Persistente Datenhaltung

Programmiersprachen stellen im allgemeinen Operationen zum Datenverkehr mit persistenten "externen" Speichermedien zur Verfügung.

Die Anforderungen an externe Datenhaltung sind aber meist höher als an Datenstrukturen eines Programmes.

Im Folgenden ...

- **motivieren wir die Notwendigkeit von Datenbanksystemen,**
- **stellen eine Methode zum Datenbankentwurf vor (das Entity-Relationship-Modell)**
- **und beschreiben verschiedene logische Datenmodelle, insbesondere das Relationenmodell**

1

Literatur

In P3 behandeln wir nur einige Modellierungsaspekte von Datenbanksystemen. Eine Vertiefung kann im Hauptstudium erfolgen.

Literatur für unsere Zwecke:

C.A. Zehnder: Informationssysteme und Datenbanken, 2. Auflage, Teubner, 1998
[angemessene Knappheit, aber manchmal schlecht erklärt]

A. Meier, T. Wüst: Objektorientierte Datenbanken, dpunkt 1997
[empfehlenswert, gute Motivation für OODBS, Vergleich mit Relationalmodell]

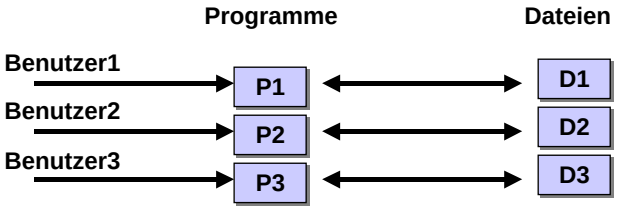
R.C. Lockemann, J.W. Schmidt (Hrsg.): Datenbank-Handbuch, Springer 1993
[wortreiche Darstellungen, trotzdem nicht immer alles Wesentliche]

G. Reese: Database Programming with JDBC and Java, O´Reilly 1997
[Beschreibung des Java-Interfaces zu relationalen Datenbanken]

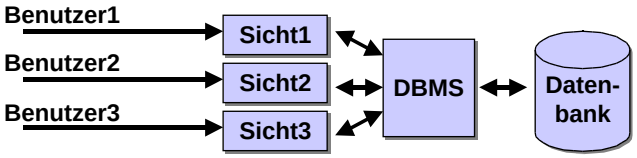
2

Dateisysteme vs. Datenbanksysteme

- **Dateien werden für einzelne Programme entworfen.**
- **Datenbanken bedienen mehrere verschiedene Programme.**



Starke Kopplung zwischen Programm und Daten



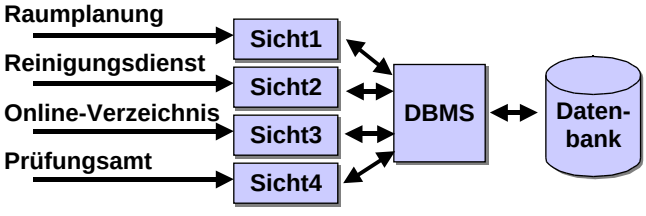
Entkopplung von Programm und Daten

3

Beispiel einer Datenbankanwendung

Sichten auf Lehrveranstaltungsdaten einer Universität

Raumplanung:	Sem Raum Zeit Lehrperson
Reinigungsdienst:	Raum Zeit
Online-Verzeichnis:	Fakultät Sem LV-Nr LV-Name Lehrperson Raum Zeit
Prüfungsamt:	Fakultät Student Sem LV-Nr Note



4

Problembereiche

Möglicher Ausschnitt aus einer Datenbank über Lehrveranstaltungen:

Bettina Meyer	1234711	18020	P3	Neumann	Phil-B	WS99/00	Di	12-14
Bettina Meyer	1234711	18016	F3	Kirsig/Lang	Phil-B	WS99/00	Di	14-16
Axel Schmidt	1255891	18020	P3	Neumann	Phil-B	WS99/00	Di	12-14
Ursel Huber	1344711	18047	Denkmasch.	Neumann	R-031	WS99/00	Di	14-16
• • •								

Dies ist sicherlich nicht die beste Modellierung! Warum?

Im allgemeinen sind zu berücksichtigen:

- **große Datenbestände** ... im Terabytebereich (10^{12} Bytes)
- **Mehrbenutzerbetrieb** ... verteilter, paralleler Zugriff
- **heterogene Benutzer** ... verschiedenen Sichten
- **Redundanz** ... Modellierungsaufgabe
- **Integritätsverletzungen** ... Integritätsbedingungen überwachen
- **Sicherheit gegen Datenverlust** ... Schutzmechanismen
- **Entwicklungskosten** ... Standards, kommerzielle Systeme

5

Grundkonzept von Datenbanken

Eine *Datenbank* ist eine selbständige, für dauerhaften, flexiblen und sicheren Gebrauch ausgelegte Datenorganisation, bestehend aus einem Datenbestand (Datenbasis) und einer Datenverwaltung (DBMS).

Das Datenverwaltungssystem (Database Management System, DBMS) vermittelt zwischen einer Benutzersicht der Daten und der internen Datenorganisation:

- **klare Strukturierung der Daten nach außen**
- **spezifische Datensichten für verschiedene Benutzer**
- **Maßnahmen zur Wahrung der Datenintegrität**
- **Flexibilität**
- **Trennung der internen Organisation von der äußeren Struktur**

Vergleiche mit Abstrakten Datentypen:

Ein DBMS bietet

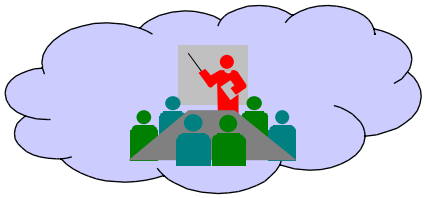
- **ein Datenmodell**
- **Operationen mittels einer Datenmanipulationssprache (DML)**

6

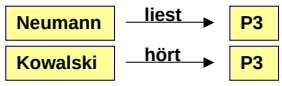
Betrachtungsebenen

Beispiel: Daten über Vorlesungsbetrieb an einer Universität

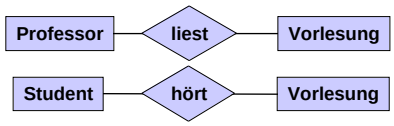
Reale Welt (Ausschnitt)



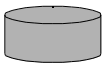
Informationen über Teile der realen Welt, Modelle



Logisches Datensystem, Konzeptuelles Schema, Datenmanipulationssprache



Technisches Datensystem



Dateien, Blöcke

unser Thema!

7

Datenbankentwurf mit dem Entity-Relationship-Modell (ERM)

Modellierung eines Realweltausschnittes durch Identifikation von

- **Gegenständen, Objekten (entities)**
- **Beziehungen (relationships)**

Eine *Entität* ist ein individuelles Exemplar von Elementen der realen Welt (oder Vorstellungswelt).

Gleichartige Entitäten werden zu *Entitätsmengen* gruppiert.

Entitätsmengen können in *Beziehungen* zu einander stehen

Beispiele:

die Hinterachse von HH-PK 475
Helmut Kohl
die LV 18.015 im WS 99/00

Hinterachsen
Politiker
Lehrveranstaltungen

Teil-von
verfolgt-Ziel
findet-statt-in

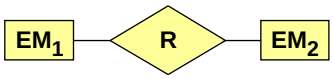
8

Beziehungstypen

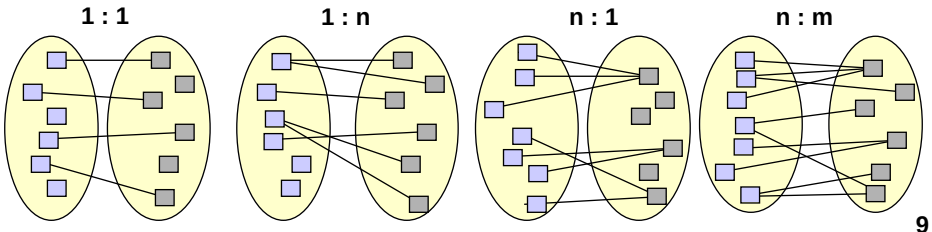
Grundbegriffe:

$EM_1, EM_2, \ldots, EM_n$	Entitätsmengen
$R \subseteq EM_1 \times EM_2 \times \ldots \times EM_n$	Beziehung (Relation)
$t \in R$	Tupel

Graphische Notation:



Man unterscheidet 4 grundsätzliche Beziehungstypen:



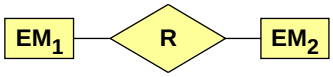
9

Funktionalitätsangaben

Für eine redundanzfreie Gestaltung von Datenbanksystemen ist es wichtig, funktionale Abhängigkeiten zwischen Attributen (oder Gruppen von Attributen) zu identifizieren.

