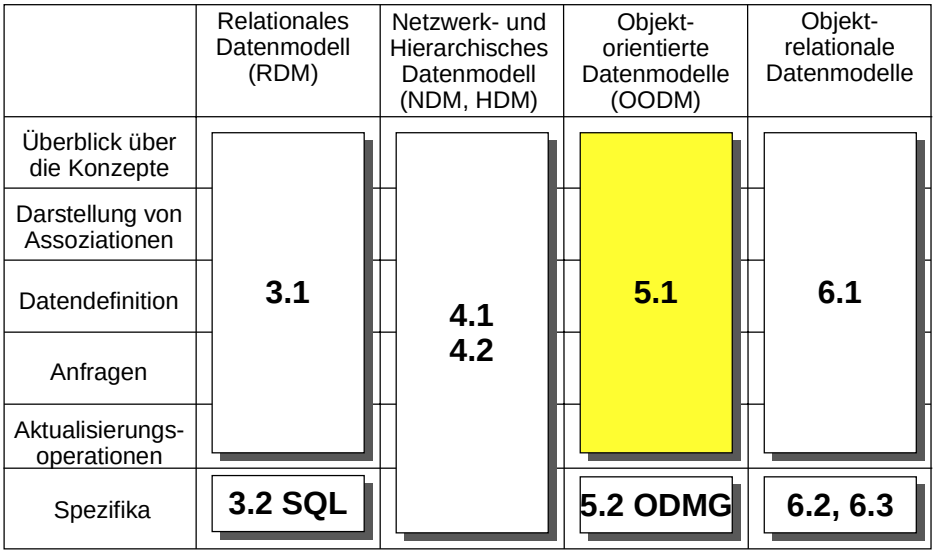
Kapitel 5: Objektorientierte DB-Modelle & ODMG



5.1: Überblick über objektorientierte DB-Modelle

Nachfolgend wird das ODMG Datenbankmodell als das wichtigste OO-Datenmodell behandelt.

Das ODMG-Datenmodell basiert auf dem *Core Component* Objektmodell der *Object Management Group* (OMG) und stellt ein spezielles Objektorientiertes Datenmodell dar.

Modellierungsmächtigkeit gegenüber dem RDM stark erweitert durch:

- ❏ Orthogonale Persistenz
- ❏ Typkonstruktoren und komplexe Objekte
- ❏ Objektidentität
- ❏ Typen und Klassen
- ❏ Methoden und Kapselung
- ❏ Vererbung
- ❏ Methodenredefinition und späte Bindung

Anforderungen der OO-Manifesti

OODM: Typkonstruktoren und komplexe Objekte

Komplexe Objekte werden aus einfacheren Objekten durch die Verwendung von Konstruktoren aufgebaut.

- ❏ Grundbausteine sind die Basistypen
- ❏ Konstruktoren für Aggregattypen
- ❏ Konstruktoren für Kollektionen

`Integer, Boolean`

`Tuple`

Ungeordnete Kollektion:
`Set, Bag`

Geordnete Kollektion:
`List, Array`

- ❏ Orthogonale Schachtelung der Konstruktoren
- ❏ Ein ODMG Objekt ist Instanz genau eines Typs.

OODM: Objektidentität

Jedes Objekt besitzt eine von seinem Wert unabhängige Identität. Es erhält bei seiner Erzeugung eine Objektidentifikator:

- ❏ Der Identifikator ist systemweit eindeutig. Ein Objekt ist damit von allen anderen Objekten unterscheidbar.
- ❏ Der Identifikator ist unabhängig vom Zustand und den Zustandsänderungen des Objektes.
- ❏ Der Objektidentifikator wird vom System verwaltet und kann vom Benutzer nicht geändert werden.

Vorteil: Objekte können durch mehrere andere Objekte referenziert werden (*object sharing*), wodurch sich beliebige (auch zyklische) Objektgraphen aufbauen lassen.

