Arhitectura sistemelor soft enterprise. Platforma .NET

Curs 3

Asocierea cu bazele de date relationale

Motivatie

- Subiect: Data source layer
- Bibliografie: cap 3 din PoEAA
- Rolul stratului de sursa de date este comunicarea cu infrastructura ce mentine sau furnizeaza datele
- Datele preluate din infrastructura de date sunt apoi disponibile stratului de modelare a logicii aplicatiei
- O partea dominanta a lui data source layer: conversatia cu colectiile de date persistate
- Mod de stocare a datelor
  - Solutii mostenite (“legacy”)
  - Cel mai des: baze de date relationale => interogari in limbaj SQL – dialecte dependente de producatorul serverului de BD

Pattern-uri arhitecturale (1)

- Scop: precizarea modurilor in care se face comunicarea cu BD
- Deciziile luate sunt influentate de stratul de modelare a logicii aplicatiei, deci cele din cursul anterior trebuie intelese
- Motivatia considerarii unui strat de acces la date: nu toti programatorii cunosc bine limbajul SQL sau scrierea de cod pentru servere de baze de date
  - avem elemente specifice dialectului SQL
  - probleme de optimizare, plecand de la scrierea interogarilor pana la reglari pe baza de date: indecsi, partitionari de tabele etc.
- Modelarea relationala este diferita fata de cea folosita in domain layer

Dialecte SQL

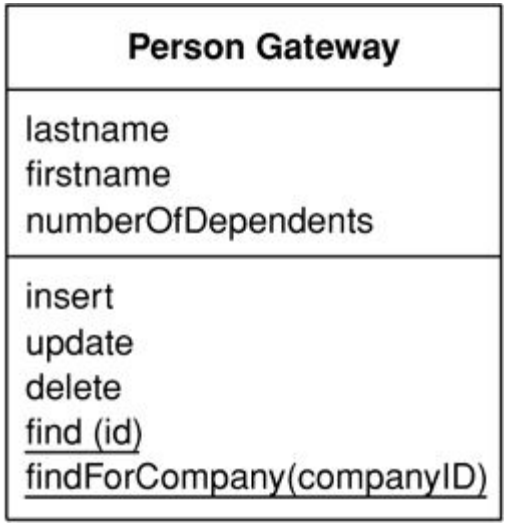
- Select pentru primele *n* inregistrari:
  - PostgreSQL/MySQL: SELECT columns FROM tablename ORDER BY … ASC LIMIT n
  - TSQL: SELECT TOP n columns FROM tablename ORDER BY … ASC
  - PL/SQL: SELECT columns FROM (SELECT ROW_NUMBER() OVER (ORDER BY … ASC) AS rownumber, columns FROM tablename) WHERE rownumber <= n
- Left outer join:
  - Oracle 8i: select last_name, department_name from employees e, departments d where e.department_id = d.department_id(+)
  - Oracle 9i+/SQL Server: select last_name, department_name from employees e left outer join departments d on e.department_id = d.department_id
  - PostgreSQL/MySQL: select last_name, department_name from t1 left [outer] join t2 on t1.num = t2.num

Pattern-uri arhitecturale (2)

- Solutie: izolarea partii de SQL de cea scrisa de catre programator in limbajul folosit pentru restul aplicatiei (C#, Java etc.)
- Beneficii:
  - specializarea pe domenii: cod separat in limbajele C#/Java si SQL
  - posibilitatea de trecere de la un tip de BD la altul
  - acces simplificat la mecanismele de imbunatatire a performantei de catre administratorii de baze de date, fara a fi nevoie sa intre in codul de logica de domeniu, UI etc.
- Rezultat: decuplarea implementarii logicii aplicatiei fata de accesul la date

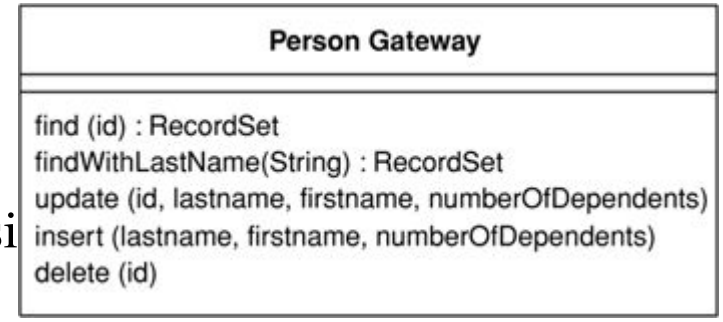
Pattern-uri arhitecturale: Row Data Gateway

- Concret: se obtine un **Data Gateway** (pattern arhitectural) care permite accesarea datelor
- Mai multe moduri de implementare de Gateway:
  - O instanta pentru fiecare inregistrare care e returnata de o interogare = **Row Data Gateway**
  - Un **Row Data Gateway** se potriveste (intr-o masura mica) cu modelarea obiectuala (Domain Model) din stratul de modelare a logicii de domeniu