Zweistellige Beziehungen

$R \subseteq EM_1 \times EM_2$



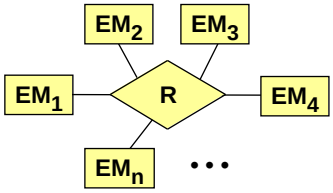
Funktionalitätsangaben:

R: $EM_1 \Rightarrow EM_2$ (Beziehungstypen 1:1 und n:1)

R: $EM_2 \Rightarrow EM_1$ (Beziehungstypen 1:1 und 1:n)

Mehrstellige Beziehungen

$R \subseteq EM_1 \times EM_2 \times \ldots \times EM_n$

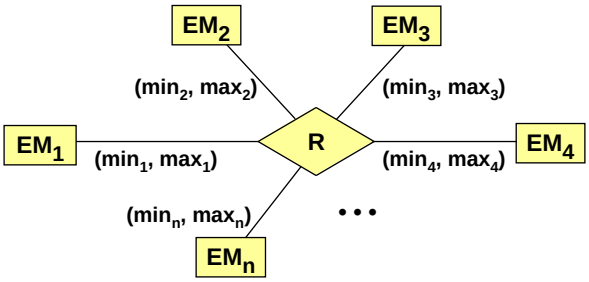


Funktionalitätsangaben:

R: $EM_1 \times \ldots \times EM_{k-1} \times EM_{k+1} \times \ldots \times EM_n \Rightarrow EM_k$

10

(min, max)-Notation



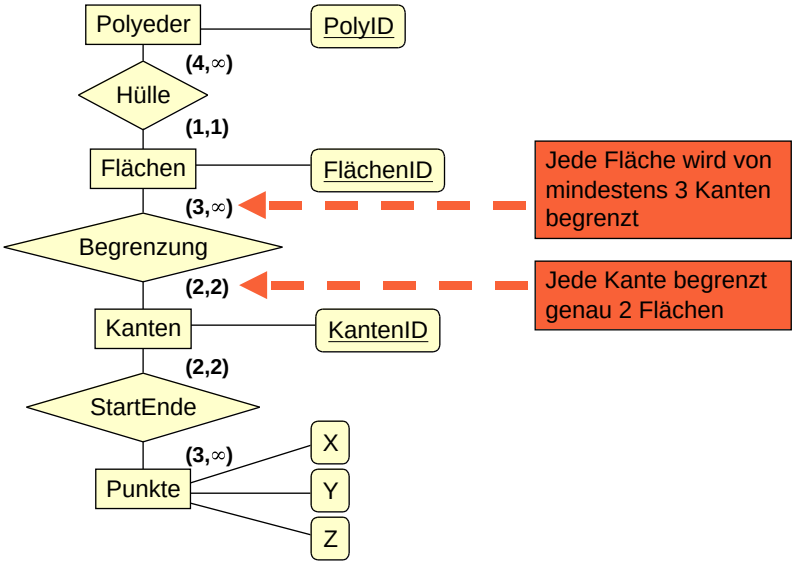
Für jedes e_i aus EM_i gibt es

- **mindestens min_i Tupel der Art (..., e_i, ...)**
- **höchstens max_i Tupel der Art (..., e_i, ...)**

Achtung: Bei der (min, max)-Notation steht die Angabe gegenüber dem (den) betroffenen Attribut(en)

11

Beispiel: Modell für Begrenzungsflächen



12

Kombinierte qualitative Beziehungstypen (1)

Bei praktischen Anwendungen ist es häufig wichtig, zwischen Beziehungen zu unterscheiden, die bestehen müssen oder bestehen können.

Beziehungstyp (EM_1, EM_2)		Entitäten aus EM_2 für jede Entität aus EM_1
1:	einfache Beziehung	genau eine
p:	potentielle Beziehung	keine oder eine
m:	multiple Beziehung	mehrere (≥ 1)
pm:	potentiell multiple Beziehung	keine, eine oder mehrere

Beispiel:

Achtung: Bei dieser qualitativen Kennzeichnung steht die Angabe am betroffenen Attribut.

13

Kombinierte qualitative Beziehungstypen (2)

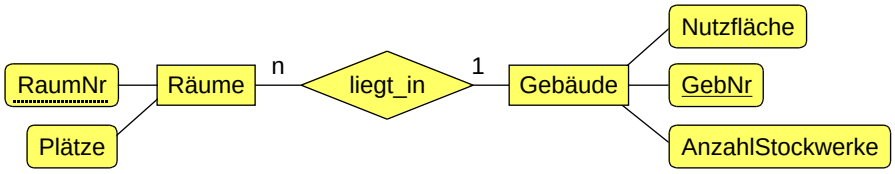
Die 4 qualitativen Beziehungstypen 1, p1, m, pm beschreiben die Beziehung aus der Perspektive EM_1=> EM_2. Durch Kombination mit der Perspektive EM_2=> EM_1 entstehen 10 kombinierte qualitative Beziehungstypen (ohne Spiegelungen).

Typ	Beispiel
1-1	Linker_Schuh - Rechter_Schuh
p-1	Student - Nummer
m-1	Kind - Mutter
pm-1	Kind - Frau
p-p	Ehemann - Ehefrau *(nicht notwendigerweise verheiratet)*
m-p	Parteimitglied - Partei
pm-p	(Unter)projekt - Projekt
m-m	Geschwister - Geschwister
pm-m	Vorlesung- Student
pm-pm	Freund - Freund

14

Schwache Entitäten

Schwache Entitäten hängen von der Existenz starker Entitäten ab.

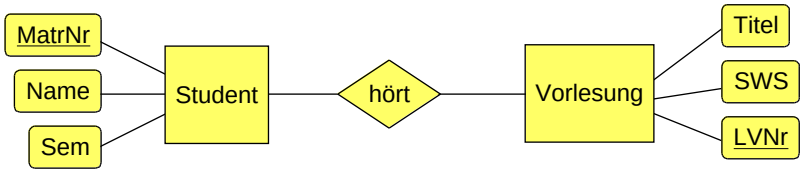


- **Beziehungen zwischen starken und schwachen Entitäten ist immer 1 : n**
- **Schlüssel für schwache Entität setzt sich zusammen aus eigenem Schlüssel und Schlüssel für starke Entität**

15

Ausweisung von Attributen und Schlüsseln

Entitäten werden häufig durch mehrere Attribute beschrieben, z.B. Student mit Name, Matrikelnummer und Semesterzahl.



Ein Attribut oder eine Kombination von Attributen heißt *Schlüssel (Identifikationsschlüssel)*, wenn damit jede Entität einer Entitätsmenge eindeutig identifiziert werden kann.

Schlüssel werden üblicherweise durch *Unterstreichen* gekennzeichnet.

16

Spezialisierung

Die Beziehung is-a beschreibt eine *Teilmengenbeziehung* zwischen zwei Entitätsmengen.

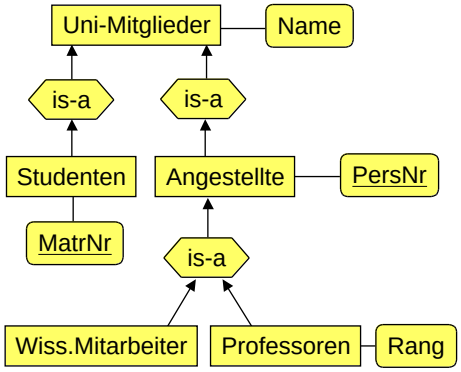
B is-a A:

Entitäten aus B erben alle Attribute und Beziehungen der Entitäten aus A.

B kann durch zusätzliche Attribute und Beziehungen beschrieben werden.

A ist Generalisierung von B.

B ist Spezialisierung von A.



17

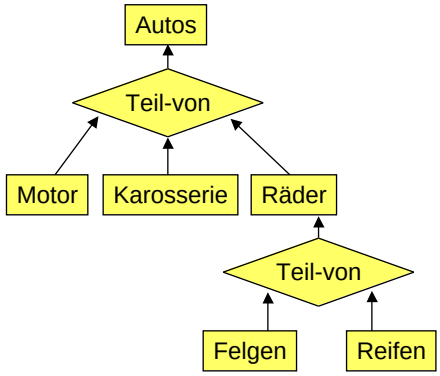
Aggregation

Die Beziehung Teil-von verbindet ein Aggregat mit seinen Teilen.

B Teil-von A:

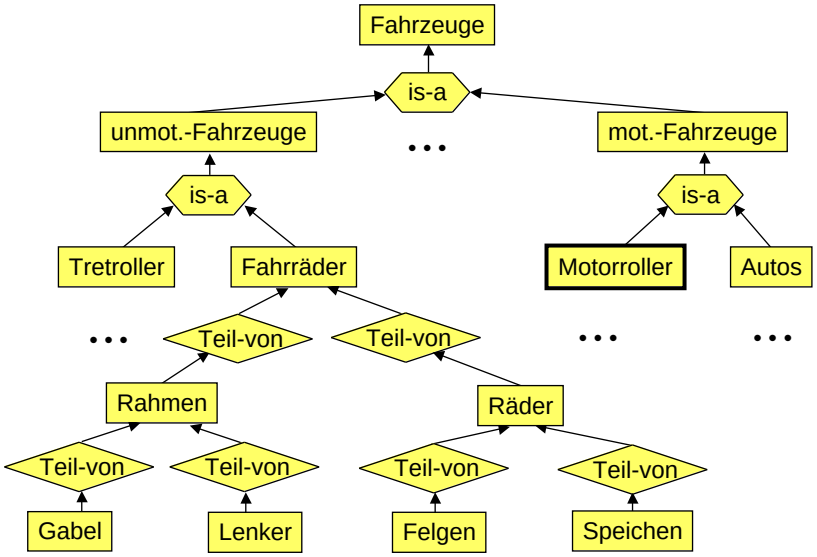
Teil-von (part-of) ist häufig anwendbar, hat aber i.A. keine präzise Bedeutung.

Die Attribute und Beziehungen eines Teils hängen nicht zwingend mit denen des zugehörigen Aggregates zusammen.