OODM: Typen und Klassen

- ❏ Typen dienen der statischen Klassifikation von Werten und Operationen. Zweck: Vermeidung von Laufzeitfehlern, Wahl von Speicherungsstrukturen und Optimierungen.
- ❏ Typen sind syntaktische Bedingungen auf Ausdrücken, die die Operator-/Operand-Kompatibilität (zum Übersetzungszeitpunkt) sicherstellen sollen.
- ❏ Ein Typ definiert das Verhalten und die möglichen Zustände seiner Instanz.
  - Methodensignaturen
  - Attributtypen
- ❏ Bei der Typdefinition können *Attribute* für das Objekt und *Beziehungen* zu anderen Objekten festgelegt werden.
- ❏ Zu einem Typ kann eine *Extension* verwaltet werden, in die Objekte dieses Typs bei ihrer Erzeugung automatisch eingefügt werden.
- ❏ Zu einem Typ kann es mehrere Implementationen geben. Eine Typdefinition zusammen mit einer seiner Implementierungen wird als *Klasse* bezeichnet.
  - Methodenrümpfe
  - Attributinitialisierungen

OODM: Kapselung

- ❏ Kapselung ist ein aus dem Bereich der Programmiersprachen bekanntes Konzept (→ *Modulkonzept, Abstrakter Datentyp*).
- ❏ Ein Objekt besitzt einen Zustand (die aktuellen Attributwerte) und eine Menge von Methoden, über die auf den Zustand des Objektes zugegriffen werden kann.
- ❏ Die Kapselung heißt *strikt*, wenn ausschließlich über die Objektmethoden auf den Zustand zugegriffen werden kann. Das ODMG-Modell erzwingt strikte Kapselung.

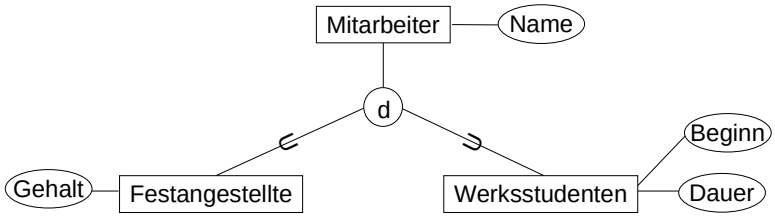
Vorteile der Kapselung:

- ❏ Die Implementationen der Methoden können geändert werden, ohne Anwendungsprogramme zu invalidieren (*logische Datenunabhängigkeit*).
- ❏ Integritätsbedingungen können in den Methoden überprüft werden.
- ❏ Zugriffsberechtigungen können überprüft werden.
- ❏ Objekte stellen Einheiten dar, die potentiell über die Grenzen von Anwendungen, Sitzungen, Systemen und Plattformen hinaus, also *global*, aufbewahrt, verteilt und verwendet werden können. Andererseits können sie *zentral* definiert, implementiert und gewartet werden.

OODM: Vererbung

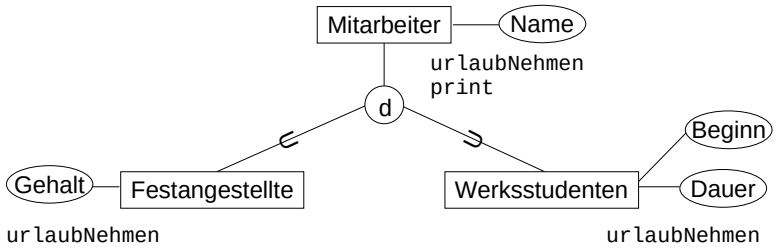
- ❏ Typen sind in einer *Subtyp-/Supertyphierarchie* organisiert, wobei ein Typ mehrere Supertypen besitzen darf.
- ❏ Es gibt eine vordefinierte, durch Subtypen erweiterbare ODMG-Typhierarchie.

Die Subklassen erben die Attribute und Methoden der Superklassen. In der Subklasse können Methoden und Attribute redefiniert werden.



- ❏ Vererbung ist ein mächtiges Modellierungskonzept zur Beschreibung von Gegebenheiten der realen Welt (is-a Beziehungen = Spezialisierungsbeziehung).
- ❏ Vererbung unterstützt die Wiederverwendung von Definitionen und Implementationen unter Berücksichtigung von Klassenunterschieden.

