

Kapitel 5: Objektorientierte DB-Modelle & ODMG

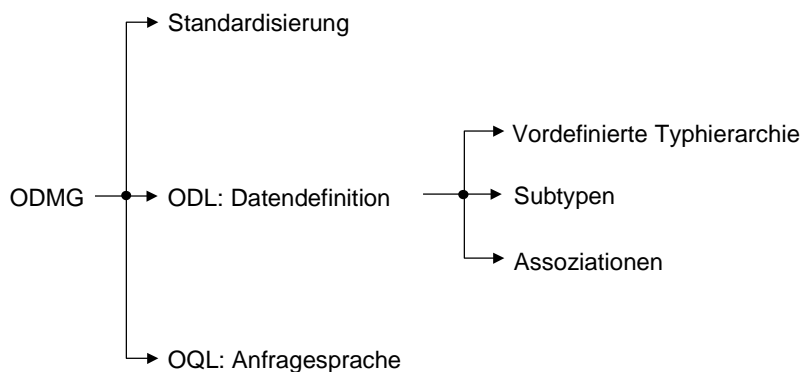
	Relationales Datenmodell (RDM)	Netzwerk- und Hierarchisches Datenmodell (NDM, HDM)	Objekt-orientierte Datenmodelle (OODM)	Objekt- relationale Datenmodelle
Überblick über die Konzepte	3.1	4.1 4.2	5.1	6.1
Darstellung von Assoziationen				
Datendefinition				
Anfragen				
Aktualisierungsoperationen				
Spezifika	3.2 SQL		5.2 ODMG	6.2, 6.3

Datenbanken und Informationssysteme

ODMG 5.2.1

5.2 ODMG im Detail

Überblick:



Datenbanken und Informationssysteme

ODMG 5.2.2

Standardisierung (1)

Motivation für Standardisierung:

- Anwender wollen systemunabhängige Applikationen schreiben.
- Systeme sollen ausgetauscht werden.
- Daten sollen zwischen Institutionen ausgetauscht werden.
- Daten eines Systems sollen von anderen Systemen verwendet werden.

Standardisierungsgremien

- ANSI, ISO, DIN
 - Formale Standards
- OMG, ODMG, ...
 - Herstellerunabhängige industrielle Standards

Standardisierung (2)

ODMG ist ein Standardisierungsvorschlag der ODMG (*Object Database Management Group*) für eine objektorientierte Datenbanksprache.

Ziele:

- Standardisiertes Datenmodell (Objektmodell)
- Kompatibilität des Datenmodells mit OMG / CORBA Begriffen
- Standardisierte Datendefinitionssprache (ODL)
- Standardisierte Anfragesprache (OQL)
- Standardisierte Gastprachenanbindung (C++, Smalltalk, Java)

Teilnehmer:

- Hewlett-Packard, AT&T, Object Design, POET, Servio Corporation, O2 Technology, Versant, Objectivity, ONTOS, Persistence Software, Itasca, Texas Instruments u.a.

Aktueller Stand:

- Version 2.0 (<http://www.odmg.org>)

Inhalt des ODMG Standards (1)

Datenmodell:

- Basis: Das abstrakte *Core Component* Objektmodell der OMG (Object Management Group)
- Festlegung von Konzepten wie Objekt, Typ, Attribut, Beziehung, Operation, Schnittstelle, Implementierung und Supertyp

Datendefinitionssprache:

- Object Definition Language* (ODL) definiert C++-ähnliche Syntax zur Beschreibung von ODMG-Typen
- ODL ist eine aufwärtskompatible Erweiterung der *Interface Definition Language* (IDL) der OMG.
- ODL ist programmiersprachenunabhängig.
- Definition aller Strukturen des ODMG-Modells außer Operationen
- Erweiterbare Syntax

Literatur: R.G.G. Cattell. The Object Database Standard: ODMG 2.0. Morgan Kaufmann, San Francisco, Kalifornien, 1997.