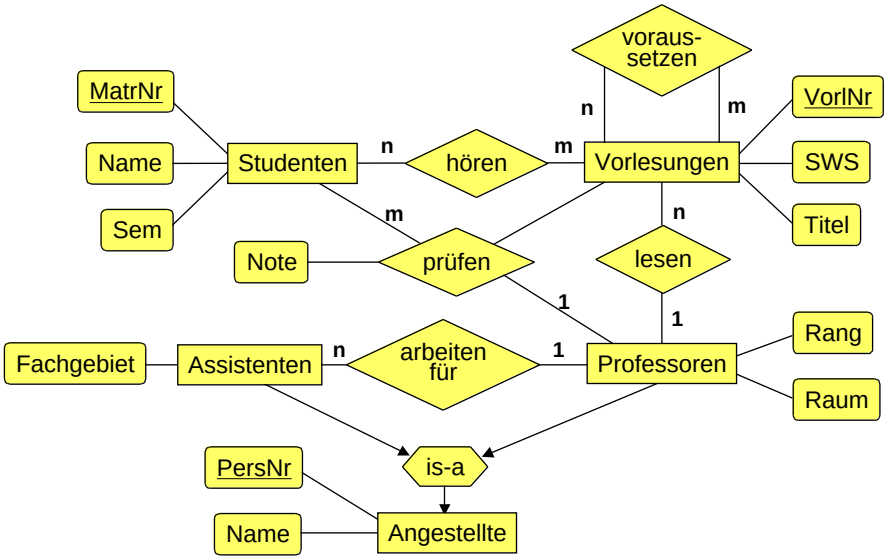
18

Zusammenspiel von Aggregation und Spezialisierung



19

ER-Schema für eine Uni-Datenbank



20

Von der Realwelt zu einem Datenbanksystem

A Abgrenzen des Problembereichs

B Objekte (Entitäten) des Problembereiches identifizieren

C Beziehungen zwischen Objekten (Entitätsmengen) identifizieren

D Indentifikationsschlüssel festlegen

E Konsistenzbedingungen festlegen

F Abbildung auf logisches Datenmodell

G Operationen (Transaktionen) formulieren

21

Datenbankmodelle

Datenbankmodelle unterscheiden sich durch ihre "logische Sicht" auf die Daten, d.h. ihre formale Struktur. Die wichtigsten (teilweise historischen) Arten sind:

- Hierarchisches Datenbankmodell
- Netzwerk-Datenbankmodell
- Relationales Dantenbankmodell
- Objektorientiertes Datenbankmodell
- Deduktives Datenbankmodell

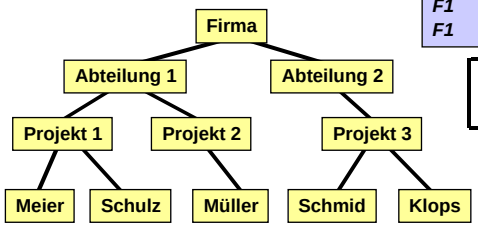
22

Hierarchisches Datenbankmodell

Hierarchische Datenbankmodelle gehören zu den ersten kommerziell angebotenen Datenbanksystemen (z.B. Information Management System IMS von IBM).

Ein hierarchisches Datenbanksystem besteht aus hierarchisch von einander abhängenden Tabellen.

Beispiel:



Firma	*Abteilung*
F1	*A1*
F1	*A2*

Abteilung	*Projekt*
A1	*P1*
A1	*P2*
A2	*P3*

Projekt	*Beginn*	*Ende*	*Mitarbeiter*
P1	*1.3.99*	*30.9.99*	*Meier*
P1	*1.3.99*	*30.9.99*	*Schulz*
P2	*1.10.98*	*31.1.99*	*Müller*
P3	*15.5.98*	*31.12.98*	*Schmid*
P3	*15.5.98*	*31.12.98*	*Klops*

23

Eigenschaften eines hierarchischen Datenbankmodells

Vorteile

- **Zusammengehörige Daten sind gemeinsam zugreifbar**
- **Kurze Zugriffszeiten für festgelegte Suchpfade**

Nachteile

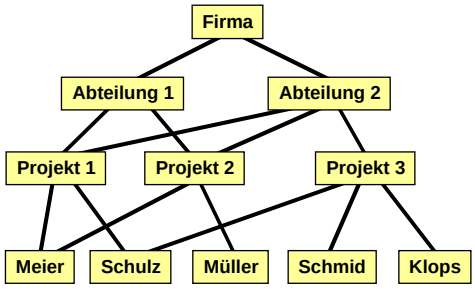
- **Geringe Flexibilität (nur eine Hierarchie)**
- **Vermischung von semantischen Eigenschaften und Speicherstrukturen**
- **Benutzer muß Zugriffspfad kennen**

24

Netzwerkdatenbankmodell

Das Netzwerkdatenbankmodell wurde 1971 von der CODASYL entworfen (COnference on DAta SYstems Languages) und steht in engem Zusammenhang mit COBOL (COmmon Business Oriented Language).

Datenobjekte werden *referentiell* (nicht assoziativ) identifiziert. Im Gegensatz zu hierarchischen Modellen können auch n:m Beziehungen dargestellt werden.



25

Relationales Datenbankmodell

Das Relationale Datenbankmodell wird heute am häufigsten verwendet. ORACLE-Datenbanken, SAP-Systeme und Java Datenbank-Standards verwenden das relationale DB-Modell.

Grundlegende Arbeiten zum Relationenmodell stammen von Codd [Codd 70], unabhängig von Arbeiten zum Entity-Relationship-Modell [Chen 76].

Mathematische Notation:

$D_1, D_2, \ldots, D_n$	Definitionsbereiche (Domänen)
$R \subseteq D_1 \times D_2 \times \ldots \times D_n$	Relation (Beziehung)
$t \in R$	Tupel

Notation für Datenbank-Relationen:

$A_1, A_2, \ldots, A_n$	Attribute
$D_1, D_2, \ldots, D_n$	primitive Datentypen
$R \in Rel(A_1:D_1, \ldots, A_n:D_n)$	Relation über den Attributen $A_1, \ldots, A_n$ mit den Domänen $D_1, \ldots, D_n$

26

Relationen als Tabellen

Benutzer hat logische Sicht in Form von Tabellen:

Relation "hat-Projekt"

Abteilung	*Projekt*
A1	*P1*
A1	*P2*
A2	*P1*
A2	*P2*
A2	*P3*

Attribute

Werte eines primitiven Datentyps

Relation "hat-Tätigkeit"

Mitarb.-Nr	*Name*	*Tätigkeit*
M1	*Meier*	*Programmierer*
M2	*Schulz*	*DB-Administrator*
M3	*Müller*	*Kontrolleur*
M4	*Schmid*	*Programmierer*
M5	*Klops*	*Projektleiter*

Relation "hat-Mitarbeiter"

Projekt	*Mitarbeiter*
P1	*Meier*
P1	*Schulz*
P2	*Müller*
P3	*Schmid*
P3	*Klops*

27

Schlüssel

Für jede Relation muß ein *Identifikationsschlüssel* existieren.

Ein Identifikationsschlüssel ist ein Attribut oder eine minimale Attributkombination,

- **die jedes Tupel einer Relation eindeutig identifiziert und**
- **deren Wert sich während der Existenz des Tupels nicht ändert.**

Beispiel: R_{Person}: Name x Geburtsdatum x ID-Nr

$R_{Ortschaft}$: Breitengrad x Längengrad x Name

Konsequenz:

Jedes Attribut einer Relation ist vom Identifikationsschlüssel funktional abhängig.

Kein Attribut aus dem Identifikationsschlüssel ist von den übrigen Attributen des Identifikationsschlüssels funktional abhängig.

28

1. Normalform (1)

Eine Relation befindet sich in 1. Normalform, wenn die Attribute nur einfache (skalare, nicht zusammengesetzte) Attributwerte aufweisen.

Nicht in 1. Normalform:

Ehefrau	*ID1*	*Ehemann*	*ID2*	*Kinder*
Anna_Meier	12345711	Otto_Meier	43214711	Emil, Katrin, Lisa

Angestellter	*ID*	*Geburtstag*	*Geburtsort*
Otto_Meier	43214711	16.12.62	Lüneburg

Lexikoneintrag	*Erklärungstext*
Polyeder	Vielflächner, von Vielecken begrenzter Körper

29

1. Normalform (2)

Überführung in die 1. Normalform durch Zerlegen von zusammengesetzten Attributen.

Menge => mehrere Tupel
Verbund => mehrere Attribute

Zerlegen einer Menge:

Ehefrau	*ID1*	*Ehemann*	*ID2*	*Kinder*
Anna_Meier	12345711	Otto_Meier	43214711	Emil, Katrin, Lisa

Ehefrau	*ID1*	*Ehemann*	*ID2*	*Kind*	*ID3*
Anna_Meier	12345711	Otto_Meier	43214711	Emil	65434711
Anna_Meier	12345711	Otto_Meier	43214711	Katrin	76544711
Anna_Meier	12345711	Otto_Meier	43214711	Lisa	87654711

Die 1. Normalform erfordert hier einen zusätzlichen Schlüssel.

30

1. Normalform (3)

Zerlegen eines Verbundes:

Angestellter	*ID*	*Geburtstag*	*Geburtsort*
Otto_Meier	43214711	16.12.62	Lüneburg

Angestellter	*ID*	*G_tag*	*G_monat*	*G_jahr*	*Geburtsort*
Otto_Meier	43214711	16	12	62	Lüneburg

Die Zerlegung ist nur dann notwendig, wenn auf die Teile einzeln zugegriffen werden soll.

31

2. Normalform (1)

Eine Relation befindet sich in 2. Normalform, wenn sie

- **in 1. Normalform ist und**
- **jedes nicht zum Identifikationsschlüssel gehörige Attribut funktional vom Identifikationsschlüssel aber nicht von einem anderen Attribut abhängig ist.**

Nicht in 2. Normalform:

Ehefrau	*ID1*	*Ehemann*	*ID2*	*Kind*	*ID3*
Anna_Meier	12345711	Otto_Meier	43214711	Emil	65434711
Anna_Meier	12345711	Otto_Meier	43214711	Katrin	76544711
Anna_Meier	12345711	Otto_Meier	43214711	Lisa	87654711

Attributwerte von *Ehefrau* hängen funktional von den Attributwerten von *ID1* ab, ebenso *Ehemann* von *ID2*.