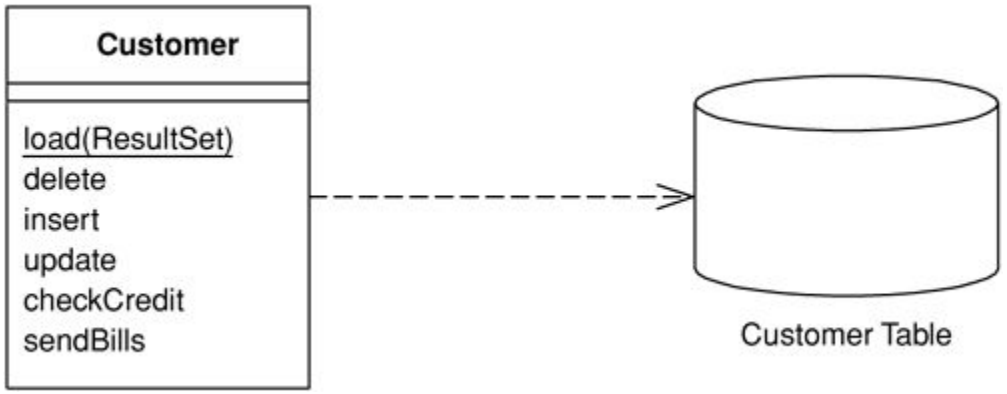
Pattern-uri arhitecturale: Table Data Gateway

- Alta modalitate:
  - Record Set: o structura de date generala pentru reprezentarea structurii tabelelor si retinerea inregistrarilor
  - Poate fi folosit in orice parte a aplicatiei, inclusiv in GUI pentru data binding
  - Pentru fiecare tabela exista o clasa => instante cu colectii de inregistrari
  - Rezultat: **Table Data Gateway** care manipuleaza un Record Set
  - Table Data Gateway se potriveste natural cu Table Module din Domain Logic Layer



Pattern-uri arhitecturale: legarea cu Domain Logic (1)

- Daca in Domain Logic (DL) se foloseste Domain Model (DM):
  - Se pot folosi row data gateway si table data gateway, dar ambele sunt considerate mai degraba nepotrivite cu DL
  - Varianta: daca domain model are cate o clasa pentru fiecare tabel (adica: un obiect din DM reprezinta o inregistrare dintr-o tabela anume), atunci fiecare domain object poate deveni responsabil pentru lucrul cu SQL => **Active record**
  - Efect echivalent: se pleaca cu un **Row Data Gateway** la care se adauga logica de domeniu

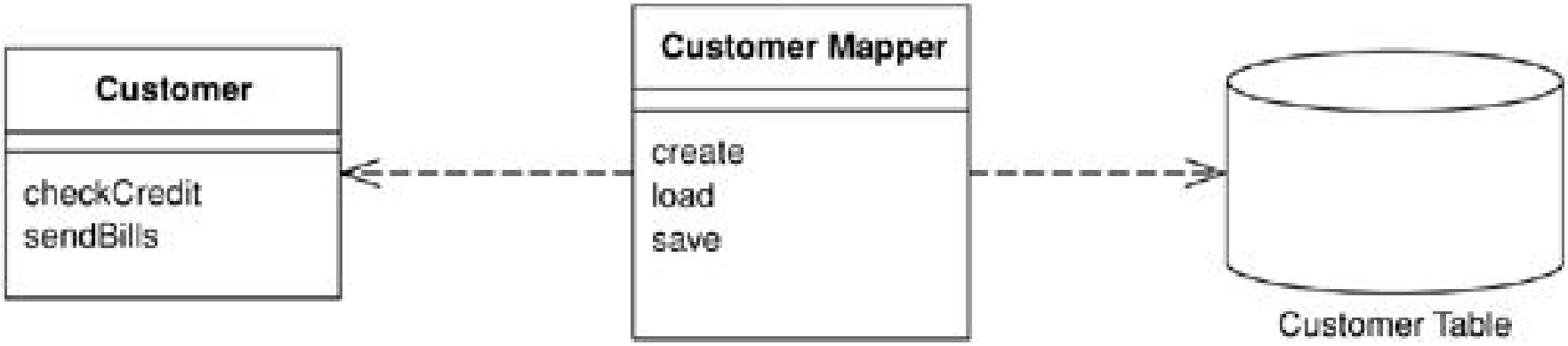


Pattern-uri arhitecturale: legarea cu Domain Logic (2)

- Active record devine inutil daca Domain Model este complex sau daca nu este asociere 1:1 intre clasele de modelare si tabelele din BD
- Exemplu: daca in domain model se foloseste mostenirea, in tabele acest lucru este de regula inaplicabil
  - Exceptie remarcabila: PostgreSQL permite mostenirea intre tabele
- Daca Domain Model este complex, atunci folosirea oricarei forme de Gateway inseamna cuplare cu structura bazei de date - nedorit

Pattern-uri arhitecturale: legarea cu Domain Logic (3)

- Izolarea totala a Domain Model-ului de stratul de acces la date: **Data Mapper**
- Data Mapper izoleaza total obiectele de domeniu de baza de date:



- Dar pattern-urile nu sunt mutual excluzive:
  - Data Mapper pentru legatura cu Domain Model
  - Un tip de **Gateway** pentru expunerea datelor catre alte servicii

