Arhitectura sistemelor soft enterprise. Platforma .NET

Curs 7

Pattern-uri arhitecturale pentru stratul de sursa de date

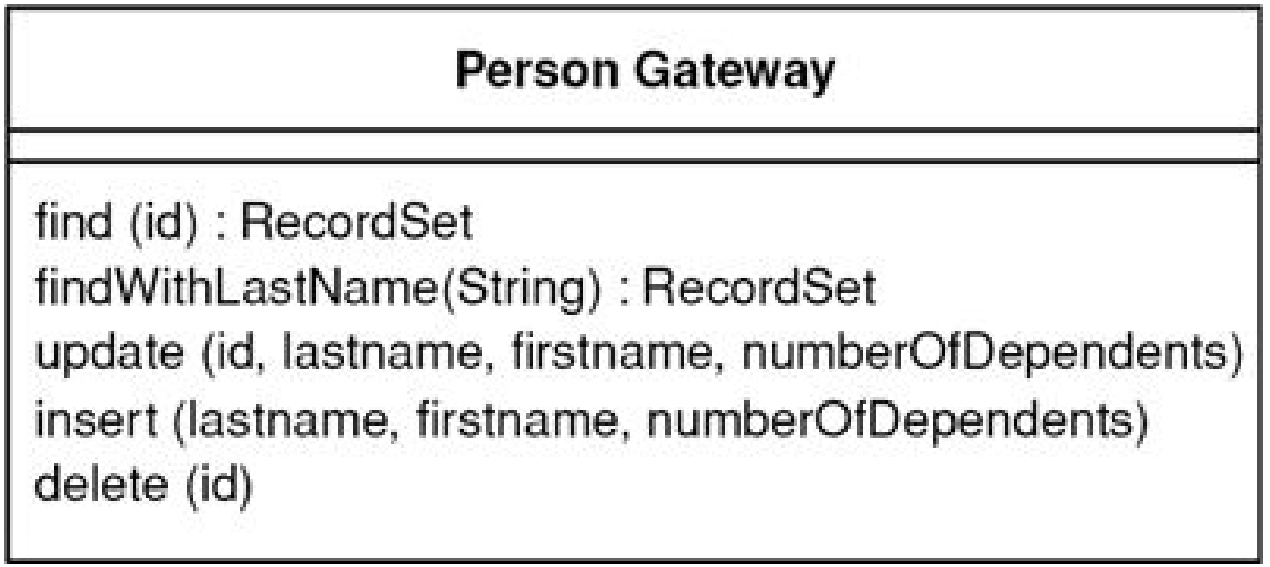
Pattern-uri arhitecturale pentru stratul de sursa de date

- Table data gateway
- Row data gateway
- Active record
- Data mapper

- Sursa: Cap 10 din PoEAA “Data Source Architectural Patterns”

Sabloane arhitecturale pentru surse de date

- Table Data Gateway: obiect care actioneaza ca o poarta de acces (gateway) catre un tabel sau un view din baza de date
  - O instanta manipuleaza un set de inregistrari dintr-o tabela/view
- Metode publice:



Sabloane arhitecturale pentru surse de date

- Un Table Data Gateway mentine toate frazele SQL sau numele de proceduri stocate pentru operatiile CRUD
- Beneficiu: izolarea codului SQL fata de codul scris in limbajul de implementare a aplicatiei (C#, Java); multi programatori nu stapanesc foarte bine limbajul SQL, multi nu stiu sa faca optimizarea codului SQL
- Codul de SQL fiind izolat de partea de logica a domeniului, poate sa fie modificat fara a afecta restul de implementare
  - fiind cod SQL scris separat, el poate fi urmarit si mentinut de catre un dezvoltator cu cunostinte bune de SQL; BD se poate configura corespunzator (indecsi, partitionare de tabele, mutarea fisierelor ce apartin BD pe diferite harddisk-uri etc.)

Sabloane arhitecturale pentru surse de date

- Table Data Gateway - mod de lucru:
  - Interfata simpla, de regula constand in cateva metode de tip “find” pentru a gasi date dintr-o tabela = apel de fraze de tip “select”
  - + metode de update, insert, delete – posibil mai multe implementari pentru fiecare tip de operatie
  - Cum se returneaza datele din interogare?

Sabloane arhitecturale pentru surse de date

- Metoda “find by ID”: desi primeste ca parametru valoare de cheie primara, este vazuta ca si cand ar putea returna mai multe inregistrari
- Avem deci nevoie de stocare de mai multe rezultate in ce returneaza functia
- Alternativa: dictionare (hashtable, dictionary) cu cheia din dictionar = numele coloanei; posibila pentru cazul in care se returneaza o singura inregistrare, cu copiere a datelor care rezulta din interogare in dictionar
- Probleme:
  - slaba verificare la compilare, chiar daca se folosesc dictionare generice – pe cat posibil

Sabloane arhitecturale pentru surse de date

- Alternativa: Data Transfer Object; avantajul e ca se pot folosi mai departe in context de apel la distanta sau ca tipuri de date de comunicare intre alte straturi
- Alta varianta: RecordSet = colectie de inregistrari:
  - posibila problema: ce se intampla daca se schimba sursa de date (e.g. XML)?
  - reprezentarile din domain layer nu ar trebui sa fie conditionate de modul in care se face structurarea datelor in sursa de date
- Dar: in anumite implementari (vezi .NET DataSet), e posibil si chiar recomandabil in anumite scenarii, e.g acces deconectat de la baza de date
- Daca se foloseste Domain Model (DM), un Table Data Gateway trebuie sa manipuleze colectie de obiecte din DM; apare cuplare nedorita intre DM si schema bazei de date

Sabloane arhitecturale pentru surse de date

- Variatie de implementare: pentru cazuri foarte simple (= putine tabele, putine interogari) se admite sa existe un Table Data Gateway pentru toata baza de date
- E posibil ca un TDG sa se asocieze unor vederi (views)
- In special pentru view-uri sau jonctiuni de tabele (join-uri): un TDG poate ascunde complexitatea update-urilor pe tabele
- Metodele de tipul delete, update, insert din TDG ascund faptul ca operarea se face pe mai multe tabele; ele se vor traduce in operatii coerente pe mai multe tabele