Die 2. Normalform erzwingt Gruppierung in Sachgebiete und eliminiert Redundanzen

32

2. Normalform (2)

Überführung in die 2. Normalform durch Zerlegen der Relation in Teilrelationen

Ehefrau	*ID1*	*Ehemann*	*ID2*	*Kind*	*ID3*
Anna_Meier	12345711	Otto_Meier	43214711	Emil	65434711
Anna_Meier	12345711	Otto_Meier	43214711	Katrin	76544711
Anna_Meier	12345711	Otto_Meier	43214711	Lisa	87654711

Ehefrau	*ID*	*Ehemann*	*ID2*
Anna_Meier	12345711	Otto_Meier	43214711

ID1	*ID2*	*Kind*	*ID3*
12345711	43214711	Emil	65434711
12345711	43214711	Katrin	76544711
12345711	43214711	Lisa	87654711

33

3. Normalform (1)

Eine Relation befindet sich in 3. Normalform, wenn sie

- **in 2. Normalform ist und**
- **kein nicht zum Identifikationsschlüssel gehöriges Attribut *transitiv* (d.h. über andere Attribute) vom Identifikationsschlüssel abhängt.**

Nicht in 3. Normalform:

ID1	*ID2*	*Kind*	*ID3*
12345711	43214711	Emil	65434711
12345711	43214711	Katrin	76544711
12345711	43214711	Lisa	87654711

Unter der Annahme, daß Frauen nur einen Ehemann haben und dieser Vater der Kinder ist, hängt *ID1* über *ID2* vom Identifikationsschlüssel *ID3* ab.

34

3. Normalform (2)

Überführung in die 3. Normalform durch Zerlegen der Relation in Teilrelationen

ID1	*ID2*	*Kind*	*ID3*
12345711	43214711	Emil	65434711
12345711	43214711	Katrin	76544711
12345711	43214711	Lisa	87654711

ID1	*Kind*	*ID3*
12345711	Emil	65434711
12345711	Katrin	76544711
12345711	Lisa	87654711

ID1	*ID2*
12345711	43214711

Relationen in der 3. Normalform heißen oft kurz "normalisiert".

35

Globale Normalisierung

Um redundante Datenspeicherung zu vermeiden, müssen Relation auch gemeinsam betrachtet werden.

Beispiel:

STUDENTEN			
MatrNr	*Name*	*Fach*	*Sem*

P3-ÜBUNGSTEILNEHMER			
MatrNr	*Name*	*Fach*	*Punkte*

Die Attribute *Name* und *Fach* sind offenbar redundant gespeichert. Dagegen ist *MatrNr* offenbar als Schlüssel mehrfach erforderlich.

Ein Attribut heißt *global*, wenn es mindestens in einer Relation als Schlüssel vorkommt.

Ein Attribut heißt *lokal*, wenn es nur in einer Relation und dort nicht als Schlüssel vorkommt.

Eine global normalisierte Datenbasis muß in 3. Normalform sein und nur Lokal- oder Global-Attribute enthalten.

36

Datenmanipulationssprachen

Ein DBMS stellt Operationen zur Manipulation eines Datenbestandes zur Verfügung: eine *Datenmanipulationssprache* (Data Manipulation Language, DML).

Eine DML soll möglichst in der Lage sein, *alle in der DB enthaltenen Informationen* abzufragen, zu ergänzen oder zu ändern.

Eine DML kann nach verschiedenen Gesichtspunkten charakterisiert werden:

- **selbständig / eingebettet**
- **deskriptiv / prozedural**
- **mengenbezogen / tupelbezogen**
- **zugeschnitten für ein bestimmtes Datenmodell**

37

Relationenalgebra

- **Operationen wurden 1972 für das Relationenmodell entworfen.**
- **Alle Fragen an eine DB sind ausdrückbar.**
- **Eine äquivalente DML heißt *relational vollständig*.**

Übersicht über die Operationen auf Relationen

Mengenoperationen angewandt auf Relationen:
Vereinigung, Differenz, Durchschnitt, symmetrische Differenz

Erweiterungen für "Relationenalgebra":
Projektion, Theta-Verbund, natürlicher Verbund,, Restriktion, Division

38

Projektion

$A = \{A_1, \ldots, A_n\}$ Attribute einer Relation R

Die Projektion von R auf eine Attributmenge $A' \subseteq A$ ist die Relation, die durch Weglassen aller Attribute $A_i \notin A'$ entsteht.

Eine Projektion verringert also die Dimensionalität einer Relation (vergl. Projektion von 3D auf 2D). Von gleichen Tupeln wird nur ein Exemplar bewahrt.

Beispiel:

Ehefrau	*ID1*	*Ehemann*	*ID2*	*Kind*	*ID3*
Anna_Meier	12345711	Otto_Meier	43214711	Emil	65434711
Anna_Meier	12345711	Otto_Meier	43214711	Katrin	76544711
Anna_Meier	12345711	Otto_Meier	43214711	Lisa	87654711

Projektion auf {*Ehefrau, ID1, Ehemann, ID2* } ergibt:

Ehefrau	*ID1*	*Ehemann*	*ID2*
Anna_Meier	12345711	Otto_Meier	43214711

39

Natürlicher Verbund (Join)

Es seien $A = \{A_1, \ldots, A_k\}$, $B = \{B_1, \ldots, B_m\}$, $C = \{C_1, \ldots, C_n\}$ Attributmengen, $A \cup B$ Attribute einer Relation R_1, $A \cup C$ Attribute einer Relation R_2.

Der *natürliche Verbund* der Relationen R1 und R2 ist die Relation R mit den Attributen $A \cup B \cup C$ und den Tupeln aus R1 und R2, die gleiche Werte in A haben.

Raum	*Tag*	*Zeit*	*Lehrperson*
Phil-A	Di	8-10	Kurz
Phil-A	Di	10-12	Oberquelle
Phil-A	Di	12-14	Neumann
Phil-A	Di	14-16	Lang
Phil-A	Di	16-18	Müller

Raum	*Tag*	*Zeit*	*LV-Nr*
D-125	Di	8-10	18.034
Phil-A	Di	10-12	18.402
Phil-A	Di	12-14	18.011
Phil-A	Di	14-16	18.013
F-334	Mi	16-18	18.214

Raum	*Tag*	*Zeit*	*Lehrperson*	*LV-Nr*
Phil-A	Di	10-12	Oberquelle	18.402
Phil-A	Di	12-14	Neumann	18.011
Phil-A	Di	14-16	Lang	18.013

40

Theta-Verbund

Der Theta-Verbund ist eine Verallgemeinerung des natürlichen Verbundes.

Es seien $A = \{A_1, \ldots, A_k\}$, $B = \{B_1, \ldots, B_m\}$, $A' = \{A'_1, \ldots, A'_k\}$, $C = \{C_1, \ldots, C_n\}$ Attributmengen, $A \cup B$ Attribute einer Relation R_1, $A' \cup C$ Attribute einer Relation R_2. A und A´ seien paarweise vergleichbar.

Der *Theta-Verbund* der Relationen R1 und R2 ist die Relation R mit den Attributen $A \cup B \cup A' \cup C$ und den Tupeln aus R1 und R2, für die die Werte in A und A´ durch Vergleichsoperatoren beschriebenen Bedingungen genügen.

Mögliche Vergleichsoperatoren: = <> < > ≤ ≥

41

Beispiel für Theta-Verbund

Studenten, deren Semesterzahl größer als die Regelstudienzeit ihres Studienfaches ist.

MatrNr	*Sem*	*Fach*
124711	6	Informatik
132457	4	BWL
132457	23	Germanistik
142346	29	Informatik

Fach	*RegelSem*
Informatik	9
BWL	8
Germanistik	10
Physik	10

MatrNr	*Sem*	*Fach*	*RegelSem*
132457	23	Germanistik	10
142346	29	Informatik	9

42

Kreuzprodukt

Das Kreuzprodukt zweier Relationen kann als Sonderfall eines Theta-Verbundes mit leerer (= für alle Tupel erfüllter) Bedingung oder als Natürlicher Verbund ohne gemeinsame Attribute betrachtet werden.

Das Kreuzprodukt zweier Relationen R_1 und R_2 mit Attributen A_1 bzw. A_2 ist eine Relation $R = R_1 \times R_2$ mit Attributen $A = A_1 \cup A_2$, deren Tupel aus allen Kombinationen der Tupel von A_1 und A_2 bestehen.

Beispiel:

MatrNr	*Sem*	*Fach*
124711	6	Informatik
132457	4	BWL

X

Beruf	*Verdienst*
Berater	100.000,-
Programmierer	80.000,-
Manager	200.000,-

MatrNr	*Sem*	*Fach*	*Beruf*	*Verdienst*
124711	6	Informatik	Berater	100.000,-
124711	6	Informatik	Programmierer	80.000,-
124711	6	Informatik	Manager	200.000,-
132457	4	BWL	Berater	100.000,-
132457	4	BWL	Programmierer	80.000,-
132457	4	BWL	Manager	200.000,-

43

Restriktion

Auswahl aus einer Relation durch Vorgabe einer Vergleichsbedingung.

Die Restriktion einer Relation R bezüglich einer Bedingung $p(A_1 \ldots A_n)$ enthält diejenigen Tupel von R, für die die Bedingung erfüllt ist.

Beispiel:

Studenten, deren Semesterzahl größer als die Regelstudienzeit ihres Studienfaches ist.

Vorgabe der Bedingung:

Sem > RegelSem

MatrNr	*Sem*	*Fach*	*RegelSem*
124711	6	Informatik	9
132457	4	BWL	8
132457	23	Germanistik	10
142346	29	Informatik	9

MatrNr	*Sem*	*Fach*	*RegelSem*
132457	23	Germanistik	10
142346	29	Informatik	9

44

Division

Auswahl von Tupeln einer Relation R_1, von denen die Werte einer Attributkombination alle entsprechenden Werte in einer Relation R_2 abdecken.