Diverse

- In loc de tabele, pot fi folosite view-uri
  - Demo view
- …sau proceduri stocate
- …sau interogari dinamice
- = 3 modalitati de ascundere a structurii tabelelor
- Potentiale probleme: atentie mare la update-uri: view-urile sunt vederi virtuale asupra datelor, deci operarea trebuie facuta de regula pe mai multe tabele
- Exista varianta folosirii de BD orientate obiect – rar utilizate, totusi
- Utilizarea de unelte Object/Relational Mapping (O/RM)

Aspectul comportamental (1)

- Intrebare: cum faci sa se incarce si salveze/modifice/stearga diferite obiecte din/in baza de date?
- Daca incarci in memoria RAM un set de inregistrari, cum pastrezi statusul in care ele se afla? Pot interveni stergeri, actualizari, modificari
- Sunt 2 locuri in care datele exista: in RAM, de exemplu sub forma de obiecte si in BD – sub forma de inregistrari
- Trebuie asigurata consistenta intre cele doua locatii
- Ex: daca citesti un obiect din BD, cum te asiguri de faptul ca o modificare de catre alt client a aceleiasi inregistrari nu va trece neobservata? = problema accesului concurent

Aspectul comportamental (2)

- Pastrarea statusului obiectelor si controlul concurentei pot fi rezolvate de un pattern: **Unit of Work**
- UoW pastreaza starea tuturor obiectelor incarcate din baza in RAM, impreuna cu toate obiectele modificate in vreun fel
- Programatorul nu cere update pentru fiecare obiect in parte…
- …dar cere lui UoW sa comita toate modificarile in BD; UoW apeleaza insert/update/delete, in functie de starea pe care o au obiectele gestionate

Aspectul comportamental (3)

- Problema: ce se intampla daca incarci acelasi obiect (adica provenind din aceeasi inregistrare) de doua ori in RAM?
- Modificarea celor doua instante ar cere modificarea aceleiasi inregistrari -> situatie confuza
- Solutie: se pastreaza lista inregistrarilor citite intr-un **Identity Map** (pattern)
- Identity Map este consultat ori de cate ori se vrea a se face o citire din baza de date
  - Daca el confirma ca anumite inregistrari sunt deja citite, le va returna pe acestea din RAM, prin referinta
  - Rezultat: modificarea va lucra pe o singura instanta a fiecarei inregistrari

Aspectul comportamental (4)

- Alt beneficiu al Identity Map: caching-ul datelor; citirea repetata a acelorasi date din BD poate beneficia de faptul ca o parte din date sunt deja incarcate in memorie
  - acesta este un efect colateral, nu scopul principal al componentei de Identity Map
- Alta problema: pentru un Domain Model bogat, sa presupunem ca pentru un client vrei sa aduci si lista de comenzi facuta
- Fiecare comanda se leaga de produse, acestea de tipuri de produse, discount-uri etc.
- Rezultat: trebuie incarcat un graf de obiecte enorm in RAM => problema legata de performanta si presiunea asupra memoriei RAM + trafic mare de date intre aplicatie se serverul de baze de date

Aspectul comportamental (5)

- Solutie: Lazy Loading
  - Incarca datele satelit doar cand ai nevoie cu adevarat de ele
  - Se pastreaza un “placeholder” care poate fi completat mai tarziu, daca se cere acces la datele respective
  - Se aduce strictul necesar pentru lucrul imediat cu datele
  - Discutie la laborator, pattern-ul de design Proxy

Citirea datelor (1)

- Necesita metode care sa aduca datele in functie de anumite criterii
- Exemple:
  - find(id)
  - findForCustomer(customer)
- Unde scrii aceste metode?
- Pentru table data gateway: in clasa asociata tabelei/view-ului/procedurii stocate
- Pentru row data gateway:
  - Pseudo-solutie: metode statice (de ce?) in clasa asociata unei inregistrari
  - Problema cu metodele statice: nu poti folosi polimorfismul de mostenire, nici mocking

Citirea datelor (2)

- Solutie: folosirea de clase “finder” separate
- Fiecare clasa are implementari dedicate pentru aducerea datelor
- Se pot folosi pentru testare: clase finder care simuleaza aducerea din sursa de date, dar nu se leaga de fapt la aceasta (Service Stub, mocking)
- Alte sfaturi:
  - Citirea datelor poate fi costisitoare (round trip time pana la BD)
  - E deci nerecomandat sa se faca citiri multiple pentru aducerea de date
  - Recomandat: citire “la gramada” (e.g. jonctiuni, multiple active record sets = MARS, https://www.c-sharpcorner.com/uploadfile/sudhi.pro/multiple-active-result-sets-mars/)

Citirea datelor (3)

- Alte sfaturi (continuare):
  - Folosire de jonctiuni: un Data Mapper poate forma mai multe obiecte din DM
  - Dar o jonctiune pe prea multe tabele poate insemna penalizare pe serverul de BD
  - Varianta: folosirea mecanismelor de caching
    - O parte din date s-ar putea sa fie deja prezente in RAM, in memoria serverului de aplicatii
  - La interventii majore asupra modului de operare pe BD: profiling, masurarea performantelor, calculul impactului asupra performantei totale folosind legea lui Amdahl