Datenbanken und Informationssysteme

ODMG 5.2.5

Inhalt des ODMG Standards (2)

Anfragesprache:

- Object Query Language* (OQL) definiert SQL-ähnliche Syntax für eine deklarative Sprache zur Formulierung von Anfragen.
- OQL ist nicht algorithmisch vollständig. Methoden, die in einer prozeduralen Sprache implementiert sind, können in OQL benutzt werden.
- Benutzung entweder aus dem Anwendungskode heraus oder interaktiv durch den Datenbanknutzer

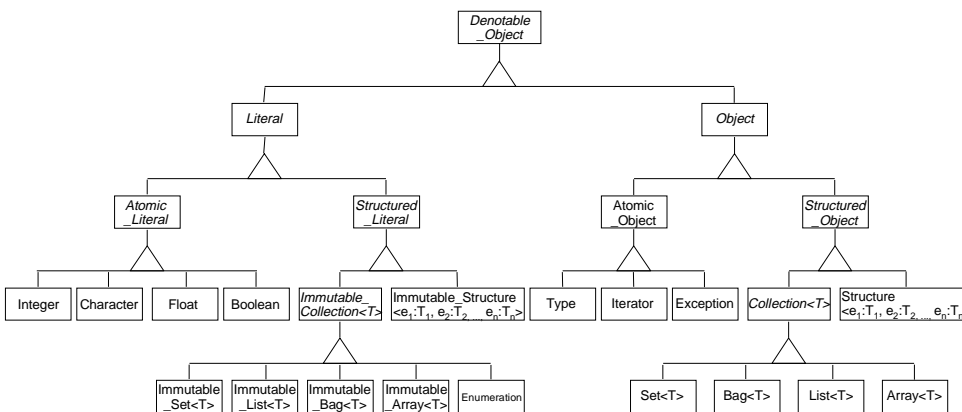
Gastsprachenanbindung:

- C++, Smalltalk und Java *language bindings* definieren die Details der Abbildung des ODMG-Objektmodells und der ODMG-Anfragesprache in die bestehende Syntax der verwendeten Programmiersprache (C++, Smalltalk bzw. Java).
- Realisierung der Sprachenanbindung durch einen Präprozessor oder einen erweiterten Compiler
- Festlegung der konkreten Namens- und Typenkonventionen für die ODMG-Standardtyphierarchie

Datenbanken und Informationssysteme

ODMG 5.2.6

ODMG-Typhierarchie (1)



ODMG-Typhierarchie (2)

Abstrakte Typen:

- Definieren Eigenschaften, die von ihren Subtypen geerbt werden
- Erlauben keine Instanziierung
- Definieren Signaturen, die für ihre Subtypen gelten
- Können Operationen definieren, die auf Instanzen aller ihrer Subtypen anwendbar sind
- Beispiel:** *Literal, Object, Atomic_Literal, ...* kursiv s. Folie 5.2.7

Parametrisierte Typen:

- Besitzen einen (oder mehrere) Typparameter
- Sind keine geschlossenen Typen sondern Typparametern, die erst mit einem konkreten Typ als Parameter versehen einen konkreten Typ darstellen (z.B. `Set<Integer>`, `Immutable_Set<Integer>`).
- Kollektionstypen* enthalten beliebige Subtypen von *Denotable_Object* als Parameter (z.B. `Set<Mitarbeiter>`, `Bag<Abteilung>`).

Objekte und Literale (1)

	Mutabilität	Identifikation	Erzeugung	Gleichheit	Zuweisung
Objekt	veränderbarer Zustand	systemweit eindeutige OID, unabhängig vom Zustand	explizit	gleiche OID	Referenzsemantik
Literal	unveränderlich	keine OID, Identifikation über Werte	atomare Literale existieren implizit, zusammengesetzte Werte werden erzeugt	Wertgleichheit	Kopiersemantik

Objekte und Literale (2)

