O/R Mapper

O/R Mapper anhand von NHibernate & Entity Framework

Thomas Mentzel

März 2010

Agenda

Object-relational impedance mismatch

Mapping

Session

Abfragen

Object-relational impedance mismatch

Object-relational impedance mismatch

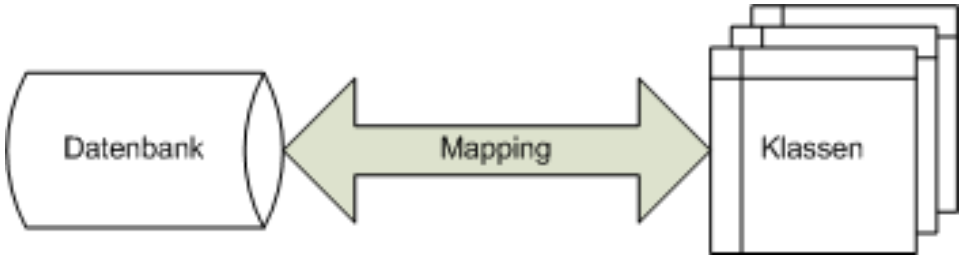
- Struktur
  - Objekt enthält Daten und Verhalten
  - Relationales Modell enthält ausschließlich Tupel (Daten)
- Identität
  - Objekt hat zustandsunabhängige Identität
  - Tupel enthalten Identität (Primärschlüssel)
- Datenkapselung
  - Objekt schützt Daten vor Veränderung (Methoden/Setter)
  - Tupel können beliebig geändert werden (im relationalen Sinne)
- Arbeitsweise
  - Objekte sind Netzwerke interagierender Objekte. Kapselung durch Verhalten
  - Relationale Datenbank arbeitet prozedural

Lösungsansätze

- Objektorientierte Datenbanken
  - Kein “Mapping”
  - Applikationsgebunden
  - NoSQL
- Relationale Funktionen in Programmiersprache
  - “Rückwärtslösung”
  - Keine Objektorientierung
- InterSystems Cachè
- Objekt-Relationale Mapper

Objekt-Relationale Mapper

- Abbildung der Tabellen auf Klassen



- Wer war zuerst? Die Henne oder das Ei?
  - "Datenbank generiert Klassen" (DB zentriert) oder
  - "Klassen generieren Datenbank" (OOD zentriert)

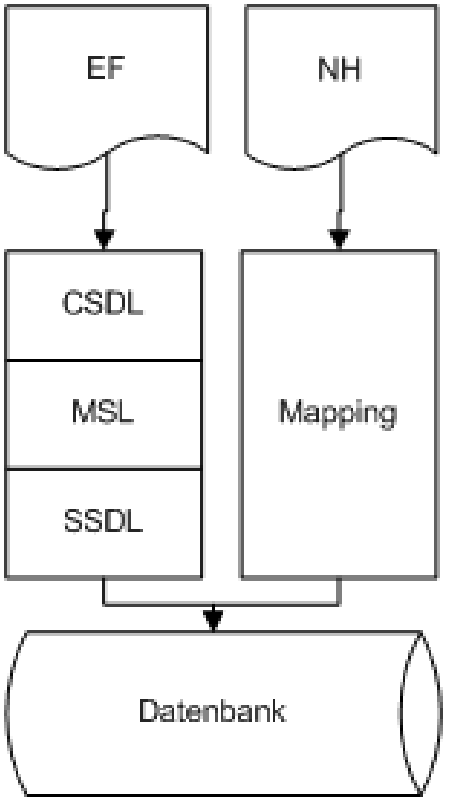
Mapping

Herausforderungen

- Klassen/Eigenschaften
- Primärschlüssel/Zusammengesetzte Schlüssel
- Vererbung
- Löschweitergabe
- Änderungsverfolgung

Konfiguration

- XML (NHibernate, EntityFramework) bzw.
- Proprietäre Konfiguration
- Attribute (NDO)
- Code (FluentNHibernate)



Klassen und Eigenschaften

- Klasse → Tabelle
- Eigenschaft → Spalten
- Objektreferenz → Klasse/Tabelle
- Listen → Klasse/Tabelle

- Komplexe Typen → Tabelle oder Spalten?
- Properties und Spalten nicht namensgleich
- Klassen und Tabellen nicht namensgleich
- Eine Klasse auf zwei Tabellen (Join)

*-Schlüssel

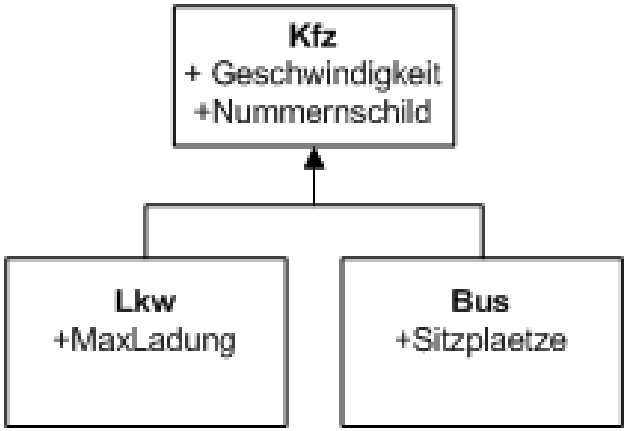
- Objektidentität != Primärschlüssel
- Primärschlüssel Strategien
  - int, auto int, GUID, fachlicher Schlüssel
- Composite Keys
  - GetHashCode() & Equals()
- Fremdschlüsselverweise
  - Eineindeutiges Objekt