Pattern-uri de asociere structurala: asocierea relatiilor (1)

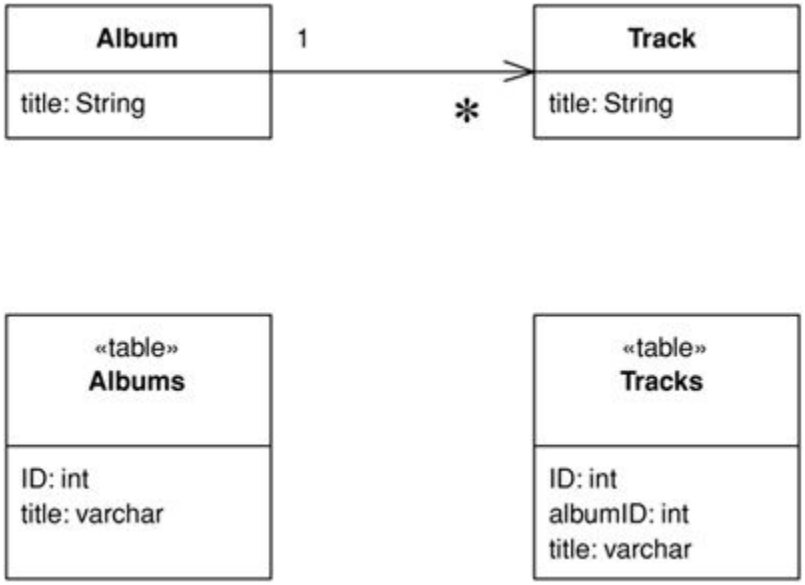
- Intrebare: cum se face legatura intre datele aflate in memorie si cele din BD?
- Asocierea relatiilor
  - Intre obiecte: folosind referinte/pointeri
  - Intre inregistrari: folosind chei straine
  - In modelarea obiectuala: un obiect poate mentine o colectie de alte obiecte asociate (client -> comenzi)
  - In modelarea relationala: forma normala 1 obliga la crearea de tabele dedicate ce stocheaza valorile multiple, sub forma de inregistrari
  - => discrepante intre modelarea legaturilor din cele 2 lumi

Asocierea relatiilor (2)

- Solutie:
- **Identity Field** (pattern) permite refacerea legaturii intre obiect si inregistrarea din baza de date
- Identity Field este folosit de catre **Identity Map**
- Pentru legatura intre obiecte: **Foreign Key Mapping**
- Legatura dinspre partea many spre partea one

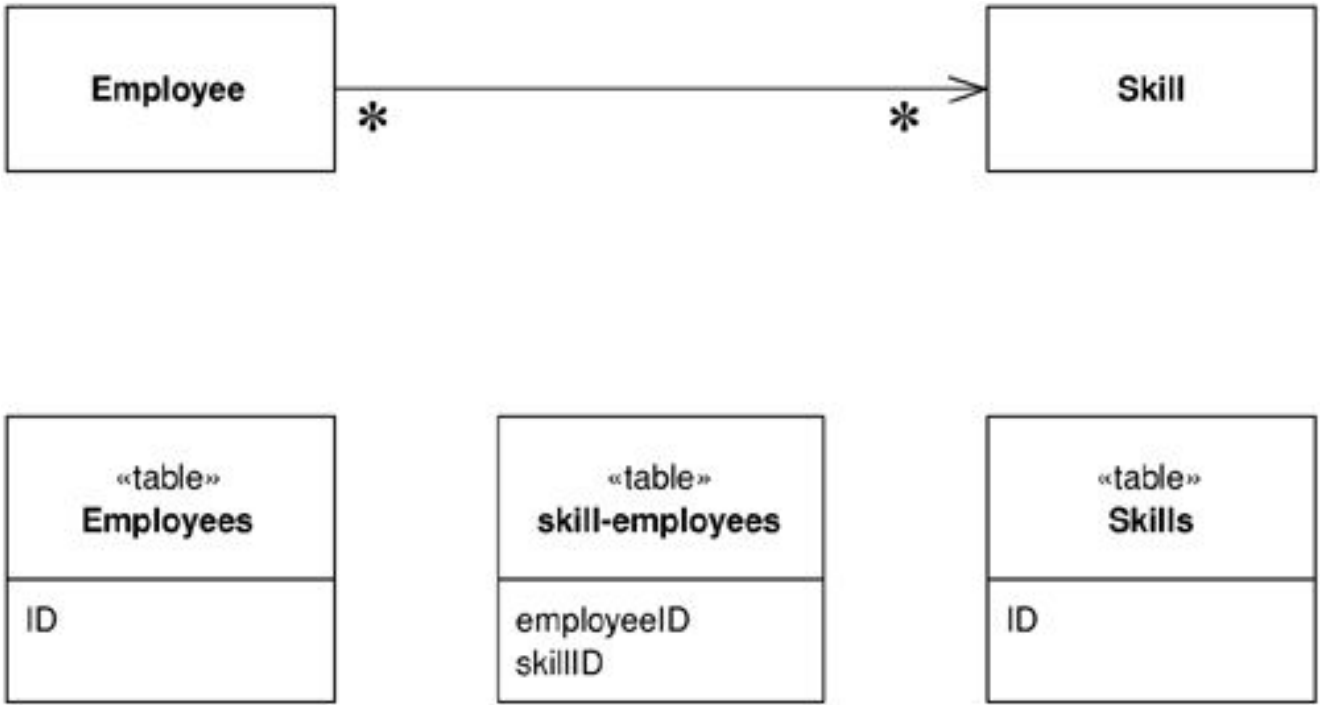
Asocierea relatiilor (3)