OODM: Methodenredefinition



Begriffe:

- ❏ **Overloading**: Für einen Methodennamen (`urlaubNehmen`) existieren unterschiedliche Methodenimplementationen.
- ❏ **Overriding**: Die Methodenimplementation der Subklasse hat Vorrang vor der ererbten gleichnamigen Methodenimplementation der (transitiven) Superklasse(n).
  Beispiel:
  `Festangestellte.urlaubNehmen` redefiniert `Mitarbeiter.urlaubNehmen`

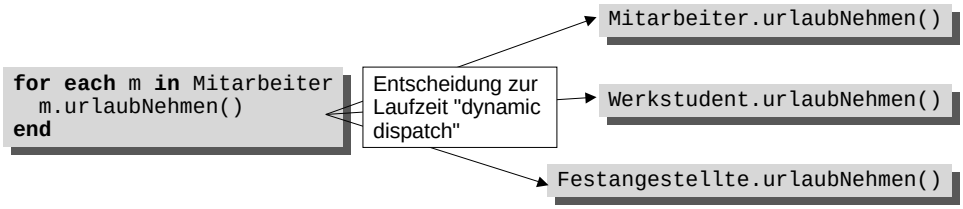
OODM: Späte Bindung *(late binding)*

- In Abhängigkeit von der Klassenzugehörigkeit eines Objektes wird *zur Laufzeit* eine Methodenimplementation für eine überladene Methode ausgewählt.

  Vergleiche: frühe statische Bindung für überladene Bezeichner in C.

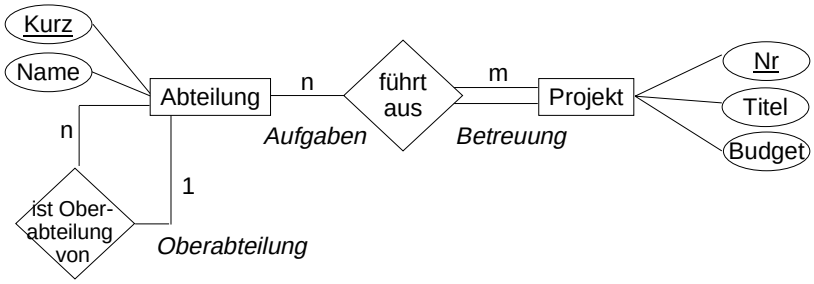
- Die auszuführende Methode kann durch Konsultieren des Klassenobjektes (Metaobjekts) und entsprechendes Traversieren der Klassenhierarchie gefunden werden.

Der Mechanismus der dynamischen Bindung erlaubt sowohl dem Klienten als auch dem Implementierer einer Methode von der Auswahl der korrekten Implementation zu abstrahieren.



OODM: Durchgängiges Beispiel

Zur Erinnerung das durchgängige Beispiel:



OODM: Darstellung von Assoziationen

Beispiel: ODL-Schemadefinition der Firmendatenbank (Ausschnitt)

```
database schema FirmenDB

interface Abteilung
( extent Abteilungen
  keys Kurz)
{ attribute string Kurz;
  attribute string Name;
  relationship set<Projekt> Aufgaben
   inverse Projekt::Betreuung;
  relationship Abteilung Oberabt;
  boolean neuesProjekt(in Projekt p) raises (...)
};

interface Projekt
( extent Projekte
  keys Nr)
{ attribute unsigned short Nr;
  attribute string Titel;
  attribute float Budget;
  relationship set<Abteilung> Betreuung
   inverse Abteilung::Aufgaben;
};
```

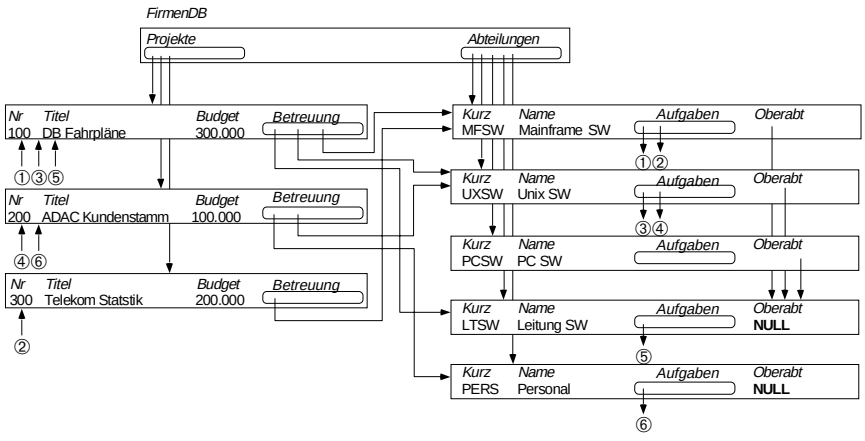
Typ & Klasse

Assoziation über OID-Mengen

Typ & Klasse

OODM: ODMG-Objektmodell (1)