Es seien $A = \{A_1, \ldots, A_k\}$, $B = \{B_1, \ldots, B_m\}$, $C = \{C_1, \ldots, C_n\}$ Attributmengen, $A \cup B$ Attribute einer Relation R_1, $A \cup C$ Attribute einer Relation R_2.

Die Division von R_1 durch R_2 bezüglich A ist eine Relation R mit Attributen B und den Tupeln aus R_1, deren Wertekombinationen für die Attribute B mit allen Wertekombinationen von A verbunden sind, die in R_2 vorkommen.

Beispiel:

Gesucht sind Lieferanten aus R_L, die alle Teile in R_T liefern können.

Division von R_L durch R_T bezüglich TeilNr ergibt $R_{L/T}$.

R_L

Lieferant	*TeilNr*
Otto	1
Otto	2
Otto	3
Bahr	2
Bahr	3
Quelle	1
Quelle	2
Quelle	3

R_T

TeilNr	*Name*	*Typ*
1	Reifen	XL-007
2	Felge	Al-1008
3	Kappe	RK4711

$R_{L/T}$

Lieferant
Otto
Quelle

45

Die Datenmanipulationssprache SQL

Durch Kombination der Operationen der Relationenalgebra kann grundsätzlich auf jede Information einer relationalen Datenbank zugegriffen werden. Die mengenalgebraischen Formulierungen sind allerdings häufig schwer lesbar. Deshalb sind relational äquivalente, aber leichter verständliche DML entwickelt worden.

SQL (Structured Query Language) wurde um 1975 von IBM unter dem Namen SEQUEL für relationale Datenbanksysteme entwickelt, 1980 als SQL für IBMs DB2 eingeführt und von ORACLE übernommen. SQL ist heute die am weitesten verbreitete DML.

Grundstruktur einer SQL-Abfrage:

```
SELECT<Attributliste>
FROM   <Relation(en)>
WHERE <Bedingungen>
```

Bildbereich der Anfrage, Projektion auf diese Attribute

Definitionsbereich der Anfrage

Auswahlkriterium, Ausdrücke aus dem Prädikatenkalkül

46

Funktionalität von SQL

Die wichtigsten Sprachelemente

Kreieren einer Datenbanktabelle (Relation):

CREATE TABLE <Name> (
<Attribut$_1$> <Datentyp$_1$> <Wertebedingung$_1$>, ← 1. Attribut ist Schlüssel
<Attribut$_2$> <Datentyp$_2$> <Wertebedingung$_2$>,
... ,
<Attribut$_n$> <Datentyp$_n$> <Wertebedingung$_n$>);

Wertebedingung:

NOT NULL Attribut muß Wert haben

Anweisungstypen:	**Logische Operatoren:**	**Vergleichsoperatoren:**
UPDATE	AND	= < <= > >=
DELETE	OR	BETWEEN ... AND ....
INSERT	NOT	
	IN	

47

SQL-Beispiele (1)

Studenten

MatrNr	*Name*	*Fach*	*Sem*
123456	Hans Meier	Informatik	3
124711	Klaus Schmid	BWL	5
134256	Anne Döring	Informatik	3
145267	Fred Fuchs	Musik	13
145627	Fritz Teger	Informatik	5
132456	Katrin Scheu	BWL	7

Preisträger

Rang	*Name*	*MatrNr*
1	Anne Döring	134256
2	Klaus Schmid	124711
3	Fritz Teger	145627

Namen aller Preisträger mit Fach Informatik:

```
SELECT Name
FROM Preisträger
WHERE MatrNr IN
        SELECT MatrNr
        FROM Studenten
        WHERE Fach = ´Informatik´
```

Semesterzahl des Preisträgers mit Rang 1:

```
SELECT Sem
FROM Studenten
WHERE MatrNr =
        (SELECT MatrNr
        FROM Preisträger
        WHERE Rang = 1)
```

48

SQL-Beispiele (2)

Ändere das Fach eines Studenten mit bekannter MatrNr:

```
UPDATE Studenten
SET Fach = ´Informatik´
WHERE MatrNr = 145267
```

Erhöhe die Semesterzahl aller Studenten:

```
UPDATE Studenten
SET Sem = Sem + 1
```

Entferne alle Studenten mit Semesterzahl > 20:

```
DELETE
FROM Studenten
WHERE Sem > 20
```

Füge neuen Datensatz in Studenten ein:

```
INSERT
INTO Studenten (MatrNr, Name, Sem)
VALUES (132451, ´Lisa Klug´, 1)
```

49

Sichten in SQL

Sichten dienen dazu, verschiedenen Benutzerkreisen die jeweilig gewünschte (oder restringierte) Teilmenge eines Datenbestandes zugänglich zu machen.

Sichten werden in SQL durch eine Auswahlanweisung definiert.

Beispiel:

Kreiere positive Sicht auf Studenten:

```
CREATE VIEW Pos_Studenten AS
(SELECT Name, Fach, Sem
FROM Studenten
WHERE Sem < 12)
```

Sichten sind temporäre Tabellen, die etwaigen Änderungen der zugrundeliegenden Tabellen automatisch folgen.

50

Integritätsbedingungen

Datenintegrität bedeutet:

Daten genügen
- logischen Konsistenzbedingungen
- den Anforderungen der modellierten Welt
- den Anforderungen des Datenbanksystems

SQL erlaubt die Formulierung von Integritätsbedingungen, die vom DBMS überprüft werden.

		CREATE TABLE Inf-Studenten (...
1.	Angabe eines Primärschlüssels	PRIMARY KEY (MatrNr)
2.	Angabe eines Alternativschlüssels	UNIQUE (Name, Vorname, Geburtstag)
3.	Angabe eines Fremdschlüssels	FOREIGN KEY (MatrNr) REFERENCES Studenten
4.	Kardinalität von Attributwerten	NOT NULL, UNIQUE

51

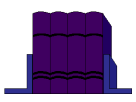
Legende für SQL-Syntax

{ A }	A ist fakultativ
{ A | B }	A oder B können vorgegeben werden
{ A }...	A kann keinmal, einmal oder mehrmals (dann durch Kommata getrennt) vorgegeben werden
{ A | B }...	A oder B können keinmal, einmal oder mehrmals (dann durch Kommata getrennt) vorgegeben werden
[A | B]	A oder B müssen vorgegeben werden
[A | B]...	A oder B müssen mindestens einmal oder können mehrmals (dann durch Kommata getrennt) vorgegeben werden

CREATE TABLE-Anweisung ::=
CREATE TABLE Tabellenname ([Spaltenname Datentyp { NOT NULL }]...)
{ IN Datenbankspeicherbereichsname }

erlaubt z.B. folgende Anweisung:

```
CREATE TABLE Studententabelle
    (MatrNr   SMALLINT    NOT NULL,
    Name      CHAR (15)   NOT NULL,
    Sem       SMALLINT    NOT NULL,
    Tel       CHAR (10)   )
```



Rick F. van der Lans
Das SQL Lehrbuch
Addison Wesley 1987

Rick F. van der Lans
Introduction to SQL
Addison Wesley 2000

52

Auszüge aus der SQL-Syntax (1)

CREATE TABLE-Anweisung ::=
CREATE TABLE Tabellenname
([Spaltenname Datentyp { NOT NULL }]...)
{ IN Datenbankspeicherbereichsname }

CREATE DBSPACE-Anweisung ::=
CREATE [PUBLIC | PRIVATE] DBSPACE
NAMED Datenbankspeicherbereichsname }
{ ([Parameter]...) }

Datentyp ::=
[SMALLINT | INTEGER | FLOAT
| DECIMAL (ganzzahlige Konstante , ganzzahlige Konstante)
| CHAR (ganzzahlige Konstante)
| VARCHAR (ganzzahlige Konstante)]

CREATE VIEW-Anweisung ::=
CREATE VIEW Viewname
{ ([Spaltenname]...) } AS SELECT-Anweisung

53

Auszüge aus der SQL-Syntax (2)

UPDATE-Anweisung ::=
UPDATE Tabellenspezifikation
SET [Spaltenname = [Ausdruck | NULL]]...
{ WHERE Bedingung }

INSERT-Anweisung ::=
INSERT INTO Tabellenspezifikation { ([Spaltenname]...) }
[VALUES ([Konstante | NULL]...)
| SELECT-Anweisung

DELETE-Anweisung ::=
DELETE FROM Tabellenspezifikation
{ WHERE Bedingung }

54

Auszüge aus der SQL-Syntax (3)

```
SELECT-Anweisung ::=
        SELECT { DISTINCT | UNIQUE | ALL } [ * | [ Spaltenausdruck ]... ]
        FROM [ Tabellenspezifikation { Pseudonym } ]...
        { WHERE Bedingung }
        { GROUP BY [ Spaltenspezifikation ]... { HAVING Having-Bedingung } }
        { ORDER BY [ Sortierung ]... }
SELECT-Anweisung-mit-UNION ::=
        SELECT [ * | [ Spaltenausdruck ]... ]
        FROM [ Tabellenspezifikation { Pseudonym } ]...
        { WHERE Bedingung }
        { GROUP BY [ Spaltenspezifikation ]... { HAVING Having-Bedingung } }
        { ORDER BY [ Sortierung ]... }
        { UNION  SELECT [ * | [ Spaltenausdruck ]... ]
                 FROM [ Tabellenspezifikation { Pseudonym } ]...
                 { WHERE Bedingung }
                 { GROUP BY [ Spaltenspezifikation ]...
                          { HAVING Having-Bedingung } } }
        { ORDER BY [ Sortierung ]... }
```