Beispiel: Objektidentität vs. Wertgleichheit

<code>define p1 as Person(Name: "Peter", Alter: 20)</code>	
<code>define p2 as Person(Name: "Peter", Alter: 20)</code>	<code>p1 ≠ p2</code>
<code>define p3 as p1</code>	<code>p3 = p1</code>
<code>define p4 as struct Person(Name: "Peter", Alter: 20)</code>	
<code>define p5 as struct Person(Name: "Peter", Alter: 20)</code>	<code>p4 = p5</code>

Kollektionstypen

	Duplikate	Ordnung	Größe	Operationen
Set	nein	nein	dynamisch	OQL
Bag	ja	nein	dynamisch	
List	ja	ja	dynamisch	OQL + Listen- operationen
Array	ja	ja	statisch	

Datenbanken und Informationssysteme

ODMG 5.2.11

Iteratoren

- Zugriff auf Kollektionen uniform über Iteratoren möglich
- Anwender kann typisierten, generischen Code zur Iterationsabstraktion schreiben.
- Verwaltung der aktuellen Position in der Kollektion
- Mehrere Iteratoren für eine Kollektion möglich
- Vergleichbar mit *Cursor* in SQL, aber Iteratoren sind Objekte erster Klasse.

```
interface Iterator<T> ()
{
  attribute Boolean stable
  attribute Enumeration order
    (forward, backward)

  T next()
  T first()
  T last()
  Boolean more()
  void reset()
  void delete() };
```

Datenbanken und Informationssysteme

ODMG 5.2.12

Typorthogonalität

```

Set<Structure<
  Titel :String
  Autoren :List<Autor>
  Stichworte :Bag<String>
  Auflagen :Set<Structure<
    Auflage :Integer
    Jahr :Integer>>
>>
    
```

Titel	Autoren	Stichworte	Auflagen	
xxx		<div style="border: 1px solid black; padding: 2px;"> yyy uuu zzz yyy </div>	Auflage	Jahr
			1	88
			2	89
.
.
.

Subtypdefinition

- Das ODMG-Modell unterstützt *strikte* Vererbung: Alle Eigenschaften (Attribute und Relationen) und Operationen werden vererbt und können verfeinert werden.
- Da ein Typ mehrere Supertypen haben kann, sind Subtyphierarchien nicht auf Baumstrukturen beschränkt (→ Mehrfachvererbung).
- Eine Instanz eines Subtyps kann als Instanz jedes seiner Supertypen betrachtet werden und ist in dessen Extension enthalten.

```

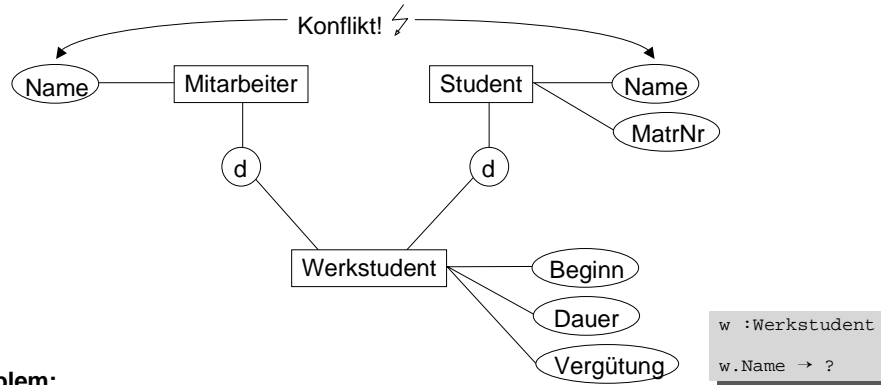
interface Mitarbeiter (extent mitarbeiter key PID)
{attribute Structure<Vorame :String Nachname :String> Name
...}

interface Student(extent studenten)
{attribute Structure<Vorame :String Nachname :String> Name
attribute Integer MatrNr}

interface Werkstudent: Mitarbeiter, Student (extent werkstudenten)
{attribute Datum Beginn
attribute Unsigned Short Dauer
attribute Preis Verguetung}
    
```

Supertypen

Mehrfachvererbung (1)



Problem:

Welche Attributwerte oder Methoden erbt die Unterklasse, wenn die Oberklasse eine Menge von Attributen und Methoden mit gleichem Namen besitzen?

