Transformation vom objektorientierten in das relationale Modell

basierend auf „The Definitive Guide to db4o“
von Paterson, Edlich, H. Hörning und R. Hörning

Wintersemester 2012/2013

Stefan Werner

Einleitung

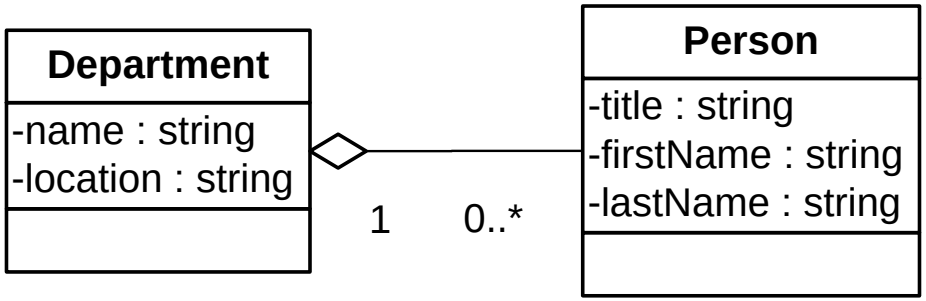
- Eine einfache Klasse kann meist direkt als Tabelle aufgefasst werden
- Allg. werden Klassenvariablen als Attribute aufgefasst
- Objektidentität muss über Primärschlüssel abgebildet werden
- Sonderfälle bei komplexen Klassenbeziehungen notwendig
  - Aggregation
  - Vererbung
  - Many-to-Many-Beziehungen

Objektidentität vs. Gleichheit

- Im oo-Modell besitzt jedes Objekt implizit eine eindeutige Identität
- Zwei Objekte mit den gleichen Eigenschaften sind nicht zwangläufig identisch (das gleiche Objekt)
- Im rel. Modell werden Primärschlüssel eingesetzt, um eindeutige Tabelleneinträge zu generieren
  → Objekte benötigen zusätzliche Schlüsselattribute um sie auf Relationen abbilden zu können

Aggregation

- Teil/Ganzes-Beziehung
- Assoziationen werden gleichermaßen abgebildet, da das rel. Modell nicht zwischen Teil/Ganzes-Beziehungen und einfacher Referenz unterscheidet



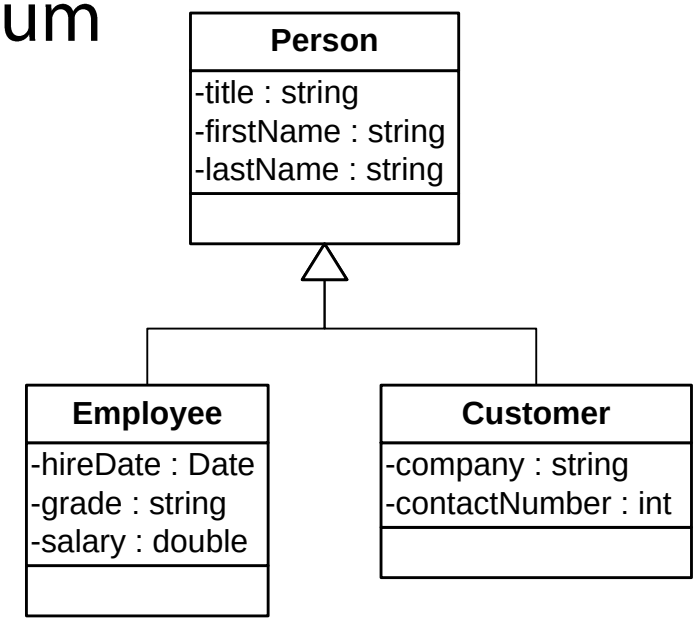
Aggregation - Mapping

- Über Fremdschlüssel in der Tabelle auf der Ganzes-Seite der Beziehung

- **Departments**(DepartmentID, Name, Location)
- **Persons**(PersonID, Title, FirstName, LastName, *dID* → **Departments**)