Zustand der Projektdatenbank:

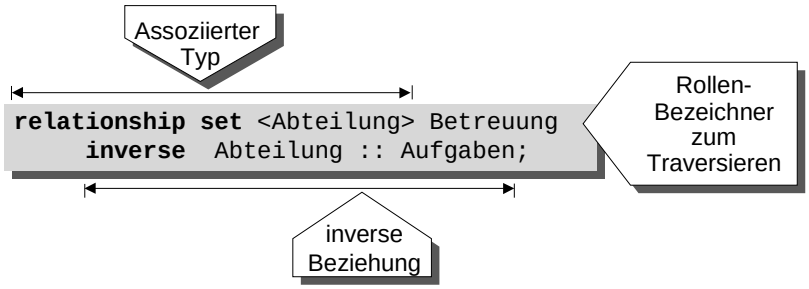


OODM: ODMG-Objektmodell (2)

- ❏ Definition der ODMG-Typen *Abteilung* und *Projekt*
- ❏ Die Datenbank besteht aus einem Aggregat mit den beiden benannten Mengen *Projekte* und *Abteilungen* als Komponenten.
- ❏ Diese Mengen sind die Extensionen der beiden Objekttypen *Projekt* und *Abteilung.*
- ❏ Die Mengen enthalten Referenzen auf die Objekte (*Objektidentifikatoren*), die als Instanzen des entsprechenden Objekttyps existieren.
- ❏ Die einzelnen Objekte sind Aggregate, die sich aus den Eigenschaften des Objekts zusammensetzen:
  - Die *Attribute* der Objekte können wieder Kollektionen oder Aggregate sein, die sich mit beliebiger Schachtelungstiefe aus den Basisdatentypen zusammensetzen.
  - Im Beispiel treten allerdings nur atomare (flache) Attributwerte auf.

OODM: Assoziationen im ODMG-Objektmodell

- ❏ Darstellung von Beziehungen zu anderen Objekten nicht wertbasiert durch die Verwendung von Fremdschlüsseln, sondern durch Objektidentifikatoren (OIDs)
- ❏ Assoziationen werden ähnlich wie Attribute definiert:



- ❏ Diese speziellen Attribute besitzen Referenzen auf andere Objekte als Werte (→ Kollektionen von Referenzen werden möglich).

OODM: Datendefinition im ODMG-Objektmodell (1)

Die Datendefinitionssprache ODL im ODMG-Standard:

- ODL: Object Description Language definiert eine Datenbank als Sammlung von Typdefinitionen (`interface`)
- **Objekttyp:**
  - Eigenschaften für die Objekte dieses Typs
    - Attribute (Name und Typ)
    - Assoziationen (Name, Typ und Inverse-Assoziation)
    - Methoden (Name und Signatur)
  - Eigenschaften für den Typ selbst
    - Extension
    - Schlüssel
    - Supertypen und Vererbung

OODM: Datendefinition im ODMG-Objektmodell (2)

Attribut:

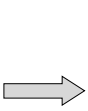
- Extern sichtbarer Zustand des Objekts
- Nur Literale (auch zusammengesetzte) als Attributwerte
- Attribut spezifiziert zwei Methoden:
  `getValue()` → Literal
  `setValue(new_Value :Literal)`
- Attribute können mit benutzerdefinierten Funktionen überschrieben werden:
  Beispiel: Alter-Berechnung
  `getValue()` → Tagesdatum - Geburtsdatum
  `setValue(...)` → Differenz berechnen und Geburtsdatum aktualisieren
- Invariante:
  `getValue(setValue(x)) = x`

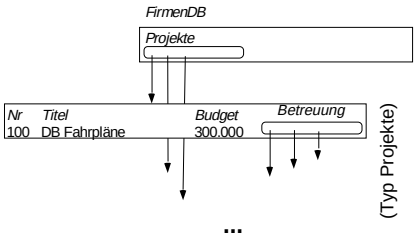
OODM: Extension

- ❏ Automatische Verwaltung der aktuellen Objektmenge zu einem Typ (optional)
- ❏ Enthält alle erzeugten und nicht wieder gelöschten Objekte zu diesem Typ
- ❏ Wichtig im DB-Bereich für die Verwaltung von Massendaten
- ❏ Typ als Objektfabrik und Sammelbehälter

```
database FirmenDB

interface Projekte
(extent Projekte)
```