Sabloane arhitecturale pentru surse de date

- Table Data Gateway - cand (nu) se foloseste:
  - Nenatural in conjunctie cu Domain Model: preferabil aici un Data Mapper
  - Lucreaza bine cu Table Module, pentru care produce un set de inregistrari pe care acesta il preia ca atare
  - Usor de utilizat si de Transaction Scripts
  - Pot fi utilizate ca furnizoare de servicii pentru o implementare de Data Mapper
  - Populare datorita mecanismului de data binding = legare declarativa a sursei de date cu un control ce permite afisarea lor

Sabloane arhitecturale pentru surse de date

- Exemple: (PoEAA, pag 116-120)

```
class PersonGateway…
public IDataReader FindAll() {
    String sql = "select * from person";
    return new OleDbCommand(sql, DB.Connection).ExecuteReader();
}

public IDataReader FindWithLastName(String lastName) {
    String sql = "SELECT * FROM person WHERE lastname = ?";
    IDbCommand comm = new OleDbCommand(sql, DB.Connection);
    comm.Parameters.Add(new OleDbParameter("lastname", lastName));
    return comm.ExecuteReader();
}

public IDataReader FindWhere(String whereClause) {
    String sql = String.Format("select * from person where {0}", whereClause);
    return new OleDbCommand(sql, DB.Connection).ExecuteReader();
}
```

Sabloane arhitecturale pentru surse de date

Sau:

```
class PersonGateway…
    public Object[] FindRow (long key) {
        String sql = "SELECT * FROM person WHERE id = ?";
        IDbCommand comm = new OleDbCommand(sql,
        DB.Connection);
        comm.Parameters.Add(new OleDbParameter("key",key));
        IDataReader reader = comm.ExecuteReader();
        reader.Read();
        Object [] result = new Object[reader.FieldCount];
        reader.GetValues(result);
        reader.Close();
        return result;
}
```

Sabloane arhitecturale pentru surse de date - insert

```
class PersonGateway…
public long Insert(String lastName, String firstName, long
    numberOfDependents) {
    String sql = "INSERT INTO person VALUES (?,?,?,?)";
    long key = GetNextID();
    IDbCommand comm = new OleDbCommand(sql, DB.Connection);
    comm.Parameters.Add(new OleDbParameter ("key", key));
    comm.Parameters.Add(new OleDbParameter ("last", lastName));
    comm.Parameters.Add(new OleDbParameter ("first", firstName));
    comm.Parameters.Add(new OleDbParameter ("numDep",
            numberOfDependents));
    comm.ExecuteNonQuery();
    return key;
}
```

Sabloane arhitecturale pentru surse de date

```
class PersonGateway…
    public void Update (long key, String lastname, String firstname,
    long numberOfDependents){
        String sql = @"
        UPDATE person
        SET lastname = ?, firstname = ?, numberOfDependents = ?
        WHERE id = ?";
        IDbCommand comm = new OleDbCommand(sql, DB.Connection);
        comm.Parameters.Add(new OleDbParameter ("last", lastname));
        comm.Parameters.Add(new OleDbParameter ("first", firstname));
        comm.Parameters.Add(new OleDbParameter ("numDep",
            numberOfDependents));
        comm.Parameters.Add(new OleDbParameter ("key", key));
        comm.ExecuteNonQuery();
    }
```

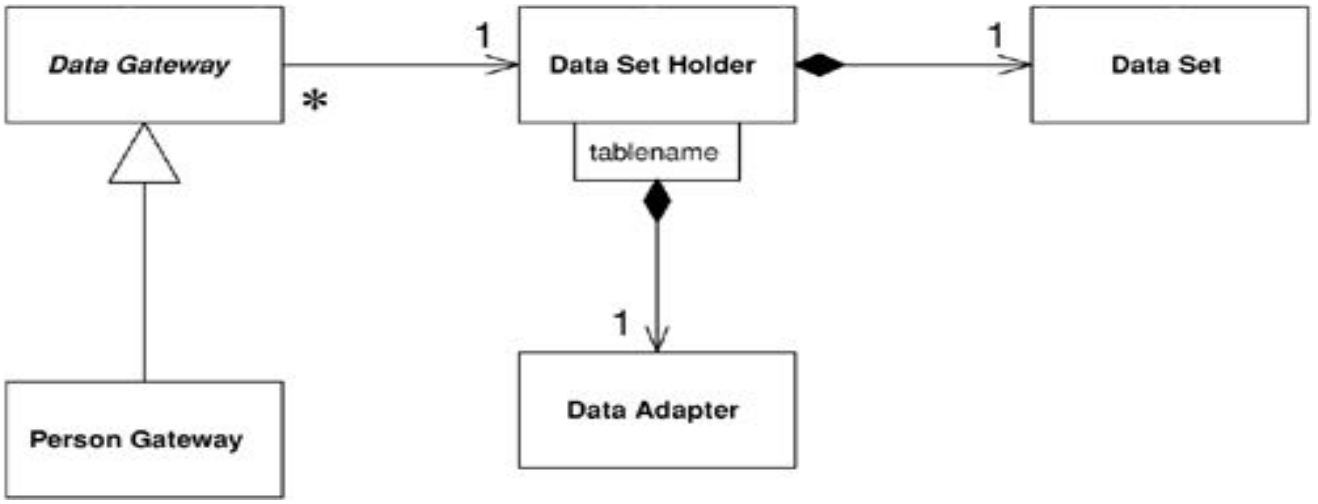
Sabloane arhitecturale pentru surse de date

```
class PersonGateway…
public void Delete (long key) {
    String sql = "DELETE FROM person WHERE id = ?";
    IDbCommand comm = new OleDbCommand(sql,
      DB.Connection);
    comm.Parameters.Add(new OleDbParameter ("key", key));
    comm.ExecuteNonQuery();
    //posibil: verificarea numarului de inregistrari sterse; daca e
      diferit de 1 => problema
}
```