Vererbung

- Kein äquivalentes Konzept zur oo-Vererbung im relationalen Modell
- Mehrere Möglichkeiten die Vererbung aufzulösen, abhängig vom Vererbungsbaum



Vertical Mapping: One Table per Class

- **Persons**(PersonID,
  Title, FirstName, LastName)
- **Employees**(EmployeeID,
  HireDate, Grade, Salary, *pID* → **Persons**)
- **Customers**(CustomerID,
  Company, ContactNumber, *pID* → **Persons**)

Vertical Mapping: One Table per Class

- Für jede (Unter-)Klasse jeweils eine Tabelle
- Unterklassen stellen per Fremdschlüssel eine Beziehung zur Oberklasse her
- Es müssen jeweils Schlüssel für jede Ausprägung erstellt werden (`PersonID`)
- Um ein `Employee`-Objekt zu erstellen, müssen die Tabellen `Person` und `Employee` über den Schlüssel verknüpft werden (join)
- Kompliziert, aber gut wartbar und robust gegenüber Änderungen

Horizontal Mapping: One Table per Concrete Class

- **Employees**(EmployeeID, Title, FirstName, LastName, HireDate, Grade, Salary)
- **Customers**(CustomerID, Title, FirstName, LastName, Company, ContactNumber)

Horizontal Mapping: One Table per Concrete Class

- Für jede konkrete Klasse eine Tabelle
- Vererbungsbaum wird aufgelöst und jede mögliche Klassenausbildung betrachtet
- einfach
- Nachteile:
  - Redundanz
  - Änderungen in der Oberklasse, müssen an allen Tabellen geändert werden
  - Änderung des Schemas aufwendig

Filtered Mapping: One Table per Tree

- **Persons**(PersonID, Title, FirstName, LastName, HireDate, Grade, Salary, Company, ContactNumber, PersonType)

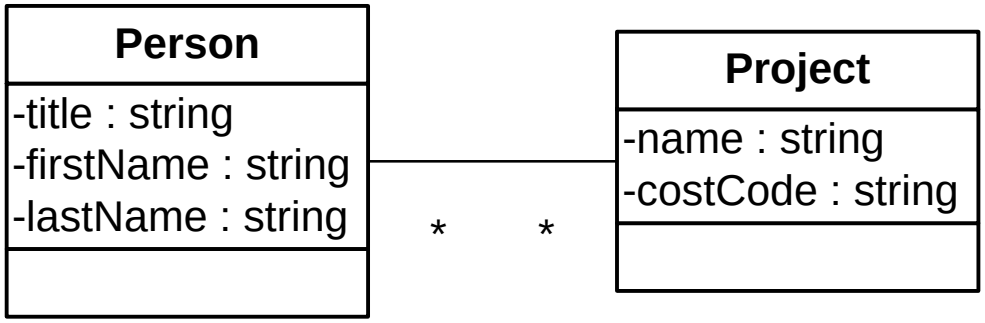
Person ID	Title	FirstName	LastName	HireDate	Grade	Salary	Company	ContactNumber	PersonType
1	Mr	Bob	B.	2003	2	120	null	null	employee
2	Mrs	Alice	A.	null	null	null	MS	5005704	customer

Filtered Mapping: One Table per Tree

- Filterattribut (`PersonType`) erlaubt Unterscheidung zwischen den verschiedenen Ausprägungen
- Keine separaten Tabellen → keine Joins notwendig und einfacherer Zugriff
- Nachteil:
  - Eine große Relation
  - Unterklassen verwenden unnötige Attribute (meist NULL)
  - Typsicherheit muss genauer betrachtet werden

Many-to-Many-Beziehung

- Beispiel
  - Eine Person kann an mehreren Projekten arbeiten
  - An einem Projekt können beliebig viele Personen arbeiten



Many-to-Many-Beziehung

- Analog zum ER-Modell eine separate Beziehungstabelle notwendig
- Dieses Mapping existiert daher schon länger als das oo-Modell

- **Persons**(PersonID, Title, FirstName, LastName)
- **Projects**(ProjectID, Name, CostCode)
- **Assignment**(*perID*→**Persons**, *proID*→**Projects**)