55

Auszüge aus der SQL-Syntax (4)

```
Bedingung ::=
        [ Ausdruck Vergleichsoperator [ Ausdruck | Unterabfrage ]
        | Ausdruck { NOT } BETWEEN Ausdruck AND Ausdruck
        | Ausdruck { NOT } IN ( [ Konstante ]... }
        | Ausdruck { NOT } IN ( [ Unterabfrage ]... }
        | Ausdruck Vergleichsoperator [ ANY | ALL ] ( Unterabfrage )
        | alphanumerischer Ausdruck { NOT } LIKE Schablone
        | Having-Ausdruck IS { NOT } NULL
        | EXISTS ( Unterabfrage )
        | NOT Bedingung
        | Bedingung [ AND | OR ] Bedingung
        | ( Bedingung ) ]
Tabellenspezifikation ::=
        [ Tabellenname | Viewname | Eigner.Tabellenname | Eigner.Viewname ]
Spaltenspezifikation ::=
        [ Spaltenname | Tabellenspezifikation.Spaltenname ]
Spaltenausdruck ::=
        [ Tabellenspezifikation.* | Ausdruck | Funktion | NULL ]
Ausdruck ::=
        [ numerischer Ausdruck | alphanumerischer Ausdruck ]
```

56

Auszüge aus der SQL-Syntax (5)

Unterabfrage ::=
SELECT Spaltenausdruck
FROM [Tabellenspezifikation { Pseudonym }]...
{ WHERE Bedingung }
{ GROUP BY [Spaltenspezifikation]... { HAVING Having-Bedingung } }
{ ORDER BY [Sortierung]... }

Having-Bedingung ::=
[Having-Ausdruck Vergleichsoperator [Ausdruck | Unterabfrage]
| Having-Ausdruck { NOT } BETWEEN Ausdruck AND Ausdruck
| Having-Ausdruck { NOT } IN ([Konstante]... }
| Having-Ausdruck { NOT } IN ([Unterabfrage]... }
| Having-Ausdruck Vergleichsoperator [ANY | ALL] (Unterabfrage)
| alphanumerischer Ausdruck { NOT } LIKE Schablone
| Having-Ausdruck IS { NOT } NULL
| EXISTS (Unterabfrage)
| NOT Having-Bedingung
| Having-Bedingung [AND | OR] Having-Bedingung
| (Having-Bedingung)]

Having-Ausdruck ::=
Ausdruck | Funktion

57

Auszüge aus der SQL-Syntax (6)

alphanumerischer Ausdruck ::=
[alphanumerische Konstante
| Spaltenspezifikation
| USER | DATE | TIME]

numerischer Ausdruck ::=
[numerische Konstante
| Spaltenspezifikation
| numerischer Ausdruck numerischer Operator numerischer Ausdruck
| (numerischer Ausdruck)]

Funktion ::=
[[COUNT | MIN | MAX | SUM | AVG] ([DISTINCT] Ausdruck) | COUNT (*)]

numerischer Operator ::=
[* | / | + | -]

Vergleichsoperator ::=
[= | < | > | ≤ | ≥ | <>]

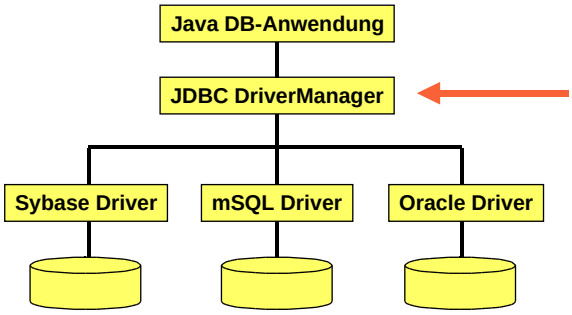
Sortierung ::=
[Spaltenspezifikation | Folgenummer] { ASC | DESC }

58

Datenbankzugriff mit Java

Ab Version 1.1 enthalten die Java Libraries das JDBC API
(Java Database Connectivity Application Programming Interface).

- SQL-Anweisungen aus einem Java-Programm heraus
- Anschluß gängiger kommerzieller Datenbanksysteme



Der JDBC DriverManager schirmt das Anwendungsprogramm vom konkret verwendeten Datenbanktyp ab.

Dasselbe Programm kann für verschiedene DB-Typen verwendet werden.

59

Objektorientierte Datenbanken

Relationale Datenbanken zwingen den DB-Programmierer dazu, komplexe Datenobjekte in "flachen" Tabellen abzulegen.

=> Kluft zwischen objektortientierter Programmiersprache und Datenbanken ("Impedance Mismatch")

Objektorientierte Datenbanksysteme (OODBS) ermöglichen

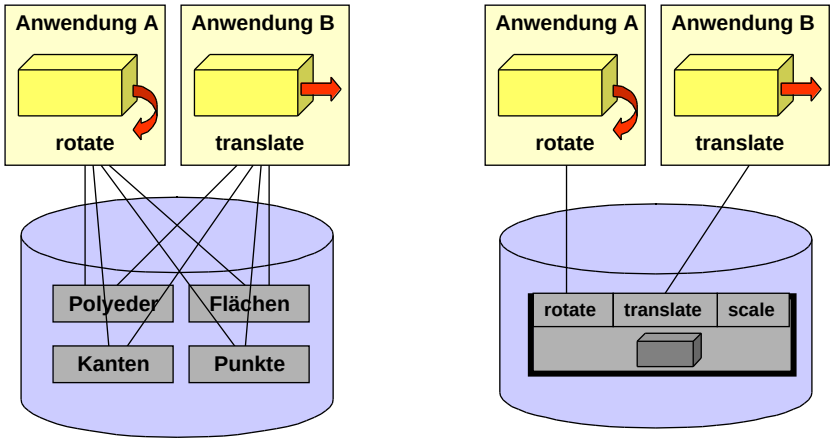
- durchgängig objektorientierten Entwurf und Implementierung,
- Wiederverwendung von DB-Konstrukten,
- Modellierung komplex strukturierter Daten.

OODBS gelten als die Zukunftstechnologie für Datenbanksysteme.

60

Relationale Datenbanken sind nicht an objektorientierte Programmierung angepaßt

Beispiel: Operationen auf Polyedern



relationale Datenbasis

objektorientierte Datenbasis

61

Eigenschaften von OODBS

1989 von einer internationalen Forschergruppe festgelegt:

13 Regeln für ein objektorientiertes Datenbanksystem

Prinzipien der Objektorientierung

1. Komplexe Objekte
2. Objektidentität
3. Datenkapselung
4. Typen und Klassen
5. Vererbung
6. Polymorphismus
7. Vollständigkeit
8. Erweiterbarkeit

Datenbankgrundsätze

9. Dauerhaftigkeit
10. Große Datenbestände
11. Mehrbenutzerbetrieb
12. Rekonstruierbarkeit
13. Ad-hoc-Abfragemöglichkeit

62

Prinzipien der Objektorientierung für ein OODBS

Regel 1: Komplexe Objekte
Attribute von strukturierten Objekten können wiederum strukturierte Objekte sein

Regel 2: Objektidentität
Jedes Objekt trägt eindeutige Identität unabhängig von Attributen

Regel 3: Datenkapselung
Verbergen von Implementierungsdetails, Zugriff über Methoden

Regel 4: Typen und Klassen
Bereitstellen von Struktur- und Verhaltensbeschreibungen

Regel 5: Vererbung
Unterstützung der Wiederverwendbarkeit

Regel 6: Polymorphismus
Anwendung derselben Methodennamen auf Objekte unterschiedlicher Klassen

Regel 7: Vollständigkeit
Jede DB-Operation muß ausgedrückt werden können

Regel 8: Erweiterbarkeit
Keine Unterscheidung zwischen benutzer- und systemdefinierten Typen

63

Datenbankgrundsätze für ein OODBS

Regel 9: Dauerhaftigkeit
Übliche Persistenzforderung an DB-Systeme

Regel 10: Große Datenbestände
Unterstützung von effektivem Zugriff durch Indexmechanismen etc.

Regel 11: Mehrbenutzerbetrieb
Konsistenzwahrung bei nebenläufigen Zugriffen mehrerer Benutzer

Regel 12: Rekonstruierbarkeit
Wiederherstellung eines konsistenten DB-Zustands im Fehlerfall

Regel 13: Ad-hoc-Abfragemöglichkeiten
Deklarative Sprachkonstrukte zur eingebetteten oder interaktiven Formulierung beliebiger Abfragen

64

Modellierung der Realwelt für ein OODBS

Verfeinerungen der Entity-Relationship-Modellierung:

- **Objekte (Entitäten) sind inhärent eindeutig identifizierbar und verfügen über Attribute und Methoden**
- **Objekte einer Klasse verfügen über dieselben Attribute und Methoden**
- **Attribute und Methoden von Objekten einer Klasse können vererbt werden**
- **Beziehungen zwischen den Objekten zweier Klassen werden durch je 2 Assoziationen mit gerichteter Bedeutung ausgedrückt**
- **Beziehungen können Beziehungsklassen angehören und über Attribute und Methoden verfügen**
- **Klassen können klassenbezogene Attribute und Methoden besitzen**

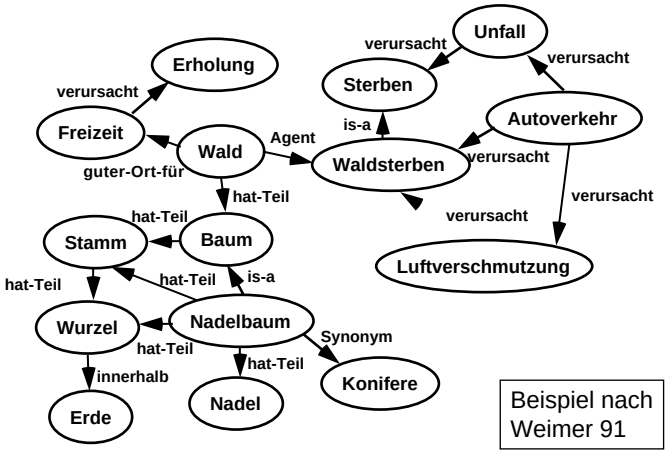
65

Vergleich mit semantischen Netzen

Assoziative Beziehungen zwischen Konzepten, ursprünglich als Modell für das menschliche Gedächtnis entworfen

Knoten:
Konzepte, Begriffe, Kategorien

Kanten:
gerichtete und benannte zweistellige Beziehungen



Beispiel nach Weimer 91

66

Eindeutige Objektidentifikation

Die Identität ist eine inhärente Eigenschaft eines Objektes.