Sabloane arhitecturale pentru surse de date

- Exemplu folosind ADO.NET DataSet

```
class DataSetHolder…{
    public DataSet Data = new DataSet();
    private Dictionary<string, DbDataAdapter> dataAdapters = new
    Dictionary<string, DbDataDataAdapter>();
}
```



Sabloane arhitecturale pentru surse de date

```
abstract class DataGateway…{
    public DataSetHolder Holder;
    public DataSet Data {
        get {return Holder.Data;}
    }

    protected DataSetGateway() {
        Holder = new DataSetHolder();
    }

    protected DataSetGateway(DataSetHolder holder) {
        this.Holder = holder;
    }
    //continuare in slideul urmator
```

Sabloane arhitecturale pentru surse de date

```
abstract public String TableName {get;}

public void LoadAll() {
    String commandString = String.Format("select * from {0}", TableName);
    Holder.FillData(commandString, TableName);//vezi slide urmator
    }

 public void LoadWhere(String whereClause) {
    String commandString =
    String.Format("select * from {0} where {1}", TableName, whereClause);
    Holder.FillData(commandString, TableName);
 }
}//sfarsit clasa DataGateway
```

Sabloane arhitecturale pentru surse de date

```
class PersonGateway : DataGateway…
    public override String TableName {
    get {return "Person";}
}

class DataSetHolder…
    public void FillData(String query, String tableName) {
    if (DataAdapters.Contains(tableName)) throw new
        MultipleLoadException();
    OleDbDataAdapter da = new OleDbDataAdapter(query,
        DB.Connection);
    OleDbCommandBuilder builder = new OleDbCommandBuilder(da);
    da.Fill(Data, tableName);
    DataAdapters.Add(tableName, da);
}
```

Sabloane arhitecturale pentru surse de date

Utilizare:

```
person.LoadAll();
person[key]["lastname"] = "Odell";
person.Holder.Update();
```

Linia a doua se bazeaza pe codul:

```
class DataGateway…
    public DataRow this[long key] {
    get {
        String filter = String.Format("id = {0}", key);
        return Table.Select(filter)[0];
        }
    }
    public override DataTable Table {
        get {return Data.Tables[TableName];}
    }
```

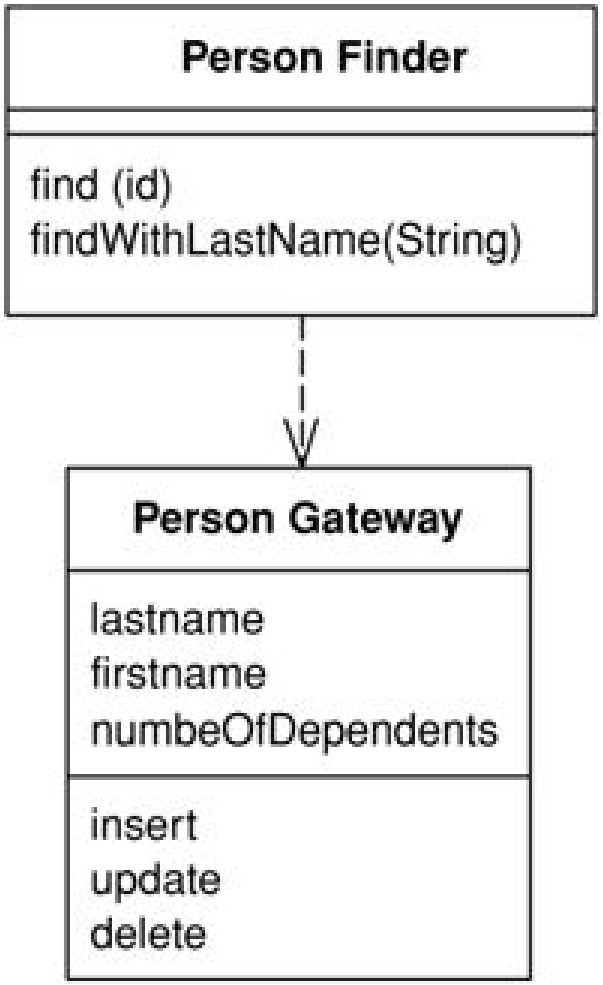
Sabloane arhitecturale pentru surse de date

…iar partea de update (linia a 3-a) se bazeaza pe:

```
class DataSetHolder…
   public void Update() {
        foreach (String table in DataAdapters.Keys)
   ((OleDbDataAdapter)DataAdapters[table]).Update(Data, table);
}
```

Row Data Gateway

- Un obiect care actioneaza ca o poarta de acces catre o singura inregistrare din baza de date
- Pentru fiecare inregistrare exista o instanta = un obiect