- Daca la o cautare a datelor **Identity Map** nu contine cheile cautate pentru acea tabela, atunci una din variantele
  - a. incarcare din baza de date
  - b. lazy loading, in diferite forme
- Alta varianta: obiectul din partea "one" contine colectia tuturor obiectelor din partea "many" = Dependent Mapping



Asocierea relatiilor (4)

- Problema: asocierea many-to-many
- Se foloseste un Association Table Mapping pentru a face o asociere M:N



Asociere structurala

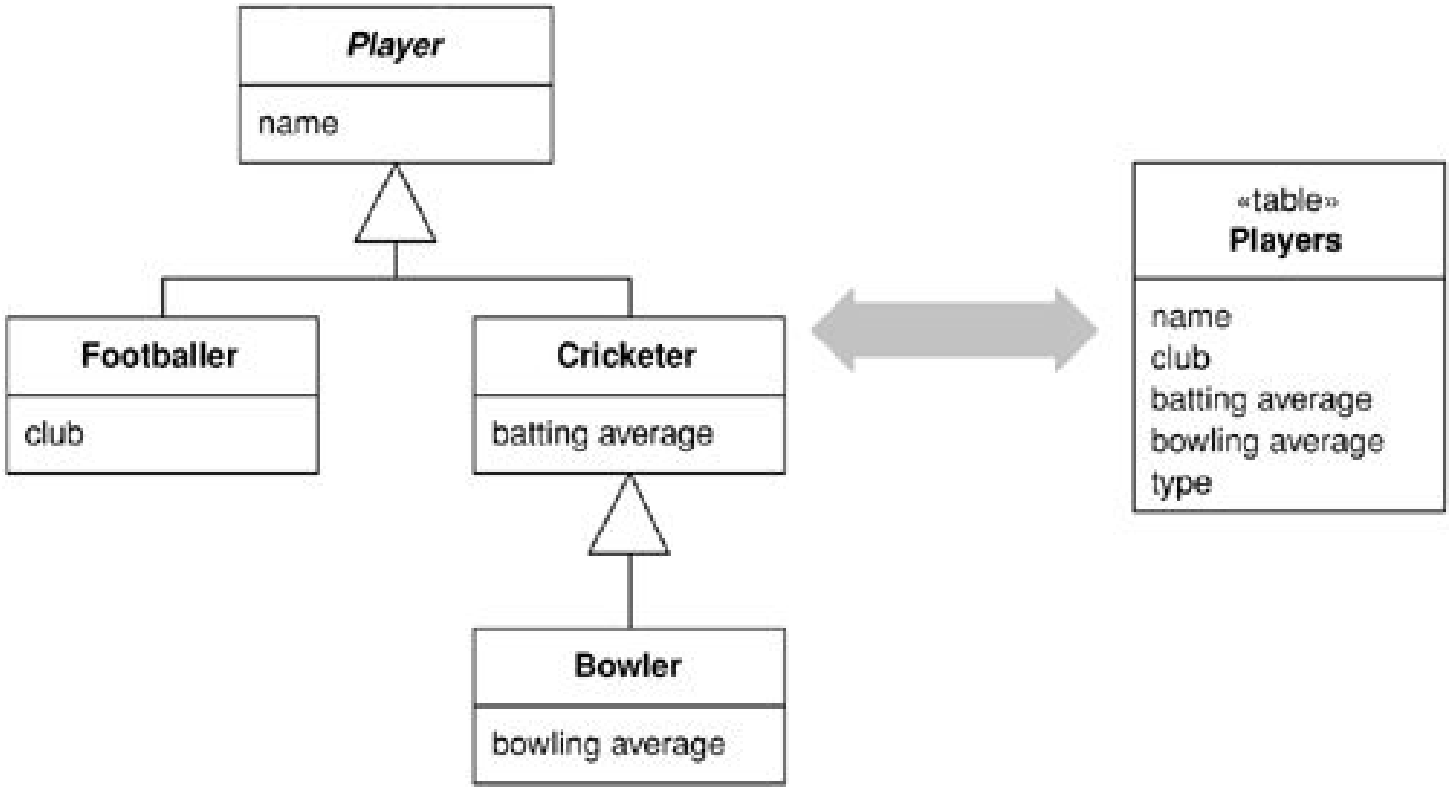
- Cazuri in care poti sa nu folosesti Identity Field:
  - Value objects, de exemplu valori monetare (suma + moneda), date sau intervale calendaristice
  - Salvarea unui intreg graf de obiecte intr-o celula dintr-o tabela: Large Objects
    - Format text (e.g. XML, JSON): Character LOB = CLOB
      - Exemplu: baza de date AdventureWorks2017, tabela JobCandidate coloana Resume
    - Format binar: Binary LOB = BLOB
    - Avantaj: nu este nevoie de jonctiune (join) peste multiple tabele pentru reconstruirea datelor, ci e suficienta o singura interogare + parsing/deserializare
    - Dezavantaj: standardul SQL nu ofera suport pentru lucrul cu datele stocate in stil LOB
    - Exceptii: SQL Server 2005+ poate "vedea" intr-un XML salvat intr-o celula; similar pentru Oracle 10g R2+
    - Protobuf, https://developers.google.com/protocol-buffers/