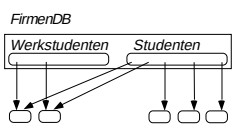


OODM: Supertypen und Vererbung

- ❏ Typen können in Subtyphierarchien angeordnet werden
- ❏ Angabe mehrerer Supertypen möglich (Mehrfachvererbung)
- ❏ Vererbung von Attributen, Assoziationen und Methoden
- ❏ Methodenredefinition und Verfeinerung weiterer Eigenschaften ist möglich
- ❏ Extension eines Subtyps ist Teilmenge der Extension seines Supertyps (z.B. *Werkstudenten* $\subseteq$ *Mitarbeiter*)

```
database FirmenDB

interface Werkstudent: Student, Mitarbeiter
(extent Werkstudenten)
{ attribute Datum  Beginn;
  attribute unsigned short Dauer;
  attribute DM Verguetung; };
```



OODM: Datendefinition

Modellinhärente Bedingungen:

- ❏ Eindeutigkeit von Schlüsseln
- ❏ Referentielle Integrität beim Löschen von Objekten
- ❏ Inklusionsbedingung bei Subtypbeziehungen

Benutzerdefinierte Bedingungen:

- ❏ Überprüfung in den Methoden
- ❏ Kapselung verhindert unkontrollierten Zugriff
- ❏ **Aber:** Ausprogrammierung der Tests notwendig (in aktueller Version des ODMG-Standards)

OODM: Anfragen

Die Anfragesprache OQL im ODMG-Objektmodell:

- ❏ OQL: Object Query Language
- ❏ Deskriptiv, mengenorientiert
- ❏ Obermenge der relationalen Anfragesprache SQL
- ❏ **Wichtigste Erweiterungen gegenüber SQL:**
  - Berücksichtigung von *Werten* und *Objekten*
  - Offenheit gegenüber verschiedenen Arten von Kollektionen (Anfragen gegen Kollektionen und *Werte* und *Objekte* als Anfrageergebnisse möglich)
  - *Höhere Sprachorthogonalität:* Die OQL-Konstrukte sind frei kombinierbar.
  - Keine Beschränkung auf flache Strukturen (geschachtelte und mengenwertige Attribute)
  - Neben einem Satz vordefinierter Funktionen können beliebig definierte Methoden in Anfragen verwendet werden.

OODM: Anfragen in OQL

Beispiel:

vgl. 5.1.11 und 5.1.12

Ermittlung der Titel aller Projekte, an denen die Abteilung *Mainframe SW* arbeitet.

```
select p.Titel
from Projekte p
where exists a in p.Betreuung : a.Name = "Mainframe SW";
```

Hier:

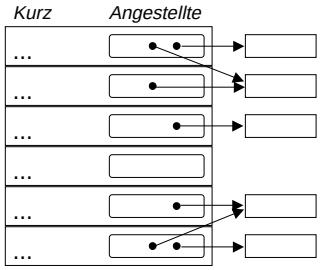
- keine *Join-Operation* notwendig (wg. der direkten Darstellung der Beziehung zwischen den Projekten und den Abteilungen, von denen es betreut wird)
- Zugriff auf die Abteilungen, die das Projekt p betreuen über `p.Betreuung`

OODM: Anfragen in OQL: Orthogonalität

Beispiel: Geschachtelte Bereichsrelation und OIDs im Ergebnis

Selektiere alle Angestellten, die am DB-Fahrplan-Projekt arbeiten,
und gib zu deren Abteilungen die Kurzbezeichnung aus.