=> Eindeutige Objektidentifikation durch *Surrogate* ("Stellvertreter")

OID	*Seriennummer*	*Typ*	*Preis*
341	142571528931	BMW-312	44890
251	526341562836	VW-Polo	23490

vom System generierte Objektidentifikatoren (OIDs)

Zugriffsschlüssel und Indexstrukturen können zusätzlich kreiert werden.

67

ODMG-Standardisierung

ODMG (Object Database Management Group) definiert Richtlinien für objektorientierte Datenbanken.

Bestandteile des Standards:

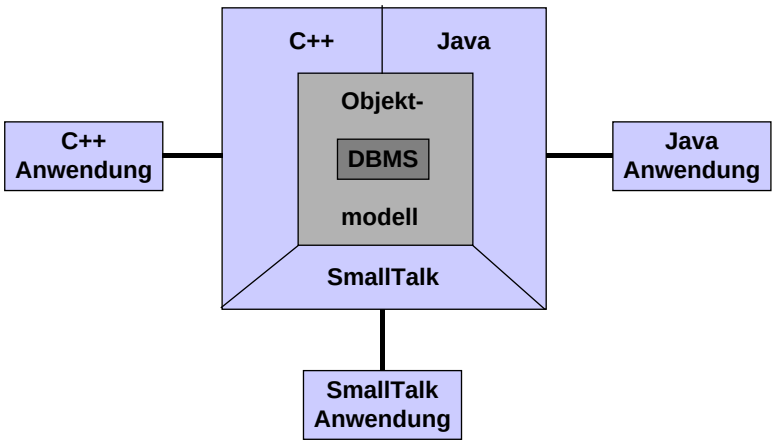
- **Objektmodell**
- **Object Definition Language (ODL)**
- **Object Query language (OQL)**
- **C++ Anbindung**
- **Smalltalk Anbindung**

Beteiligte:	Reviewer:
SunSoft	Hewlett-Packard
Object Design	Poet
Ontos	Itasca
O_2 Technology	Intellitic
Versant	DEC
Objectivity	Servio
	Texas Instruments

Hauptziel: Portabilitätsstandard (kein Interoperabilitätsstandard)

68

Integration des ODMG-Objektmodells



69

Abfragesprache OQL

OQL = Object Query Language
An SQL orientierte Abfragesprache, durch ODMG standardisiert

Grundstruktur:

SELECT	**Wie soll Resultatmenge (Objekte oder Werte) aussehen?**
FROM	**Welche Objektmenge soll inspiziert werden?**
WHERE	**Welche Eigenschaften müssen die Objekte erfüllen?**

Beispiele:

Zugriff auf Objektattribute

```
DEFINE Pos-Studenten AS
        SELECT Stud FROM Studenten Stud WHERE Stud->Sem < 12
```

Aufbau neuer Objektstrukturen

```
SELECT STRUCT (Name: Inf-Stud->Name, Fach1: "Inf", Fach2: Stud->Fach)
        FROM Informatik-Studenten Inf-Stud, Studenten Stud
        WHERE Inf-Stud->Name = Stud->Name AND Stud->Fach <> "Inf"
```

70

Objektrelationale Datenbanksysteme

Erweiterung der bestehenden relationalen Technologie um objektorientierte Konstrukte:

- **Beibehaltung des SQL-Standards**
- **systemweit eindeutige Objektidentifikationen**
- **erweiterbares Typsystem**
- **Vererbung**
- **Polymorphismus**

Beipiele kommerzieller objektrelationaler Produkte:

- **Version 2.0 DB2 (IBM)**
- **Informix/Illustra**
- **Omniscience**
- **Oracle 8**
- **OSMOS (Unisys)**
- **UniSQL/X**

71

"Intelligente" Datenbanksysteme

Die Entwicklung von Datenbankdiensten (und Programmiersprachen) zielt darauf ab, dem Benutzer Dienste auf möglichst hoher (problemnaher) Ebene anbieten zu können. Dazu werden die Aufgaben eines Datenbanksystems in verschiedene Richtungen erweitert:

- **Wahrung komplexer Konsistenzbedingungen**
- **Deduktive Fähigkeiten**
- **Induktive Fähigkeiten**

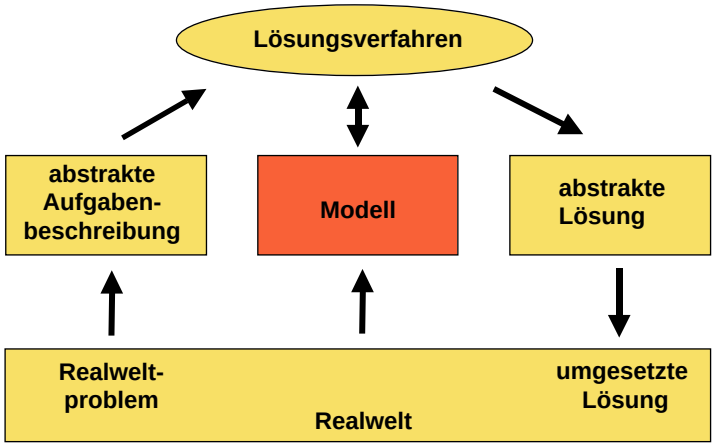
Im Rahmen von P3 behandeln wir

- **Regelbasierte Inferenzen**
- **Deduktive Datenbanken**
- **Beschreibungslogiken**

72

Denkpause: Wozu das Ganze?

Datenmodellierung soll Problemlösen unterstützen:



73

Wahrung komplexer Konsistenzbedingungen

- **Logische Zusammenhänge zwischen Entitätsmengen und Beziehungen werden durch ein ER-Modell meist nur unvollständig erfaßt und auch nicht konsequent ausgewertet.**
- **Integritätsbedingungen können in SQL zusätzlich (und optional) formuliert werden und folgen dann nicht aus dem logischen Datenmodell.**

Z.B. Attribute für Ferienhäuser in einer Tourismusdatenbank

"mückenfrei" = kein Gewässer in der Nähe

"mit Angelgelegenheit" = Angelsee in der Nähe

"Mückenfreies Ferienhaus mit Angelgelegenheit" ist inkonsistent

(Beispiel dankend von Volker Haarslev übernommen)

Z.B. Eintragung einer Geburt in eine Einwohnerdatenbank

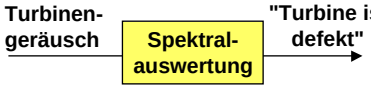
Eintragung durch DB-Verwalter:
"Anna Meier ist_Kind_von Ehepaar Meier"

Löschung zur Konsistenzwahrung:
"Ehepaar Meier ist_ein kinderloses Ehepaar"

74

Deduktive Fähigkeiten

Deduktion leitet neues Wissen aus bestehendem Wissen ab
- durch deduktive Prozeduren
- durch das Zusammenspiel von Regeln und Fakten
- mithilfe logischer Gesetzmäßigkeiten



Prozedur leitet Klassifikation "Turbine defekt" aus Daten ab

Regel: Wer Beamter ist und Familie hat, ist kreditwürdig
Fakten: Müller ist Beamter, Müller hat Familie
Anfrage: Ist Müller kreditwürdig?
Deduktion liefert: Ja, Müller ist kreditwürdig

Regelanwendung erlaubt die Beantwortung der Anfrage

OldParent = (AND (ATLEAST 1 has_child)(ALL has_child Adult)
Adult = (AND Person (ALL has_age AdultAge)
has_age(Amy, AdultAge)
has_child(John, Amy)

Aus den logischen Konzeptdefinitionen folgt (bei Abschluß der Datenbasis)
OldParent(John)

75

Induktive Fähigkeiten

Induktion erschließt regelhafte Zusammenhänge in Fakten

$$\frac{a \quad a \Rightarrow b}{b}$$ Deduktion

$$\frac{a \quad b}{a \Rightarrow b}$$ Induktion

$$\frac{b \quad a \Rightarrow b}{a}$$ Abduktion

Induktion beruht auf Verallgemeinerung und ist i.A. nicht zwingend.

(⟶ "Data Mining", Lernen)

Herst.-Datum	*Reklamation*
19.7.99	Rad fehlt
21.7.99	Lackschaden
26.7.99	Kühler leckt
30.7.99	bremst nicht
2.8.99	Lenkrad fehlt
9.8.99	hupt nicht

Datum	*Wochentag*
19.7.99	Montag
21.7.99	Mittwoch
26.7.99	Montag
30.7.99	Freitag
2.8.99	Montag
9.8.99	Montag

"Mängel entstehen bevorzugt am Montag"

76

Deduktion mit Regeln

Regeln im Format

WENN VORHANDEN <extensionale Daten> DANN FOLGERE <intensionale Daten>

WENN GEFRAGT <intensionale Daten> DANN SUCHE <extensionale Daten>

sind schon früh zum Ableiten impliziten Wissens eingesetzt worden.