Mostenirea

- Ierarhii de clase legate prin derivare
- Lipsa de standard SQL pentru manipularea derivarilor
- 3 optiuni
  - Single table inheritance: o singura tabela pentru toata ierarhia
  - Concrete table inheritance: o tabela pentru fiecare clasa concreta
  - Class table inheritance: o tabela pentru fiecare clasa

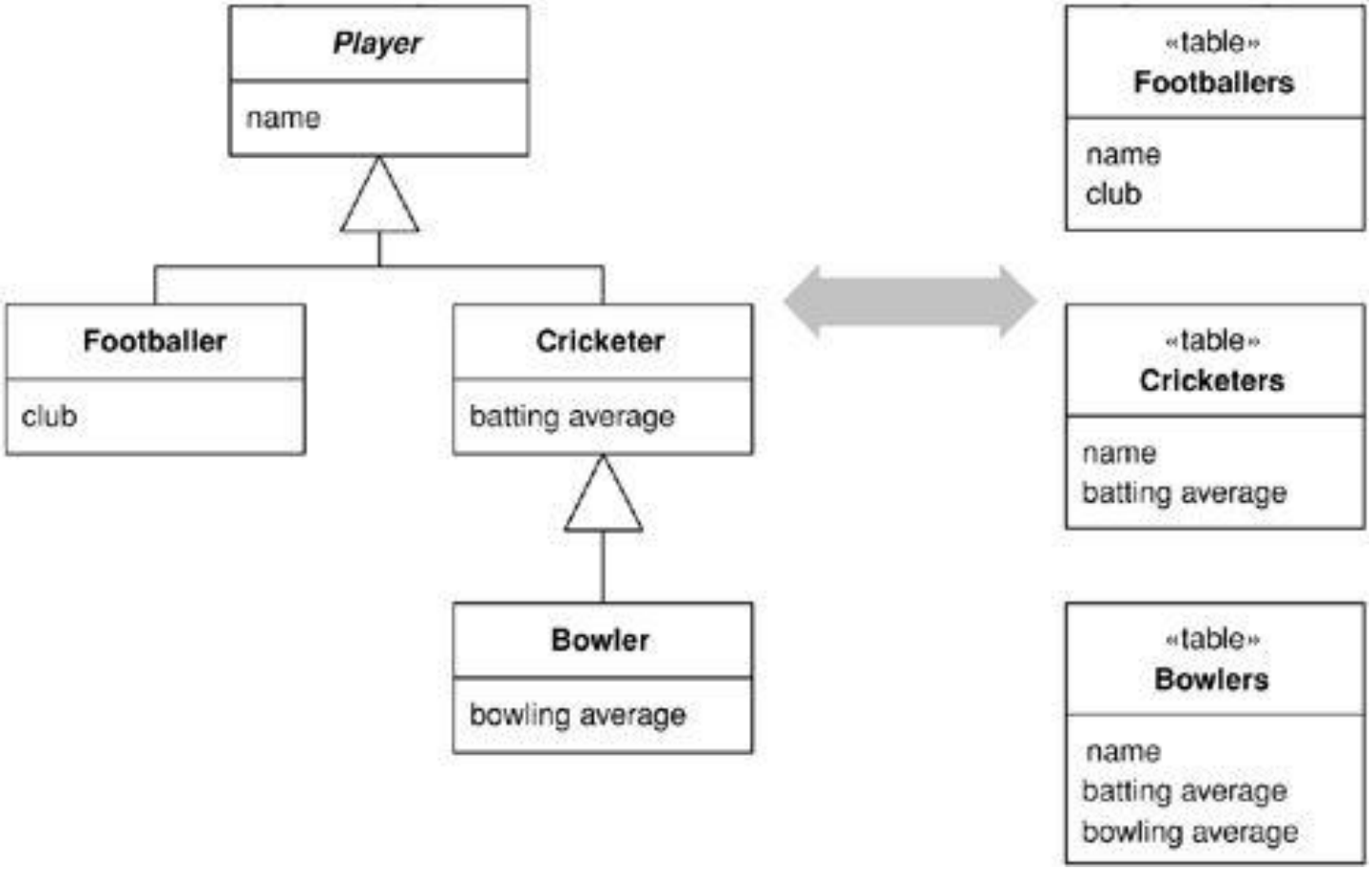
Single table inheritance

- Proprietati/Avantaje/Dezavantaje?



