunt inca incarcate
- Datele se pot incarca la dorinta
- Exemple: campuri de tip BLOB (imagini stocate pe disc sau in BD); inregistrari de tip detail (partea “many” a unei relatii de tip “one to many”)

Lazy Load

- Cum poate fi implementat:
  - Lazy initialization
  - Virtual proxy
  - Value holder
  - Ghost

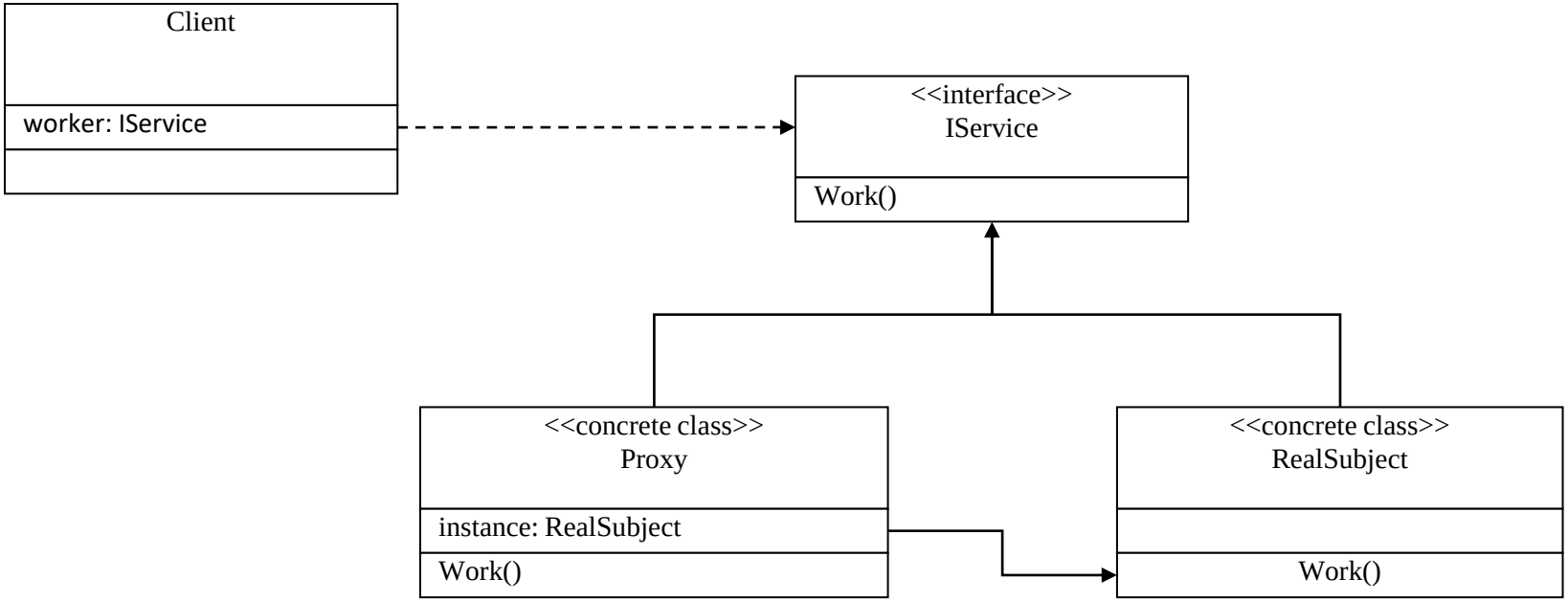
Lazy Load

- Var 1: Lazy initialization
  - Cea mai simpla varianta
  - Fiecare acces la camp verifica daca campul e null
  - Daca da, se calculeaza/aduce valoarea campului inainte de returnarea lui;
  - Conditie: incapsulare, accesare a campului doar prin getter, chiar si in interiorul clasei
  - Atentie daca nu cumva valoare de null este legala in cadrul clasei – caz in care se semnaleaza lipsa de incarcare in alt mod - e.g. indicator boolean