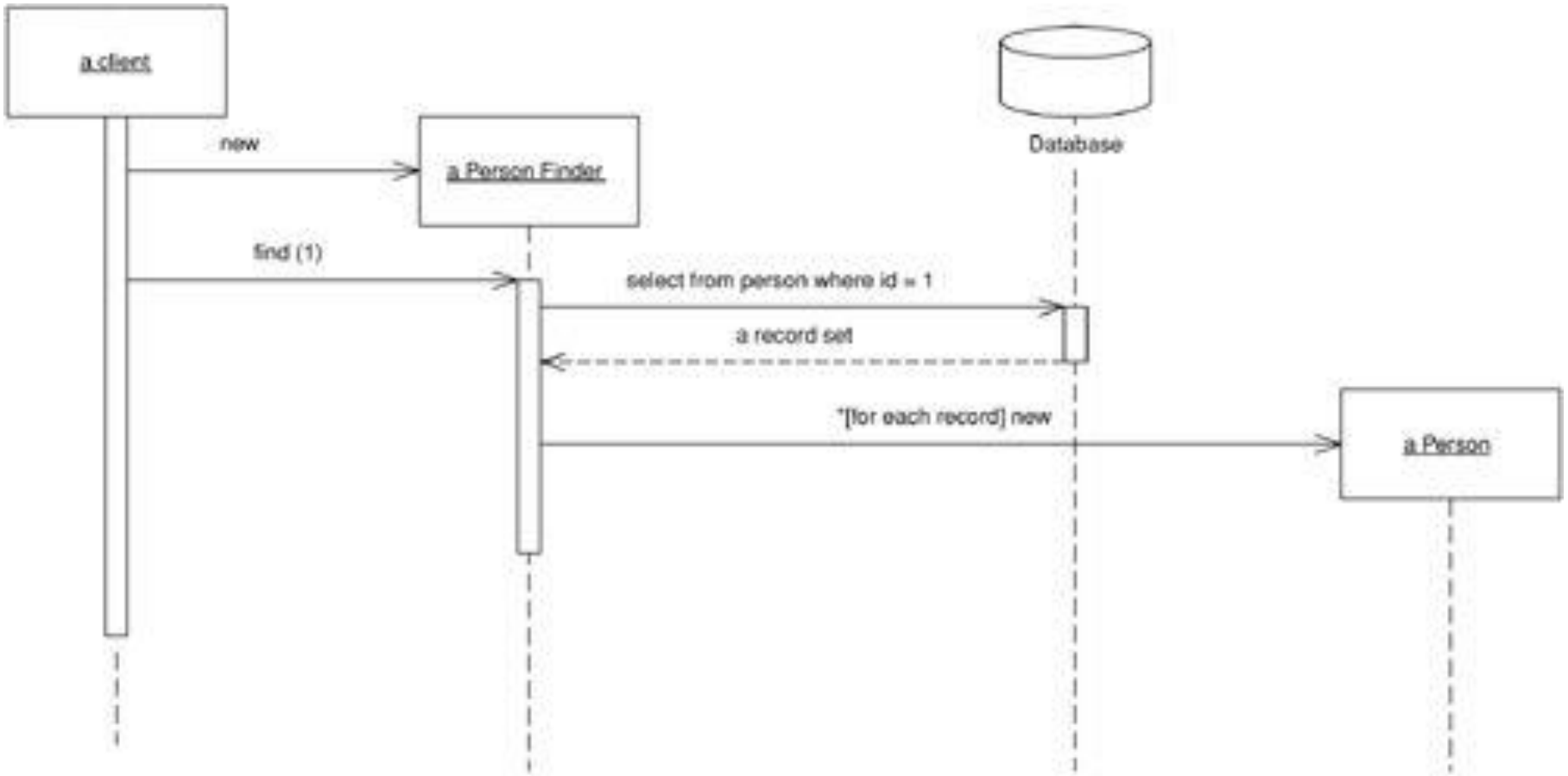
Row Data Gateway

- Un Row Data Gateway creeaza obiecte care arata exact ca inregistrarile dintr-o tabela/view, dar pot fi accesate prin mecanismele uzuale ale limbajului folosit (C#, Java etc.)
- Toate detaliile de implementare a accesului la date sunt ascunse in spatele acestei interfete
- Se poate face astfel implementare pentru mai multe servere de BD, proxy, mocking
- Se poate folosi usor de catre un Transaction Script

Row Data Gateway

- Cum se scrie:
  - Fiecare coloana din tabela devine un camp
  - Posibil conversii de tip de la reprezentarea din BD la cea din limbajul folosit pe partea de aplicatie
    - Exemplu: din varbinary(max) = tip de data in MS SQL Server se poate transforma in in obiect .NET de tip array de octeti/imagine/fisier multimedia si invers)
  - Rezultat: o interfata de acces a unei inregistrari din baza de date
- Unde se pun operatiile de tip “find”?
  - Se pot folosi metode statice, dar acestea nu mai pot fi suprascrise polimorfic => imposibilitate de suprascriere cu implementari diferite pentru surse de date diferite
  - Solutie: clasa concreta “finder” separata

Row Data Gateway



Row Data Gateway

- Se poate usor face confuzie intre Row Data Gateway si Active Record (vezi mai jos)
- Un Row Data Gateway contine doar metode pentru acces la BD si nicio metoda de tip Business Logic, spre deosebire de AR
- Se poate folosi un Row Data Gateway si pentru un view, pentru o interogare dinamica sau pentru o procedura stocata
- Numite in acest context: “virtual Row Data Gateways”
- Posibila problema: in acest caz partea de update/insert/delete poate sa devina complexa si greu de mentinut

Row Data Gateway

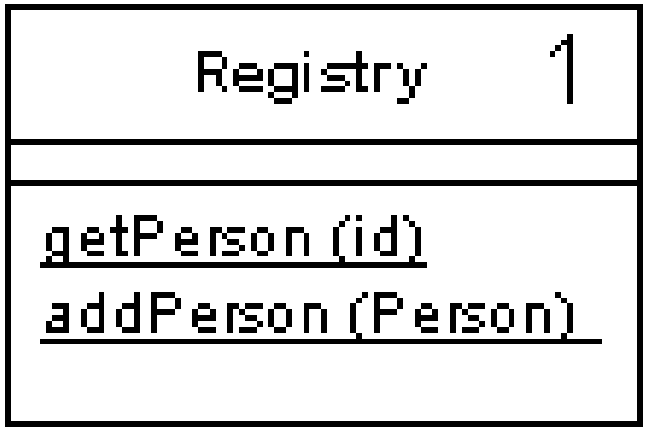
- Cand se foloseste:
  - Functioneaza bine cu Transaction Scripts
  - O astfel de clasa poate fi folosita de catre mai multe scripturi
  - Contraindicate in lucrul cu Domain Model:
    - Daca obiectele sunt simple, atunci Active Record este o alegere mai buna
    - Daca obiectele sunt complexe => recomandat Domain Model
  - De retinut: Row Data Gateway oglindeste structura bazei de date

Row Data Gateway

- Exista unelte care sa genereze clasele Row Data Gateway in mod automat
  - Mecanism: se pleaca de la metadate, e.g. schema unei baze de date
- Se pot folosi impreuna cu Data Mapper
- Daca se foloseste Row Data Gateway impreuna cu Transaction Scripts se observa ca logica de lucru cu datele (e.g. validari, calcul de valori derivate: valoare bruta, valoare totala) se repeta de-a lungul mai multor scripturi
- Solutie: mutarea acestei logici in clasele Row Data Gateway => Active Records