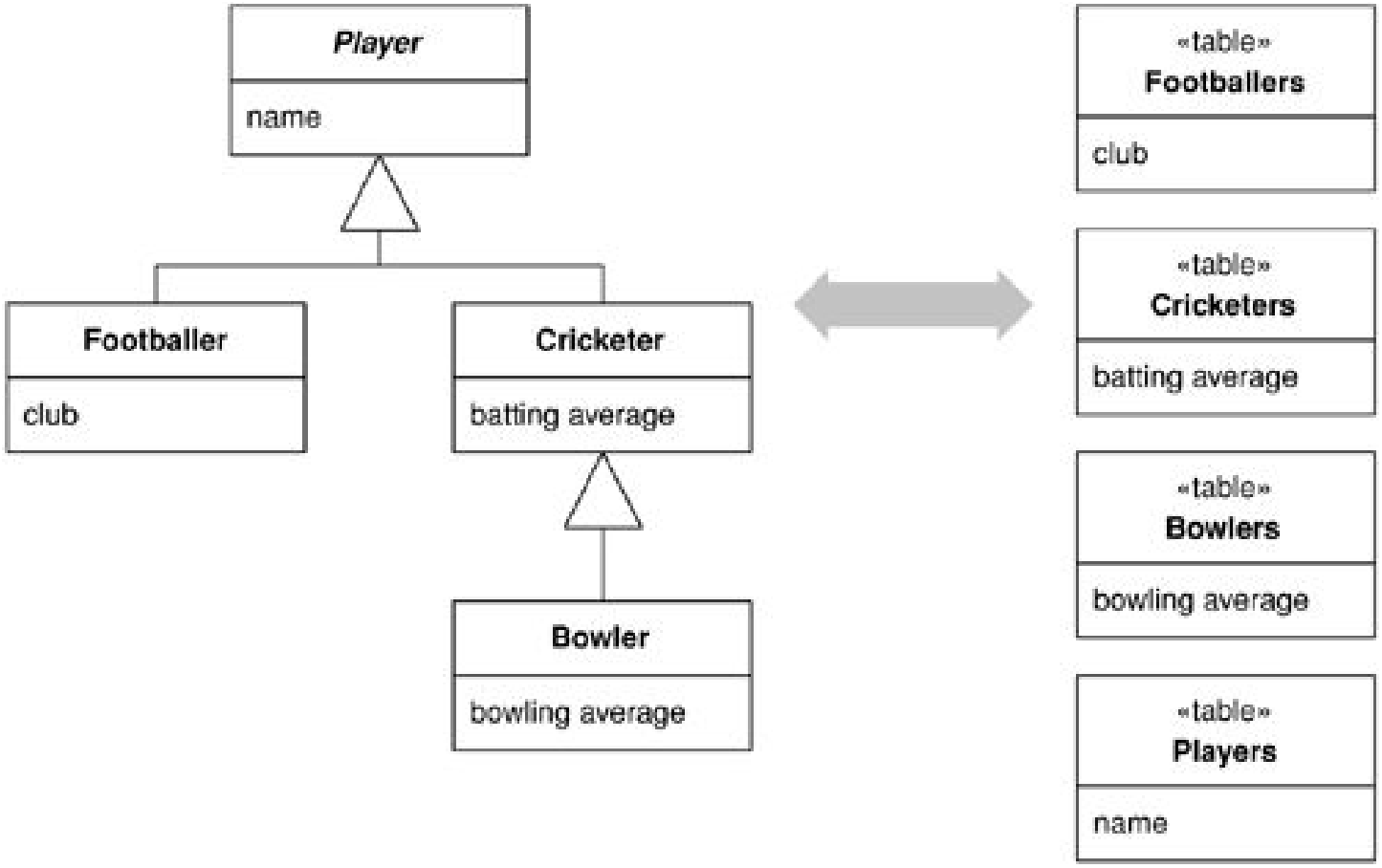
Concrete table inheritance

- Proprietati/Avantaje/Dezavantaje?



Class Table Inheritance

- Proprietati/Avantaje/Dezavantaje?



Construirea asocierii (1)

- Asocierea obiectual – relationala – situatii vis-à-vis de BD:
  - Poti sa creezi structura bazei de date
  - Primesti structura si aceasta nu se poate modifica
  - Primesti structura si poti negocia modificari
- Tu creezi structura BD
  - Daca ai complexitate moderata =>Transaction script sau Table Module in Domain Layer, Row Data Gateway sau Table Data Gateway in Data access layer
  - Domain Model => nu trebuie ca designul obiectual sa repete ce e in baza de date (Active Record se ocupa de asta); bazeaza-te pe un Data Mapper
  - Specific: iteratii scurte in care se construiesc DM si BD in paralel

Construirea asocierii (2)