Lazy Load

- Lazy initialization: simplu, dar produce dependenta intre obiect si structura bazei de date
- De aceea lucreaza cel mai bine cu: Active Record, Table Data Gateway, Row Data Gateway
- Daca se foloseste Data Mapper, atunci se va folosi un strat suplimentar de indirectare care se obtine prin virtual proxy (vezi mai jos)

Lazy Load


Client
worker: IService
<<interface>>
IService
Work()
<<concrete class>>
Proxy
instance: RealSubject
Work()
<<concrete class>>
RealSubject
Work()

Lazy Load

- Var 2: virtual proxy
  - Arata exact ca si obiectul pe care il reprezinta, dar initial e o simulare a lui
  - Nefiind insa chiar acel obiect, mare atentie la problemele legate de identitate a obiectelor (nu a inregistrarilor)
  - Iarasi mare atentie la faptul ca se pot crea mai multe implementari proxy pentru acelasi obiect (unele pentru incarcare, altele pentru verificare de drepturi de acces sau pentru operatii de logica de business suplimentare etc.)
  - Aceste proxy-uri pot fi utile in context de testare (incluzand mocking)

Lazy Load

- Problemele legate de identitate nu se manifesta in context de colectii de obiecte
- Ca atare, colectiile de obiecte sunt candidate perfecte pentru Lazy Loading
- Varianta: se aduc doar acele elemente din colectie care sunt necesare (dar atentie la a detecta cand ceea ce s-a adus nu e suficient pentru a lucra mai departe); similar cu paginarea datelor, uneori benefica pentru GUI

Lazy Load

- Var 3: Value holder
  - Value holder = un obiect care "imbraca" un alt obiect
  - Accesarea datelor se fac prin mesaje catre Value holder (deci acesta trebuie cunoscut)
  - Doar la primul acces se face apel catre BD, pentru incarcarea propriu-zisa
- Var 4: Ghost
  - Un obiect real cu stare partial incarcata
  - Exemplu (extrem): cand se incarca din baza de date, el contine doar valoarea de cheie primara
  - Ori de cate ori se acceseaza vreun camp, se incarca din baza de date
  - In afara da ID, se mai pot incarca alte cateva date la inceput, cu sanse mari de a fi cerute imediat

Lazy Load

- Posibile probleme pentru utilizare de ghost: mostenirea – trebuie stiut exact ce fel de ghost trebuie creat
- Alt posibil pericol: se pot cere mai multe accese la BD decat este nevoie (poate o jonctiune sau folosire de subselectii ar fi mai bune in anumite situatii)
- De evitat: incarcarea in stil lazy loading a obiectelor individuale dintr-o colectie, ex: prima comanda, a doua comanda; mai degraba incarcarea intregii colectii dintr-un singur acces, la prima cerere, sau paginare pe batch-uri mari
  - Model adaptiv pentru decizie lazy/eager loading

Lazy Load

- Atentie la ce se cere in scenariile de utilizare: in unele cazuri, lazy loading este nerecomandat
- Modalitate de implementare a acestor scenarii: obiecte de interactiune cu baza de date, dedicate scenariilor de acces
- Aceste obiecte incarca intr-un mod specific graful de obiecte
- Varianta: un singur obiect de incarcare a datelor, dar cu posibilitatea de a primi indicatii despre strategia de incarcare folosita

Lazy Load

- Extremele: incarcare completa si incarcare doar de campuri strict necesare (la minim: doar cheia primara) = ghost
- Cand (nu) se foloseste:
  - Nu are sens daca e vorba de un camp din inregistrarea care se acceseaza devreme (posibila exceptie: daca campul e BLOB, poate sa fie incarcat mai tarziu; BLOB-urile se folosesc mai rar in aplicatie)
  - Fiti lenesi in decizia de a folosi lazy loading ☺ (complexitatea indusa de acest mecanism e mare, daca nu se foloseste un cadru O/RM)

Intrebare

- In JDBC si (N)Hibernate se pot trimite intr-o singura comanda mai multe interogari catre baza de date?