Lösung:

Konfliktlösungsstrategien, die Namenskollisionen auflösen sollen.

Mehrfachvererbung (2)

Mögliche Konfliktlösungsstrategien bei Mehrfachvererbung:

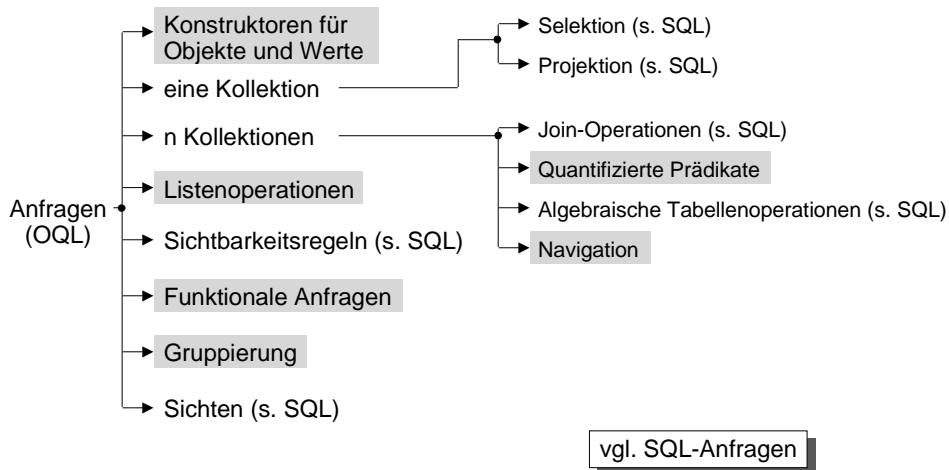
- Einführung einer linearen Ordnung aller Klassen.
Bei einem Konflikt wird immer die Methodenimplementierung der bzgl. der Ordnung früheren Klasse genommen.
- Einführung einer Ordnung bzgl. relevanter Oberklassen.
Hierbei ist die Auswahl der Methode von der Reihenfolge der Spezialisierung abhängig.
- Bei der Konfliktauflösung wird die Umbenennung beim Aufruf der Methode unter Benutzerkontrolle vorgenommen.
- Zur Konfliktvermeidung werden die betroffenen Methoden zur Zeit der Klassendefinition umbenannt.

ODMG-Lösung

```

rename Student.Name as SName
w :Werkstudent
w.SName
w.Name
    
```

Überblick: Anfragen



vgl. SQL-Anfragen

Konstrukturen für Objekte und Werte

Konstrukturen für Objekte:

- Erzeugung durch Angabe des Namen des Konstruktors parametrisiert mit den Initialisierungswerten für Attribute des Objekts.
- Nichtinitialisierte Attribute erhalten den Standardwert *nil*.

```
Abteilung(Kurz = "LTSW",
          Name = "Leitung SW")
```

Konstrukturen für Werte:

- Erzeugung durch Angabe des Schlüsselwortes **struct**.

```
struct(Kurz = "LTSW",
       Name = "Leitung SW")
```

Konstruktor	Parameter	zur Erzeugung
t	$x_1:e_1, x_2:e_2, \dots, x_n:e_n$	eines Objekts vom Typ t
struct	$x_1:e_1, x_2:e_2, \dots, x_n:e_n$	eines strukturierten Wertes
set	e_1, e_2, \dots, e_n	einer Menge
bag	e_1, e_2, \dots, e_n	einer Multimenge
list	e_1, e_2, \dots, e_n	einer Liste
array	e_1, e_2, \dots, e_n	eines Feldes