Row Data Gateway

- Exemplu de cod (Java): PoEAA, pag 122-124

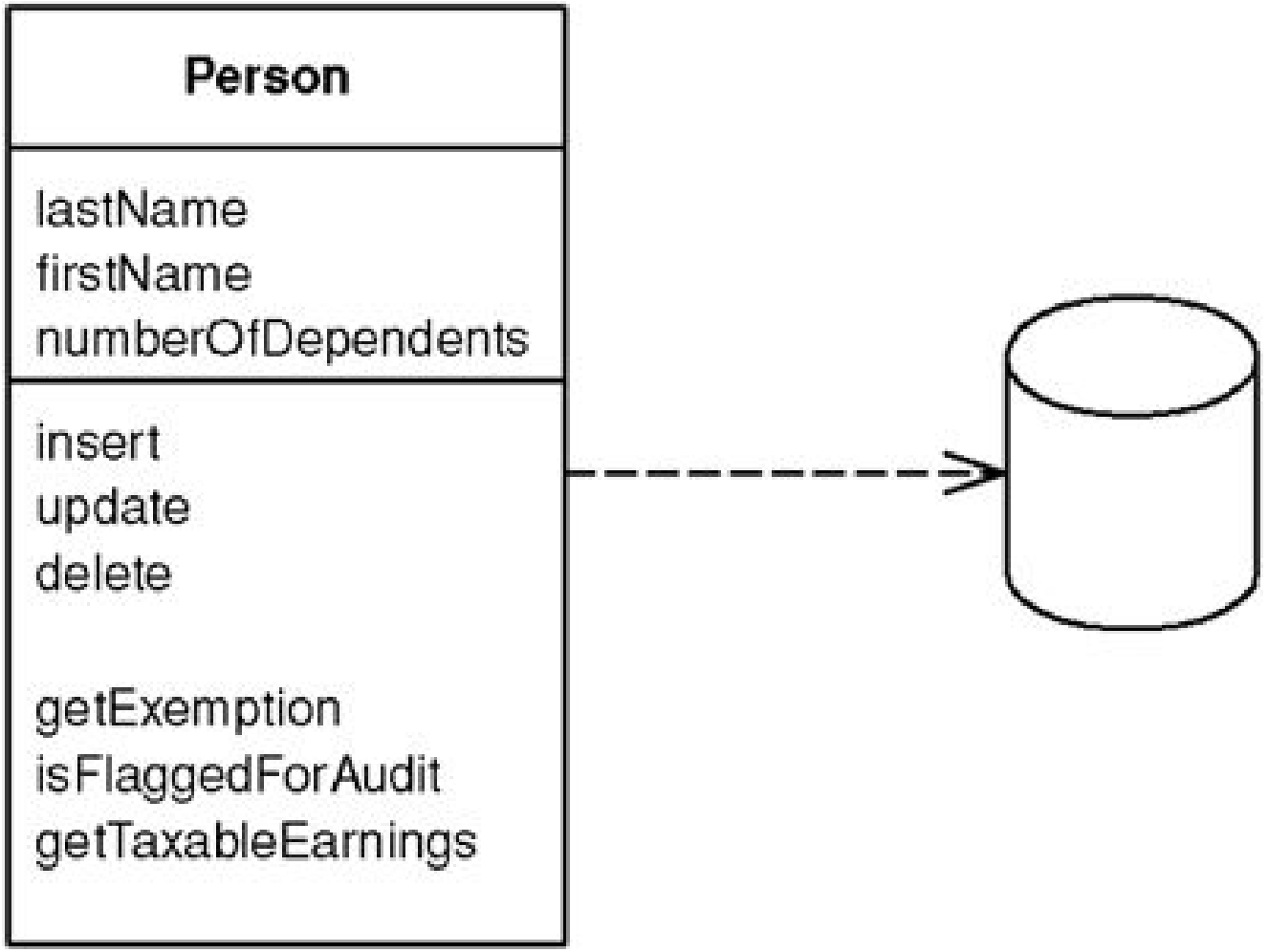


- Registry: un obiect singular care permite acces la obiecte
- Poate sa contina un Identity Map

Active Record

- Un obiect care reprezinta o inregistrare intr-o tabela sau un view, incapsuleaza accesul la baza de date si adauga logica de domeniu
- Un astfel de obiect contine atat date (campuri de inregistrare) cat si (comportament + operatii pe baza de date)

Active Record



Active Record

- Cand se foloseste:
  - Daca se pleaca de la un Domain Model in care obiectele sunt foarte apropiate de structura inregistrarilor din BD
  - Sau: daca se pleaca de la un Transaction Script in care se factorizeaza partea de logica continuta in scripturi
  - Un camp din clasa pentru fiecare coloana din tabela
  - Cheile straine sunt lasate asa cum sunt
  - Se pot folosi view-uri si interogari (proceduri stocate sau functii definite de utilizator), cu implementare specifica pentru create/update/delete

Active Record

- Metode tipic prezente intr-un Active Record:
  - Construieste o instanta de Active Record dintr-un rezultat SQL
  - Construieste o noua instanta pentru a fi mai tarziu inserata in tabela
  - Metode de tip “find” care contin interogarile SQL utilizate si returneaza obiecte Active Record – dar mai bine in clasa finder separate – subiect discutat deja
  - Metode de modificare a bazei de date si de inserare a unui Active Record
  - Proprietati (get/set) = boilerplate code
  - Elemente de business logic
    - valori totale
    - reduceri de pret
    - Modificare de stare pe baza unor calcule

Active Record

- Accesori get/set: pot face partea de conversie dintre reprezentarea in limbajul de pe partea de aplicatie si reprezentarea SQL
- Tot aici: un get pe o cheie straina poate sa returneze obiectul asociat pe baza datelor din alta tabela
- Metodele de tip find: pot fi izolate intr-o alta clasa => testare mai usoara, mocking, proxificare
- Principala diferenta fata de row data gateway: contin si business logic

Active Record

- Cand se folosesc:
  - Cand logica de domeniu nu este prea complexa (predomina creare, stergere, citire, modificari) + validari de date
  - Cand se lucreaza cu Domain Model, de regula se alege intre AR si Data Mapper
  - Daca DM este complex => simti nevoia de relatii/mosteniri intre clase, colectii de date
  - Grefarea acestor mecanisme peste AR duce la un design supraincarcat, deci AR sunt nerecomandate in context de Domain Layer complex