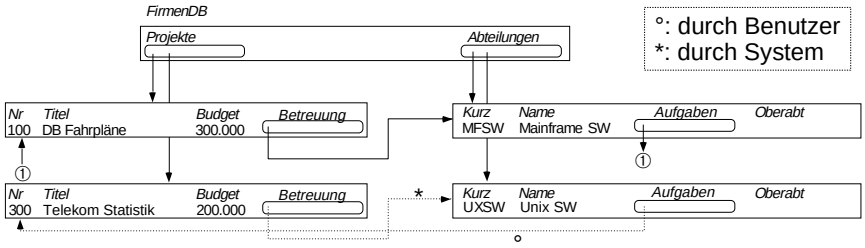
```
select (abt.Kurz, (select ang
        from   abt.Angestellte  ang
        where  dbFahrplaeneProjekt in
               ang.Projekte))
from   Abteilungen  abt
where  ...
```



OODM: Aktualisierungsoperationen (1)

Operationen im ODMG-Objektmodell:

- ❏ Die Operationen (Einfügen, Löschen, Modifizieren von Daten der Datenbank) treten sowohl für Objekte als auch für Assoziationen zwischen Objekten auf.
- ❏ Operationen auf Beziehungsstrukturen sind stets einem konkreten Objekt zugeordnet und manipulieren die Assoziationen, an denen dieses Objekt beteiligt ist.
- ❏ Bei *bidirektionalen Beziehungen* bewirken die Operationen unabhängig von ihrer Zuordnung zu einem konkreten Objekt eine Aktualisierung beider Richtungen der Assoziation:



OODM: Aktualisierungsoperationen (2)

Einfügen:

- ❏ Die Extensionen der Typen repräsentieren die Massendatenstrukturen, in denen die *Objekte* des Typs verwaltet werden.
- ❏ Bei der Objekterzeugung geschieht das Einfügen von Objekten in die zugehörige Extension automatisch. In diesem Fall stellt die Einfügeoperation eine Teilfunktionalität der Objekterzeugung dar.
- ❏ Aufgrund des *orthogonalen Persistenzkonzepts* können beliebige Objekte persistent gemacht werden und in der Datenbank eingefügt werden (vgl. RDM, wo die Datenbank mit den Relationen auf die persistente Speicherung der Massendatenstrukturen beschränkt ist).
- ❏ Das Einfügen neuer *Beziehungen* geschieht durch Operationen, die ein Objekt mit einem einzelnen Objekt oder einer Menge von Objekten des assoziierten Typs in Beziehung setzen.

OODM: Aktualisierungsoperationen (3)

Löschen:

- ❏ Jedes *Objekt* besitzt eine Methode *delete*, die vom Supertyp *Object* aller Objekttypen ererbt wird. Diese Methode löscht das Objekt, für das sie aufgerufen wird und macht noch existierende Referenzen auf das Objekt ungültig.
- ❏ Für das Löschen von *Assoziationen* zwischen Objekten gilt:
  - Selektives Löschen: Die Beziehungen zu einzelnen Objekten werden gelöst.
  - Vollständiges Löschen: Die Beziehungen zu allen assoziierten Objekten werden gelöst.

OODM: Aktualisierungsoperationen (4)

Modifizieren:

- ❏ Es existiert keine allgemeine Funktion *update* für die Änderung beliebiger Attribute.
- ❏ Für jedes *Attribut eines Objekttyps* existiert eine vordefinierte Methode *set-value,* die redefiniert werden kann.
- ❏ Außerdem kann der Objektzustand durch benutzerdefinierte Methoden verändert werden.
- ❏ Eine *Beziehung* kann, ausgehend von einem konkreten Objekt, modifiziert werden.
- ❏ Dabei kann dem Objekt abhängig von der Schemadefinition nicht nur ein einzelnes Objekt, sondern eine Menge von Objekten zugeordnet werden.

OODM: Aktualisierungsoperationen (5)

Beispiel: Aktualisierungsoperationen sind in Host-Sprache (hier: C++) eingebettet:

```
Ref<Abteilung> aOID = Abteilungen.select_element(
  "select a in Abteilungen where a.Name = \"Unix SW\"");

Ref<Projekt> pOID = Projekte.select_element(
  "select p in Projekte where p.Titel = \"Telekom Statistik\"");

aOID.Projekte.insert_element(pOID);
/* pOID.Abteilungen.insert_element(aOID); AUTOMATISCH DURCH SYSTEM
                                          s. Folie 5.1.23 */
```


OODM: Bewertung

- ❏ Hohe Ausdrucksmächtigkeit bei der Datenstrukturierung
- ❏ Integrierte Verhaltensmodellierung (Methoden, Späte Bindung)
- ❏ Noch keine Standardisierung in Sicht
- ❏ Keine systematische Schemaentwurfsmethodik vorhanden