Quantifizierte Prädikate

- ❑ Quantifizierte Mengenabfragen sind funktionalen Abfragen ähnlich, sie führen jedoch eine Bereichsvariable mit lokalem Sichtbarkeitsbereich ein.
- ❑ Eine Bereichsvariable kann auf zwei Arten quantifiziert werden:
 - **Universell:** Liefert den Wahrheitswert *true*, wenn das Prädikat *p* für alle Elemente, die in der durch den Ausdruck *c* beschriebenen Kollektion enthalten sind, erfüllt ist. `for all x in c:p`
 - **Existentiell:** Liefert *true*, wenn eines der Elemente der Kollektion *c* das Prädikat *p* erfüllt. `exists x in c:p`

Beachte: Verbesserte Orthogonalität im Vergleich zu SQL

Navigation

- ❑ Beziehungsattributen ermöglichen die Navigation zwischen Kollektionen mittels Punktnotation. Explizite Join-Operationen können dadurch vermieden werden.

```
select abt.Chef
  from abteilungen abt
  where not abt.Unterabteilungen.empty?
         and abt.Chef.Gehalt < 10000
```

- ❑ **Erklärung:** Menge der Chefs, deren Abteilung mindestens eine Unterabteilung hat und deren Gehalt kleiner als 10.000 ist.

Listenoperationen

Speziell für geordnete Kollektionen bietet OQL Anfragekonstrukte zur Selektion von einzelnen Elementen und Teilkollektionen an:

- ❑ `first(OQL-Anfrage)`: Selektiert das erste Element einer Kollektion.
- ❑ `last(OQL-Anfrage)`: Liefert das letzte Element einer Kollektion.
- ❑ `OQL-Anfrage [i]`: Das *i*-te Element einer Kollektion kann durch Angabe des Index ausgewählt werden.
- ❑ `OQL-Anfrage [i:j]`: Hiermit wird die Teilkollektion der Elemente an der Position *i* bis *j* selektiert.

Funktionale Anfragen

Aufgrund der Typvollständigkeit von OQL lassen sich Aggregatfunktionen und Selektionsprädikate "natürlicher" als in SQL definieren:

- ❑ `count(OQL-Anfrage)`: Ermittelt die Anzahl der Ergebnis-Elemente der Anfrage.
- ❑ `x in OQL-Anfrage`: Testet, ob der als erster Parameter übergebene Ausdruck in der Kollektion, die als zweiter Parameter übergeben wird, enthalten ist.
- ❑ `element(OQL-Anfrage)`: Extrahiert das (einzige) Element einer Kollektion. Dieser Operator ist wichtig, da `select from where`-Konstrukte Kollektionen als Ergebnis liefern.
- ❑ `flatten(OQL-Anfrage)`: Vermindert die Schachtelungstiefe von ineinander geschachtelten Kollektionen. Er macht aus einer Kollektion von Kollektionen von Ausdrücken vom Typ *T* eine Kollektion von Ausdrücken vom Typ *T*, indem er alle Ausdrücke in einer Kollektion vereinigt.

```
flatten (select a.Angestellte
from Abteilungen a
where ProjektDBFahrplaene in a.Projekte)
```

Gruppierung

Eine select from where-Anfrage kann durch die group by-Klausel die Elemente einer Kollektion in mehrere Partitionen gruppieren.

```
select *
from festangestellte f
group by hoch: f.Gehalt > 8000,
        mittel: f.Gehalt <= 8000 and f.Gehalt >= 3000,
        niedrig: f.Gehalt < 3000
```



```
Set<Structure
<hoch: Boolean,
mittel: Boolean,
niedrig: Boolean,
partition: Bag<Festangestellte> >
```

Objektorientierte Datenbanken: Status Quo

- Es existiert noch kein allgemein anerkannter Standard für objektorientierte Modelle.
- Es existieren weniger formale Grundlagen verglichen mit dem RDM.
- Es existiert noch keine voll entwickelte, standardisierte Anfragesprache.
- Relationale Datenbanksysteme und Objektspeichersysteme haben unterschiedliche Anwendungsbereiche.
- Aktuelle Entwicklung: Objektorientierte Systeme gewinnen zunehmend im Businessbereich an Bedeutung.