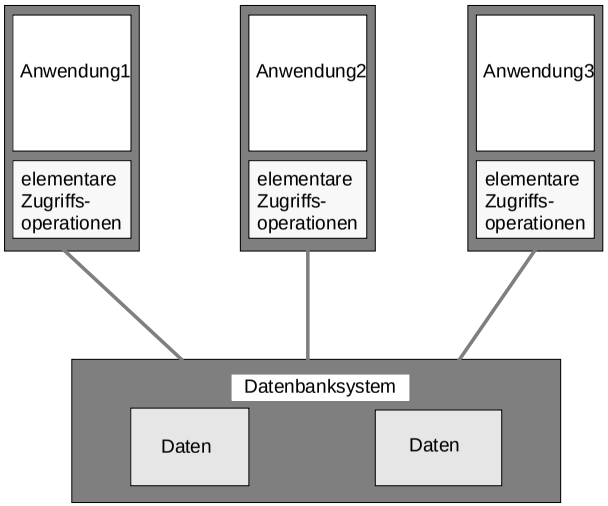
Datenbanksystem



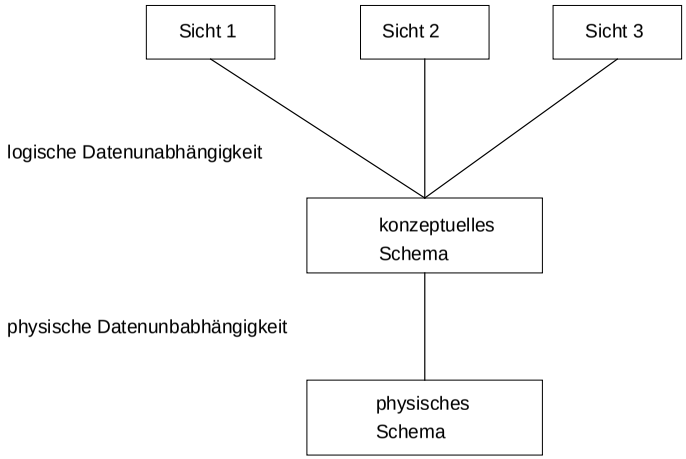
Datenbankkonzepte

1. persistente Datenhaltung
2. Hintergrundspeicherverwaltung
3. Wiederanlauf
4. Mehrbenutzerbetrieb
5. ad hoc-Abfragen
6. Zugriffskontrolle/Datenschutz

Persistente Datenhaltung

- Daten überleben die Programmausführung
- Jede Art von Daten kann persistent sein
- Die Persistenz von Daten ist *implizit*; es ist nicht notwendig Daten explizit vom Hintergrundspeicher zu lesen bzw. auf den Hintergrundspeicher zu schreiben (READ *file*, WRITE *file*)

Ansi/Sparc Architektur



Anforderungen an ein Datenmodell

1. Unterstützung der logischen Datenunabhängigkeit
2. Möglichst große Ausdruckskraft, um die Struktur und Semantik der Daten im Datenmodell festhalten zu können
3. Unterstützung der physischen Datenunabhängigkeit
4. Deklarativität der Datenmanipulationssprache wegen
   - Eignung als ad-hoc-Querysprache
   - High-Level-Modellierung

Probleme relationaler Systeme

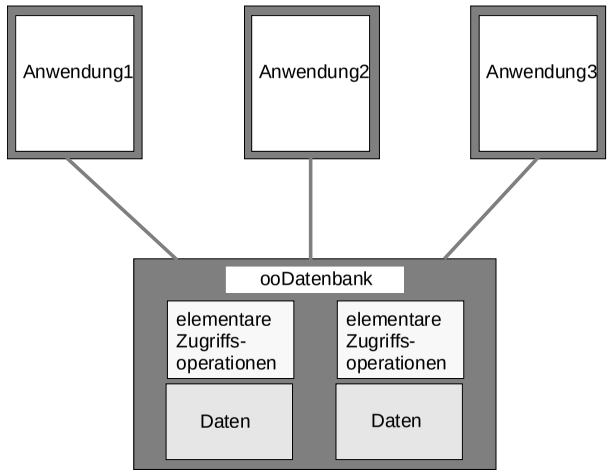
1. fehlende Semantik der Daten
   - keine Unterscheidung zwischen Dingen/Beziehungen
   - keine Darstellung von part-of, is-a, oder Element-Beziehungen
2. keine Erweiterungsmöglichkeit der Datentypen
3. keine Darstellung des Verhaltens
4. Programmiersprachen und Datenbank sind 2 unterschiedliche Welten (verschiedene Syntax, verschiedene Datentypen, explizite Übertragung mit Cursor etc. ...)
5. Referenzintegrität muß explizit festgelegt werden