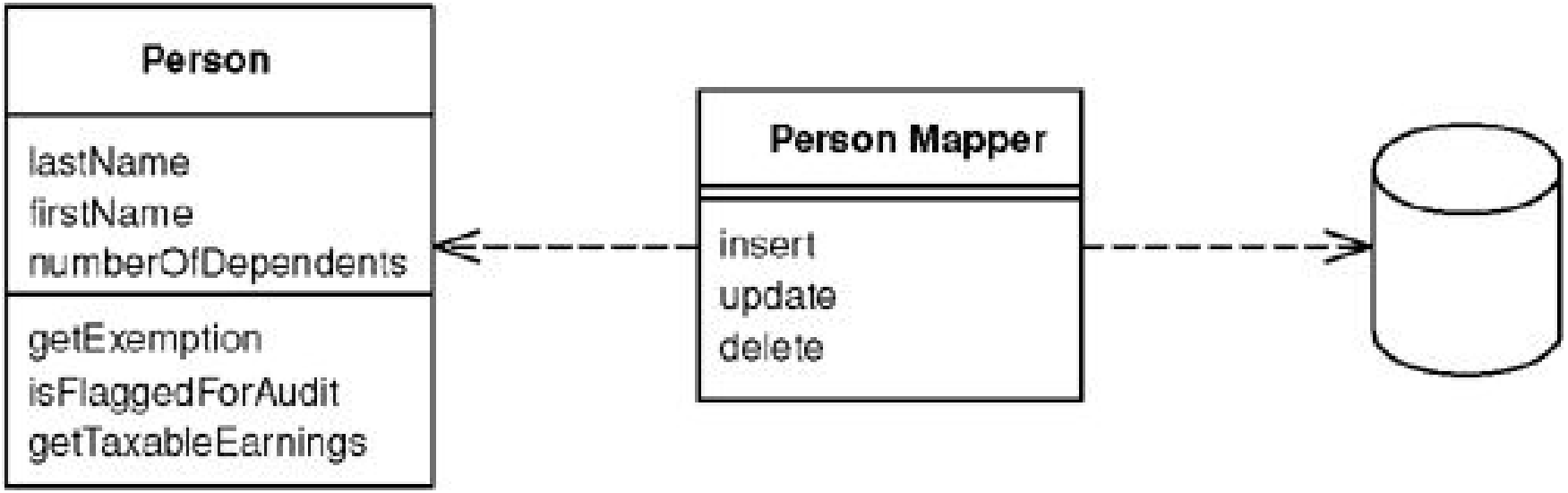
Active Record

- Un posibil argument impotriva AR: se cupleaza tipul obiectelor prea mult de structura bazei de date
  - Daca schema bazei de date este furnizata de client si nu se poate modifica si logica de domeniu este simpla atunci si clasele pot ramane fixate => se poate folosi AR
- In acest caz, refactoring-ul pe BD este ingreunat
- Cand e natural: cand se foloseste Transaction Scripts si se doreste factorizarea codului comun gasit in scripturi; in plus, partea de Domain Layer sa fie suficient de simpla pentru a se putea folosi un Transaction Script

Data Mapper

- Un strat care converteste datele dinspre reprezentare relationala spre reprezentare adecvata in Business Logic si invers
- Prin Data Mapper se obtine independenta intre Business Logic si sursa de date
- Se poate astfel face:
  - Refactoring obiectual
  - Refactoring pe baza de date
- Motivatie: obiectele si BD relationale au modalitati diferite de obtinere a designului: normalizare in BD care nu are corespondent in modelarea OO; mostenirea intre clase nu are corespondent relational (exceptie: PostgreSQL)