NHibernate: Klassen-Mapping

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" namespace="Vap.Psm.Hibernate.Fachklassen" assembly="PsmHibernate">
  - <class name="Benutzer" table="VWBenutzer" lazy="true" mutable="false">
      <!-- Primärschlüssel (ReadOnly) -->
    - <id name="_username" column="Username" type="String" access="field">
        <generator class="assigned" />
      </id>
      <!-- Relationen -->
      <many-to-one name="_orgBereich" class="OrgBereich" column="OrgBereich_ID" access="field" />


      <!-- Primärschlüssel (ReadOnly) -->
    - <id name="_utbId" column="UTB_ID" type="Int32" access="field">
        <generator class="identity" />
      </id>
      <!-- Eigenschaften (ReadOnly ) -->
      <property name="_bez" column="Bez" type="String" access="field" />
      <property name="_isAllowed" column="IsAllowed" type="Boolean" access="field" />


      <!-- Relationen -->
    - <bag name="_ueOrgDaten" inverse="true" lazy="true" access="field">
        <key column="UeOrg_ID" />
        <one-to-many class="UeOrgDaten" />
      </bag>
    </class>
</hibernate-mapping>
```

Vererbung

- Tabelle pro Vererbungshierarchie (Single Table)
  - Vorteile: ID-Contraints
  - Nachteile: Leere Spalten



- Tabelle pro Unterklasse (Joined)
  - Vorteile: Tabellen bilden „Vererbungsstruktur“ ab
  - Nachteile: Abstrakte Basisklasse „instanziierbar“, ID-Constraints beschränkt

- Tabelle pro konkrete Klasse (Table per Class)
  - Vorteile: Keine „instanziierbare“ abstrakte Basisklasse
  - Nachteile: Spalten der Basisklasse in allen Tabellen, keine ID-Constraints

Löschweitergabe

- Cascade Optionen
  - `<Bag … Cascade=„all-delete-orhpans“>`

    none, save-update, delete, delete-orphan, all, all-delete-orphan (NHibernate)
  - `<OnDelete Action="Cascade" />` (EntityFramework; SSDL & CSDL)
- Zirkuläre Referenzen
- Löschen von „Lookup“-Werten
- Wann ist ein Objekt ein „Orphan“ (Waise)

Änderungsverfolgung

- Eigenschaften-Setter überwachen
- Klassen können den Status intern ändern (Fields)

- Unterschiedliche Strategien
  - Eigenschaften mit INotifyPropertyChanged (EntityFramework)
  - Dynamische Proxys (NHibernate)
  - PostCompiler und Code Injection (NDO)

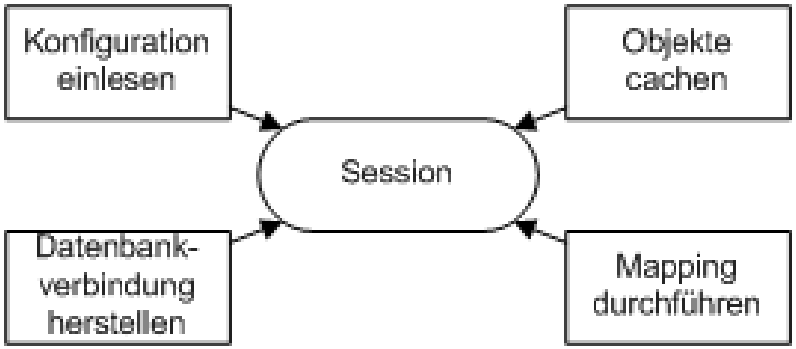
- POCO vs. ORM-Basisklasse
  - NHibernate benutzt „Plain-Old-CLR-Objects“
  - EntityFramework benutzt eine Basisklasse

EntityFramework: Generierte Klassen

```
public partial class Person : global::System.Data.Objects.DataClasses.EntityObject