Objektorientierte Datenbanken

neue Anwendungsbereiche (CAD/CAM, CIM, CASE, etc.) stellen neue Anforderungen an Datenbanksysteme

- Darstellung hochstrukturierter Information
- Modellierung von Verhalten
- Versionen
- Integration von Datenbank- und Progammiersprachenfunktionalität
- effizientere Anwendungsentwicklung

Objektorientierte DB



Regeln

M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier und S. Zdonik (1989) haben "Regeln" für objektorientierte Datenbanken definiert:

- verpflichtende (die ‘‘goldenen Regeln”)
- optionale (wünschenswerte Eigenschaften)
  - z.B.: Version Management, spezielle Transaktionsarten
- offene (Wahlmöglichkeiten, keine Übereinstimmung)

Die Goldenen Regeln

Objektorientierte Regeln	*Datenbank-Regeln*
Objektidentität	persistente Datenhaltung
Sprachvollständigkeit	Hintergrundspeicherverw.
Datenkapselung	Mehrbenutzerbetrieb
Typen/Klassen	Wiederanlauf
Vererbung	ad-hoc Abfragen
Überladen, spätes Binden	Datenschutz
Erweiterbarkeit	

Komplexe Objekte (Part-Of-Beziehungen und Collections)

Arbeiten mit einer OODB

1. Wie herkömmliche Programmierung:
   - Variable enthalten Objekte
   - Aufruf von Operationen an Variable
2. Abfragen wie bei relationalen Datenbanken
   - Ausgangspunkt: Statt Mengen von Tupeln (Relationen) jetzt Mengen von Objekten
   - Sonderfall: Klassen sind ebenfalls (vom System verwaltete) Mengen von Objekten
   - Operationen auf Mengenobjekten: Selektion, Eintragen, Löschen von Objekten

Systeme

Produkt	*Sprache:*
Gemstone	Smalltalk
Ontos, ObjectStore, Objectivity/DB	C++
Versant	Smalltalk, C++
Itasca	Lisp
O_2	CO_2 (C-basiert)
Open ODB (HP)	Object SQL
POET, MOOD,...	

Allgemeine Eigenschaften

- Beinahe durchgehend Entwicklungen kleiner Firmen
- Abzielen auf spezielle Märkte: C*- und MIS-Applikationen, dementsprechend spezielle Features (s.u.)
- Ausrichtung auf verteilten Betrieb (zumindest remote Clients, teilweise auch verteilte Server). Unterschiedlicher Grad der Verteilungstransparenz
- Durchgehend Unterstützung von Collections, teilweise auch Beziehungen (wie Assoziationen in OMT)

ODMG - Object Database Standard

ODMG 1989 gegründet, Konsortium inkl. HP, Sun (Rick Cattell), American Airlines, Canon, Philips, 3Com, *DEC, IBM*, ...

Konzepte:

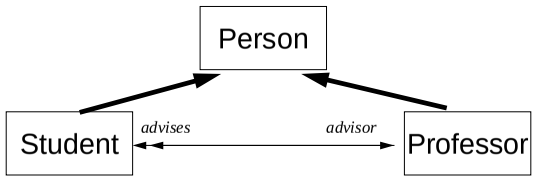
- Herkömmliches Objektmodell mit Vererbung
- *Extent*: Menge der Instanzen eines Typs(automatisch mitgeführt)
- *Key*: Eindeutig identifizierendes Attribut (≠ OID!)
- Typen können mehrere Implementierungen haben
  Klasse = Kombination Typ/Implementierung
- *Relationships* (analog Assoziationen in OMT)
  “traversal path” = “roles” in OMT

ODMG Standard (2)

- Attribute (wie OMT)
- Unveränderliche Objekte (Literale): sowohl strukturiert als auch unstrukturiert (z.B. Datum)
- Collections: List, Bag, Set, Array
  Definiert als generische (parametrisierte) Klassen
- *Entry Points*: Global definierte Variablen, die aus Programmen angesprochen werden können, z.B. Klassennamen, aber auch Einzelobjekte
- Schnittstellendefinitionen für C++, Smalltalk
- Nicht alle Features notwendigerweise tatsächlich in Datenbanken enthalten!