Data Mapper

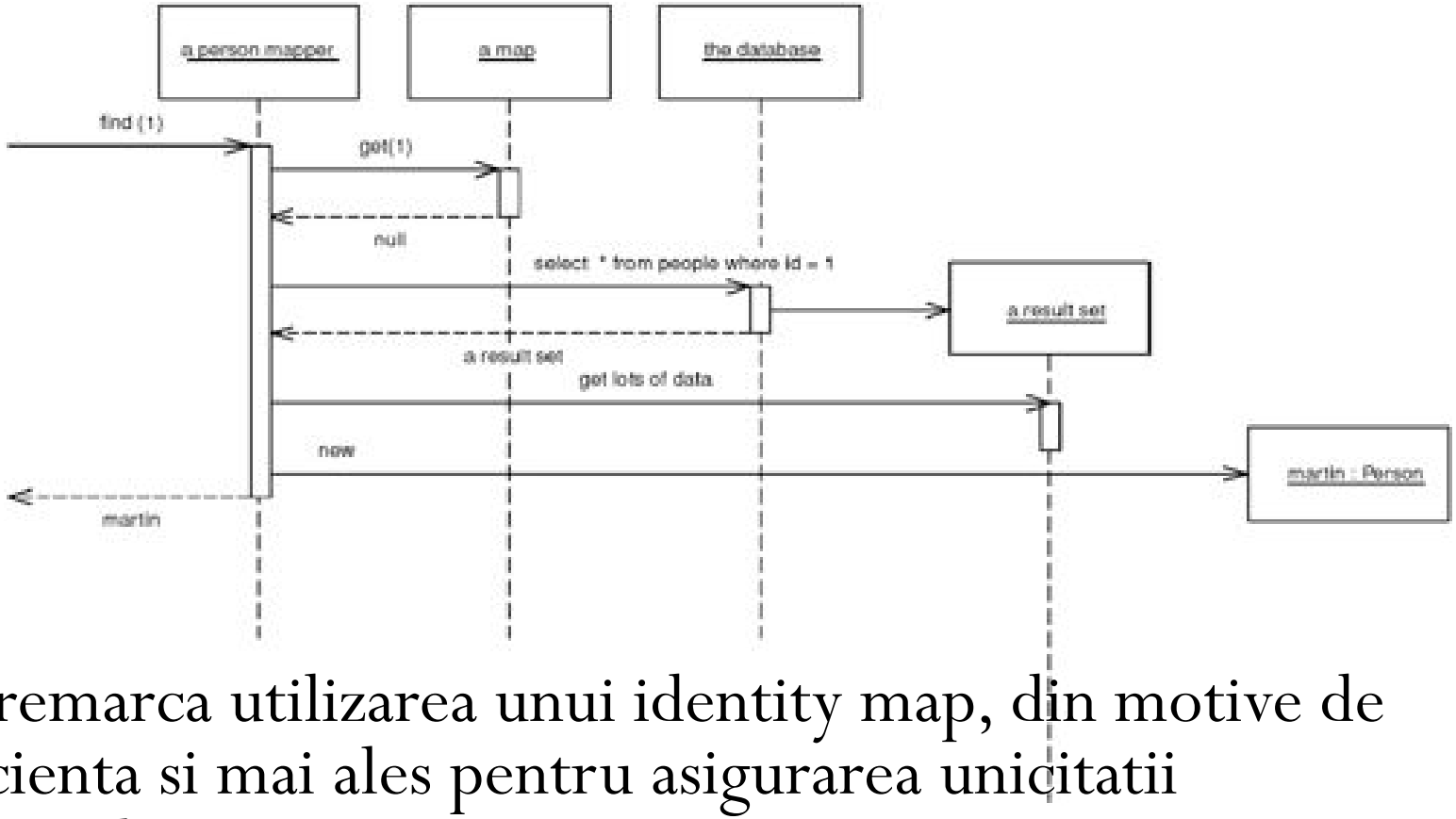

Person
lastName
firstName
numberOfDependents
getExemption
isFlaggedForAudit
getTaxableEarnings
Person Mapper
insert
update
delete

Data Mapper

- Beneficiu: izolarea BD si a BL (domenii susceptibile de schimbare)
  - Ce faci daca BL se complica? Impunerea de reguli de business in BD nu este o idee agreata de catre toata lumea, sau nu toate serverele de BD au acces la aceleasi mecanisme
  - Ce faci daca se decide denormalizarea BD?
- Intreaga complexitate de transformare este gestionata de catre Data Mapper
- Rezultat: izolare intre obiectele din memorie si serverul de baze de date
- In plus, putem folosi mecanismul de abstract factory care poate fi implementat pentru a tine cont de variatiile de implementare datorate unor servere diferite

Data Mapper

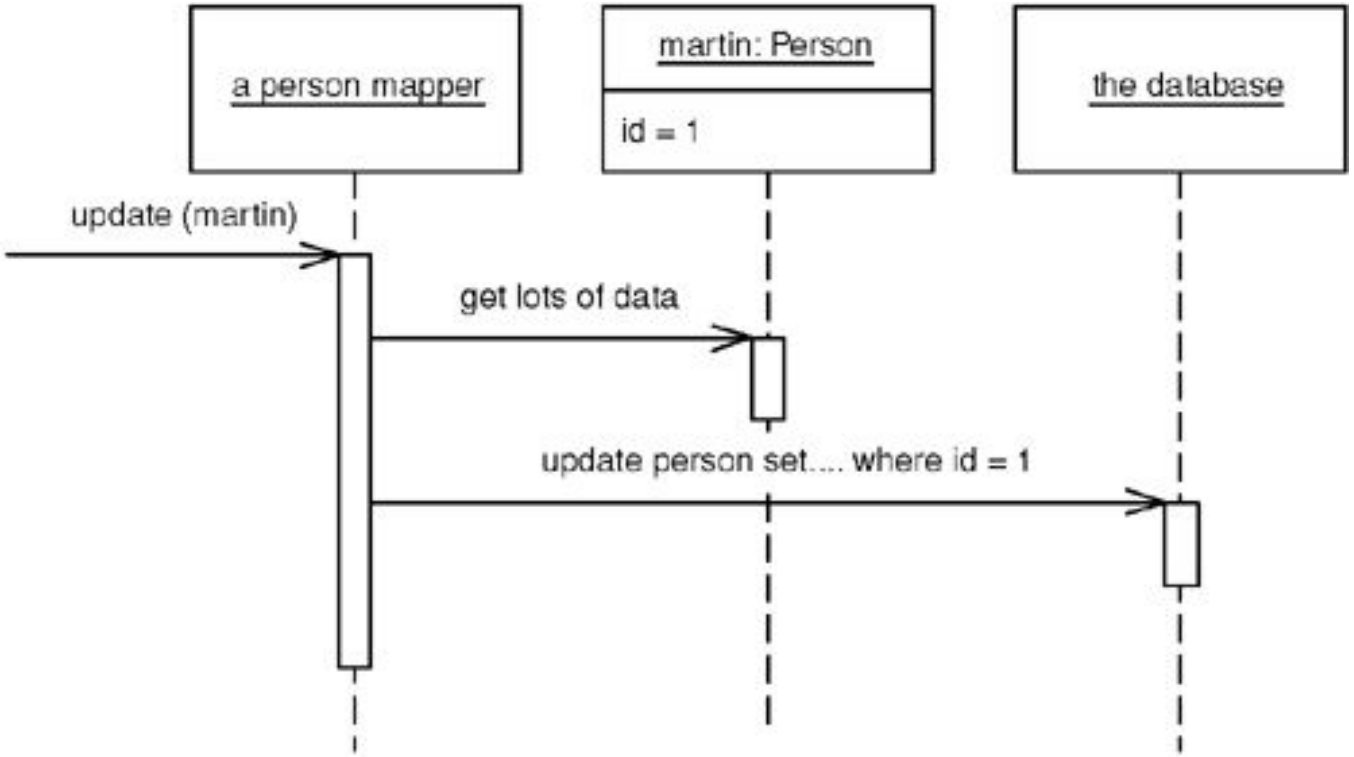
- Astfel functioneaza incarcarea unui obiect:



- Se remarca utilizarea unui identity map, din motive de eficienta si mai ales pentru asigurarea unicitatii obiectului pentru aceeasi inregistrare