/// <summary>
/// There are no comments for property Id in the schema.
/// </summary>
[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(EntityKeyProperty=true, IsNullable=fal
[global::System.Runtime.Serialization.DataMemberAttribute()]
[global::System.CodeDom.Compiler.GeneratedCode("System.Data.Entity.Design.EntityClassGenerator", "4.0.0.0"
public int Id
{
    get
    {
        return this._Id;
    }
    set
    {
        this.OnIdChanging(value);
        this.ReportPropertyChanging("Id");
        this._Id = global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value);
        this.ReportPropertyChanged("Id");
        this.OnIdChanged();
    }
}



public partial class Employee : Person
```

Session

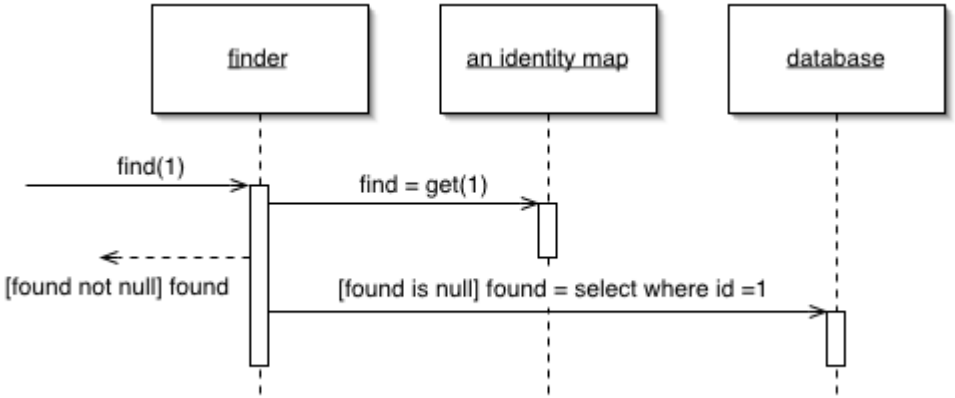
„Session“



- EntityFramework
  - new Context(); // aka. Session
- NHibernate
  - new Configuration().AddAssembly();
  - new SessionFactory(); // ThreadSafe
  - Session = SessionFactory.CreateSession(); // Non-ThreadSafe

Caching der Objekte

- Identity Map [Fowler, P of EAA 195]



- IdentityMap per Session
  - Jede Session liest Daten aus der Datenbank
  - Context1.Person(id=1) != Context2.Person(id=1)

Implementierungen

- Pro Aktion eine neue Session erstellen

```
using (var ctx = ContextHelper.CreateWebContext())
{
    IEnumerable<Benutzer> benutzerList = from benutzer in ctx.BenutzerMenge
                                         where benutzer.Benutzername.ToLower() == benut
                                         select benutzer;

    if (benutzerList.Count() != 1) return null;
    return benutzerList.Single();
}
```

- Singleton Session

```
internal static ImexCdeContext Context
{
    get
    {
        // double-check-locking
        if (_context == null)
            lock (log) // locking auf eine Variable
                if (_context == null)
                {
                    if (log.IsInfoEnabled) log.Info("Neuer Context wird erstellt");
                    try
                    {
                        var cs = new Settings().ImexCdeConnection;
                        _context = new ImexCdeContext(cs);
                    }
                    catch (Exception ex)
                    {
                        if (log.IsErrorEnabled) log.Error("Fehler beim Initialisieren
                    }
                }

        // Context zurück geben
        if (log.IsDebugEnabled) log.Debug("Returning Context");
        return _context;
    }
}
```

*-Loading

- Lazy Loading
- Deferred Loading
- Eager Loading

- Design-Pattern zum Nachladen
  - Lazy Initialization (Initialisieren bei Zugriff –"Singleton")
  - Virtual Proxy (Schnittstellen Dummy)
  - Value Holder (Wrapper)
  - Ghost (Partial Initialized)

- NHibernate: Lazy Loading/Eager Loading

- EntityFramework: Deferred Loading
  - if (!Parent.Children.Loaded) Parent.Children.Load()
  - Context.Parents.Include(„Children")

Abfragen

Abfragesprachen

- Natives SQL
  - IQuery sqlQuery = Session.CreateSQLQuery(someComplexQuery);
  - IList<Person> people = sqlQuery.List<Person>();
- Proprietäre OO Abfragesprachen
  - HQL (NHibernate)
  - var query = „from Cat where Name=‚fritz‘“;
- LINQ
  - seit .NET 3.0
  - SQL ähnlicher Syntax
  - Linq2Objects
  - Linq2SQL (Microsoft)
  - Linq2EF (Microsoft)
  - Linq2Hibernate (OpenSource)
  - Eigene Ling-Provider

LINQ Magie

- Delayed Execution
- Abfrage erzeugt ExpressionTree
- Jedes (!) .ToList() führt eine Datenbankabfrage aus (Linq2EF)

```
public IEnumerable<Cat> FindAllCats()
{
    var query = from c in db.Cats
        select c;

    return query;
}
```

- query.Where().OrderBy().ToList() wird in SQL umgewandelt
- query.ToList().Where().OrderBy() wird auf dem Objektmodell ausgeführt
- Delayed Execution benötigt einen Context

```
public IEnumerable<Cat> FindAllCats()
{
    var query = from c in db.Cats
        select c;

    return query.ToList();
}
```

Häufige Fragen

- Warum führt jede „Abfrage“ auch ein SQL aus?
  Die Session kennt doch die Daten!

Fragen?

Thomas Mentzel

Email: thomas.mentzel@logica.com
MSN: thomas.mentzel@logica.com

Twitter: http://twitter.com/ThomasMentzel
Blog: http://thomas.mentzel.name

logica

Vielen Dank

Thomas Mentzel