ODMG Object Definition Language (ODL)

- zur Beschreibung von Typen und Einzelobjekten




```
type Professor { supertypes: Person; extent: professors;
        keys: professor_id;
    instance properties: office_extension: Phone_Number;
    advises: { Student } inverse Student::advisor;
    instance operations: announce_course(); .... }
```

- Explizite Unterstützung von Relationships (Assoziationen)

ODMG Object Query Language (OQL)

- ähnlich SQL/Relationenkalkül
  Syntax abstrakt (nicht fix vorgegeben)
  konkrete Syntax üblicherweise wie SQL

  exists x **in** Miller.takes: x.taught_by.name = “Turing”

  select pair(student: x.name, professor: z.name)
  from x **in** Students,
  y **in** x.takes,
  z **in** y.taught_by
  where z.rank = “full professor”

Persistenzansätze

- Automatisch: Objekte werden persistent, wenn sie erzeugt werden
  - Löschen explizit (OpenODB, ObjectStore)
- Über “Wurzeln” (roots):
  (a) Objekt bekommt einen globalen Namen
  (b) kann über ein anderes persistentes Objekt erreicht werden
  - Löschen durch Garbage-Collection-Prinzip (Gemstone, O2)

Gemstone

Smalltalk-basiert, mit gewissen syntaktischen Erweiterungen

Klassendefinition:

Object **subclass:** ‘Document’
instVarNames: #(‘title’, ‘authors’, ‘status’, ‘content’)
classVars: #()
poolDictionaries: #()
inDictionary: UserGlobals
constraints: #[#[#[#title, String],
#[#authors, Employees],
#[#status, String], #[#contents, Text]]
instancesInvariant: false
isModifiable: false

Gemstone (2)

constraints-Klausel: Typdefinition für Instanzvariablen (notwendig, wenn über diese Variablen ein Index definiert werden soll)

instancesInvariant: Werte von Instanzvariablen können verändert werden

isModifiable: Definition der Klasse noch änderbar

inDictionary: Gibt den “Namespace” des Benutzers an (ähnlich wie die Angabe einer “Datenbank” in einem RDBS eine bestimmte Menge von Relationen definiert).

globale Objekte sind globale Variablen in diesem Dictionary (so wie globale Smalltalk-Variablen im Dictionary ‘Smalltalk’ eingetragen werden)

Gemstone (3)

Query:

smithProj :=
Projects **select:** { aProject |
aProject.manager.name = 'Smith' }.

- Wie Smalltalk-Collections
- '{'-'}'-markierte Blocks zeigen Verwendung von Indexstrukturen an
- '.'-Notation zeigt an, daß messages direkt die Instanzvariablen-Werte zurückliefern (keine berechneten Werte)

ObjectStore

- "Datenmodell": C++-Klassendefinitionen
- Einstiegspunkte: sogenannte **Root-Objekte** (globale, über die Datenbank erzeugte Variable). Bsp.:

```
persistent <db> type variable;
persistent <LVA> student stud1;
```

- Root-Objekt mit demselben Namen wie die Programmvariable wird automatisch im Namensraum der DB gesucht
- Wenn nicht gefunden: neu erzeugt
- Erzeugen: Überladen des new-Operators:
  Bsp.:

```
lva1 = new(LVA) lva(nr,titel,typ,std);
```

- Löschen explizit: `delete lva1`

ObjectStore (2)

- Collection-Typen: eigene Klassenbibliothek
- einheitliche Schnittstelle (`insert, contains, empty`, ...), unterschiedliche Implementierung
- Iteration: `foreach` (forward, reverse, ...)
- Implementierung wird möglicherweise automatisch während der Laufzeit geändert (Vorgabe von Performance-Kriterien möglich: Prozentsatz der untersch. Zugriffsarten, z.B.
- Query expressions:

```
os_Set<order*> orders1 =
   orders[: quantity>=100 && status=not_shipped &&
   customer->is_very_important_company() :]
```


ObjectStore (3)

- Pointer und Referenzen
  - Pointer sind "echte" Pointer im Hauptspeicher ("transient")
  - bleiben daher zwischen Transaktionen nicht erhalten!
  - explizites Umwandeln von/zu Referenzen nötig
- Explizites Unterstützen von Relationships (automatische Integritätserhaltung)

```
class person{company* employer inverse_member employees}
```

Stand der Dinge

- Erste Standardisierung
- Problematik der Sprachintegration
- Konzepte der unternehmensweiten Datenmodellierung werden in vielen Fällen ignoriert (Beispiel: ObjectStore)
  - Objektmodell: C++
  - Zugriffschutz: Hostsystem, Filesystem
  - Keine Überprüfung des korrekten Code-Linkings (auf DB-Ebene keine Berücksichtigung der Datenkapselung)

Einsatzchancen: kaum in traditionellen, meistens in C* und MIS Anwendungen