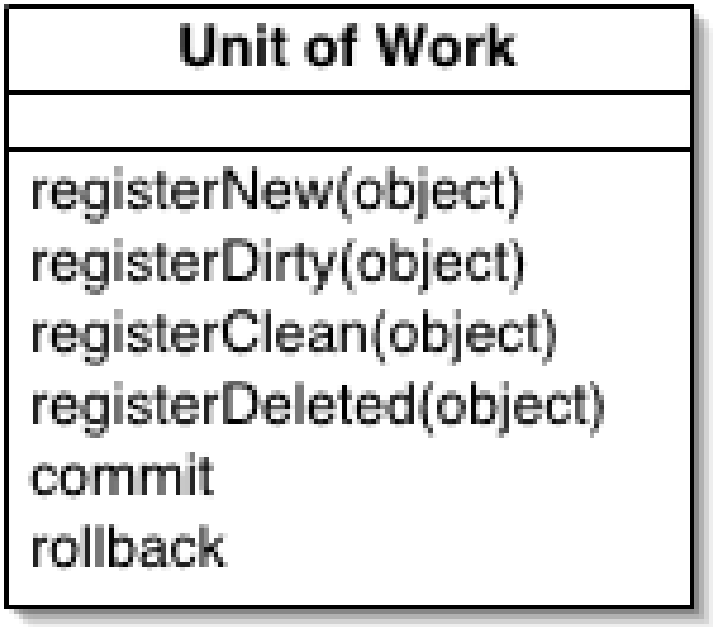
Data Mapper

- Astfel functioneaza modificarea unui obiect:



Data Mapper

- Un intreg nivel de data mapping poate fi substituit pentru a facilita testarea sau pentru a permite schimbarea serverului de baze de date – unul din avantajele lucrului cu straturi
- Trebuie atentie pentru gestionarea obiectelor care au fost modificate, create, distruse (sterse); se poate folosi aici un Unit of Work:



Data Mapper

- E posibil ca o interograre sa nu rezulte intr-o singura interogare SQL, ci in mai multe succesive, sau in jonctiune
- Exemplu: client - comanda - detalii comanda;
- Rezultat: un intreg graf de obiecte care se incarca, iar mapper-ul este cel care decide care/cand se incarca
- Tinta pentru Data Mapper: minimizarea numarului sau al costului apelurilor la serverul de BD

Data Mapper

- Rezultat: se pot incarca mai multe obiecte dintr-o singura interogare
- Evident, nu se pot aduce toate datele interconectate din baza de date in memorie: inutil de multe date + memorie RAM disponibila cu cel putin un ordin de marime mai mica fata de dimensiunea BD
- Se poate utiliza pentru reducerea cantitatii de memorie folosite prin lazy loading

Data Mapper

- Se pot folosi mai multe data mappers: unul pentru fiecare radacina a ierarhiei, sau pentru fiecare clasa din domeniu.
- Sau: daca se foloseste Metadata Mapping, se poate folosi un singur mapper
- Metodele de tip find ar trebui sa foloseasca un IdentityMap pentru a asigura unicitatea obiectelor incarcate deja din BD

Data Mapper

- Metodele de tip “find”
  - Metode accesate de regula chiar de la inceput, din interfata utilizator
  - Apelate si cand business logic are nevoie de obiecte aflate in asociere cu cele deja incarcate, pe baza grafului de obiecte
  - Poti sa ai toate obiectele deja incarcate, sau sa le ceri, sau sa folosesti lazy loading
  - Pentru aplicatii simple se poate crea un set de metode ce pot fi apelate de catre BL

Data Mapper

- Problema (?): un data mapper are nevoie de acces la proprietatile/atributele din obiectele din BL
- Dar pe de alta parte, BL nu reclama intotdeauna un asemenea tip de acces
- Dilema: pui accesorii "set" privati sau nu?
- Variante:
  - Acces la nivel de pachete (C#: internal, Java: fara niciun calificator), dar asta da impresia ca DM este puternic cuplat de tipurile din BL, din cauza ca s-ar afla in acelasi pachet/assembly
  - Ascunderea setterilor, exceptand o clasa: https://www.generacodice.blog/en/articolo/728706/How-can-i-hide-
  - Se poate utiliza reflectarea ("reflection"),
    - In felul acesta se poate trece peste regulile de vizibilitate
    - E mai lenta decat accesul direct, dar luand in considerare si intarzierea data de accesul la serverul de baze de date, s-ar putea ca intarzierea indusa de reflection sa nu conteze foarte mult

Data Mapper

- Alta chestiune: cum initializezi obiectele?
  - 1. constructor care primeste o lista lunga de parametri ce initializeaza campurile necesare din obiect
  - 2. constructor implicit/cu putini parametri si apoi apel de proprietati
  - Care credeti ca e preferabila? (ganditi-va si la obiecte/campuri imuabile)
  - Problema cu prima abordare: referinte ciclice (perechi de obiecte care se refera mutual) => incarcare recursiva

Data Mapper

- Asociere bazata pe metadate:
  - Metadatele descriu modul in care campurile obiectelor sunt transformate in coloane din BD
  - Metadatele se pot stoca fie in clase (sub forma de atribute in .NET, adnotari in Java), fie in fisiere diferite
  - Avantaj: toate variatiile din metadate pot fi gestionate prin reflectare sau regenerare de cod

Data Mapper

- Cand (nu) se foloseste:
  - Da: Cand vrei ca schema bazei de date sa evolueze independent de modelarea obiectuala
  - Da: Cand se foloseste Domain Model pentru modelarea de Business Logic
  - Nu: Daca domain logic este simplu si apropiat de modul de stocare a datelor
    - Se poate porni cu Active Record, daca codul devine complex atunci se poate face refactorizare in Domain Logic + Data Mapper

Data Mapper

- Sfat:
  - Nu se recomanda construirea unui layer de asociere obiectual - relational, ci mai degraba utilizarea unuia pre-existent (sunt multe gata integrate in platforme sau open source)
  - Pentru lumea .NET:
    - Data Access Application block – model primitiv de O/RM
    - Entity Framework
    - NHibernate
- Exemplu de cod: PoEAA, pag 134+