- Cand schema e data:
  - Similar cu ce e mai sus, dar cu minim/deloc interventie pe structura bazei de date
  - Domain logic simplu => Row Data Gateway sau Table Data Gateway
  - Domain logic complex => Domain Model si Data Mapper
- Alta situatie: existenta mai multor surse de date ce trebuie folosite concomitent
  - Simplist: straturi de asociere pentru fiecare sursa de date => se poate ajunge la duplicare de cod
  - Recomandat: 2 niveluri: conversie obiecte <–> reprezentare logica (unificata) si reprezentare logica <-> reprezentari concrete

Metadate

- < targetTable="customers" targetClass = "Customer" dbColumn = "custID" lowerBound = "1" upperBound = "1" setter = "loadCustomer"/>
- Se pot folosi generatoare de cod care pe baza unei astfel de declaratii sa creeze automat cod pentru interogari si jonctiuni
- Constructia se face de catre un **Query Object**
- Modificarile care se fac in baza de date cer modificari in metadate, dar Query Object se va adapta singur
- Exemplu: (N)Hibernate, Entity Framework

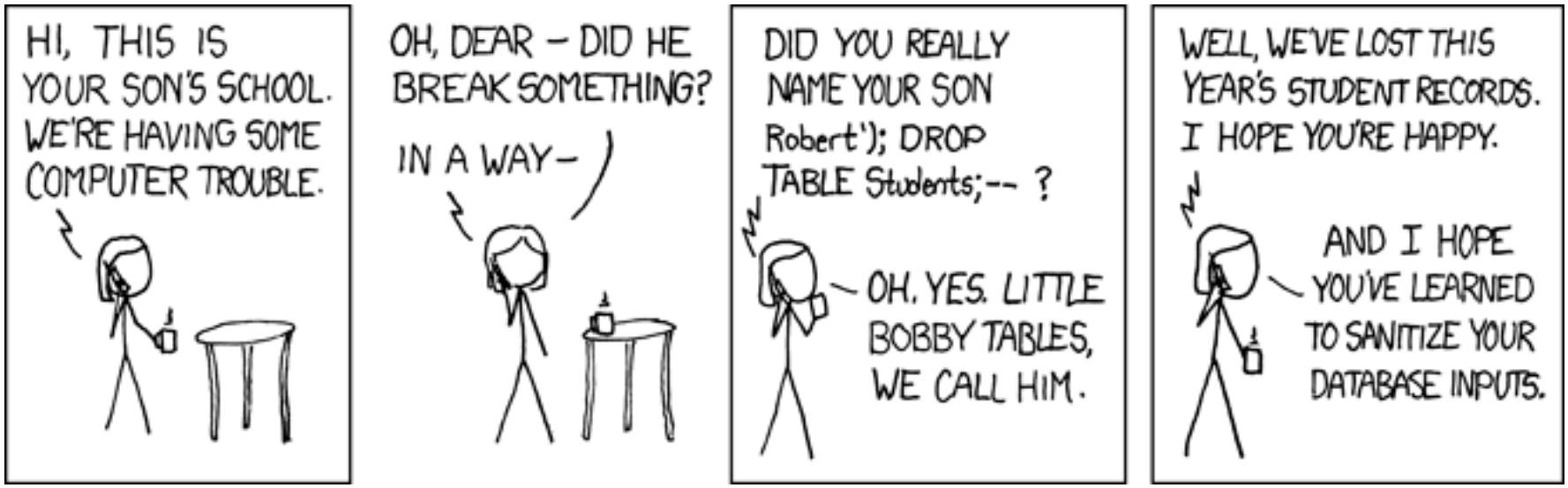
Conexiuni la baze de date

- Conexiune: obiect care permite dialogul cu serverul de BD
- Evident, pe toata durata dialogului cu baza de date conexiunea trebuie mentinuta deschisa
- Se poate lucra conectat sau deconectat
- Crearea de conexiuni poate fi costisitoare => connection pool, creat si gestionat manual sau automat (*e.g.* ADO.NET)
  - Inchiderea conexiunii duce la returnarea in colectia de conexiuni, nu la distrugerea obiectului de conexiune
- Obligatoriu: asigurarea inchiderii conexiunii in cod (blocuri garantate in C#/Java: finally, using, destructor)
- Legare a conexiunilor de tranzactii
- Conexiunile pot fi gestionate de Unit of Work

Diverse

- Nerecomandat: select * from tabela
  - Legatura intre numele de coloane din tabela si cele dintr-un Table Data Gateway
- Nerecomandat: select col1, col2, … from tabela fara clauza orderby sau fara ordonare in straturile superioare
  - Demo: execution plan
- Se recomanda folosirea de interogari precompilate (*e.g.* proceduri stocate)
  - Dezbateri aprinse pe tema: ex http://weblogs.asp.net/fbouma/archive/2003/11/18/38178.aspx
- In nici un caz: concatenare de stringuri pentru creare de interogare SQL => reteta sigura pentru SQL Injection

SQL Injection



- Sursa: http://xkcd.com/327/

Teme

- Documentati-va pe: “proceduri stocate” vs. “interogari dinamice”
- Documentati-va pe: Lista de biblioteci O/RM pentru .NET 4.7, Core? – de raspuns la lucian.sasu@yahoo.com
- Ce este NoSQL? In ce scenarii poate fi preferat fata de SGBD-urile relationale clasice?