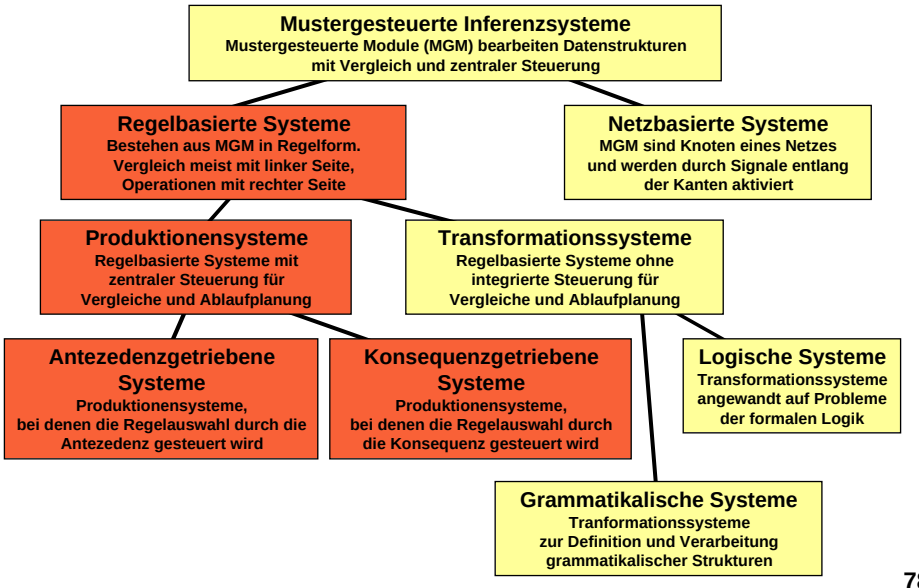
Beispiel: Experimentelle Programmiersprache PLANNER (Hewitt 72)

Daten	Konsequenztheoreme
(THASSERT (IN VOGEL KÄFIG)) (THASSERT (IN TISCH ZIMMER)) (THASSERT (IN STUHL ZIMMER)) (THASSERT (IN BLUME VASE)) (THASSERT (AUF KÄFIG TISCH)) (THASSERT (AUF VASE TISCH))	(THCONSE (X Y Z) (IN ?X ?Y) (THGOAL (IN !X ?Z) (THUSE NIL)) (THGOAL (IN !Z !Y)) (THCONSE (X Y Z) (IN ?X ?Y) (THGOAL (AUF !X ?Z)) (THGOAL (IN !Z !Y))

Anfrage	Das angefragte intensionale Datum (IN BLUME ZIMMER) wird mithilfe der Konsequenztheoreme abgeleitet.
(THGOAL (IN BLUME ZIMMER))	

77

Familie der mustergesteuerten Inferenzsysteme



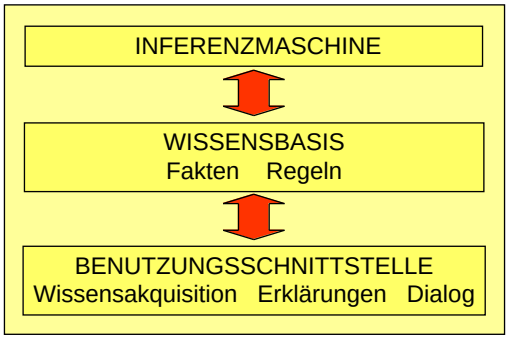
78

Regelbasierte Systeme

Regelsysteme zum Ableiten neuer Fakten wurden zwischen 1970 und ca. 1985 für *Expertensysteme* entwickelt.

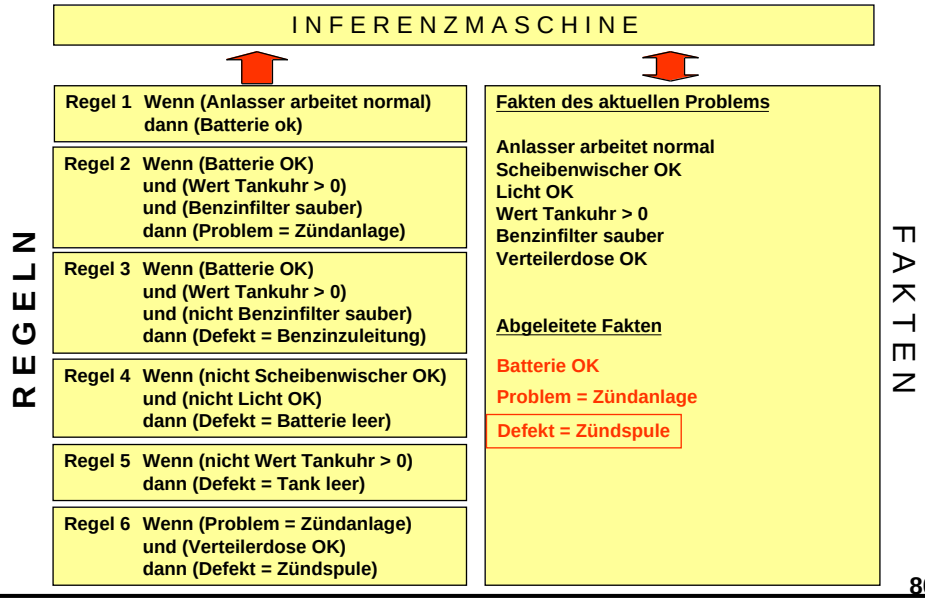
Die Idee bestand darin, menschliches Wissen in Form von Wenn-Dann-Regeln durch einen Rechner auswerten zu lassen.

Grobstruktur eines Expertensystems:



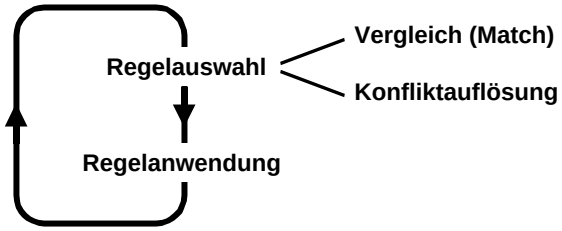
79

Regelbasierte Fehlerdiagnose an einem KFZ



80

Auswahl-Anwendungszyklus



Durch Vergleich des Antezedenzteils (bei Vorwärtsverkettung) oder Konsequenzteils (bei Rückwärtsverkettung) aller Regeln mit allen Datenobjekten werden die aktuell anwendbaren Regeln bestimmt.

Wenn mehr als eine Regel anwendbar ist, bestimmt ein Verfahren zur Konfliktauflösung, welche Regel angewendet wird.

81

Arbeitsprinzip einer Inferenzmaschine

Vorwärtsverkettung:

Wiederhole, bis Ziel abgeleitet ist:

- Bestimme Regeln, deren Bedingungen sich mit vorhandenen *Fakten* erfüllen lassen
- Wähle daraus eine Regel aus
- Wende Regel an, etabliere neue Fakten

Rückwärtsverkettung:

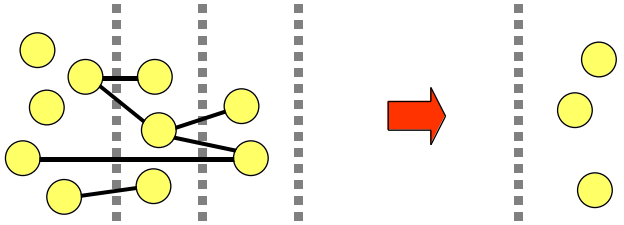
Wiederhole, bis Ziel abgeleitet ist:

- Bestimme Regeln, mit denen *Ziele* abgeleitet werden können
- Wähle daraus eine Regel aus
- Wende Regel an, etabliere neue Ziele für nicht erfüllte Bedingungen

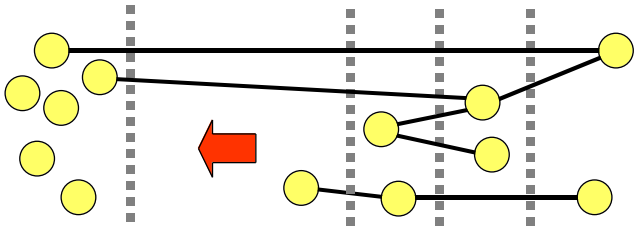
82

Verkettungsstrategien

Vorwärts-verkettung



Rückwärts-verkettung



83

Repräsentation von Regeln und Fakten

Für Regeln und Fakten sind im Forschungsgebiet Künstliche Intelligenz (KI), Teilgebiet Wissensrepräsentation, verschiedene Repräsentationsformen entwickelt worden.

Fakten:

Vorherrschend ist eine objektorientierte (besser: objektzentrierte) Sicht ähnlich wie in OO-Programmiersprachen.

Auto132	
Is-a:	**Auto**
Typ:	**BMW-302**
Baujahr:	**1995**
Farbe:	**silber-grün**
Vorgänger:	**Auto056**

Fortgeschrittene objektzentrierte Repräsentationen beruhen auf Beschreibungslogiken zur präzisen Definition von Bedeutungen

Auto ⊆ (and Fahrzeug
(atleast 3 hatTeil Rad)
(atleast 1 hatTeil Motor)
(atleast 1 hatTeil Karosserie)

Regeln: **<Bedingungsteil> ⟶ <Konsequenzteil>**

Der Bedingungsteil bezieht sich auf Fakten und und ist an ihre jeweilige Repräsentationsform angepaßt.

84

Bedeutung von Regeln

Regeln werden in regelbasierten Systemen mit unterschiedlichen Funktionen (Deutungen) verwendet und unterschiedlich beschrieben:

Prämisse	→	Konklusion	*logische Implikation*
Antezedenz	→	Konsequenz	*Folgerung bei gegebenen Voraussetzungen*
Evidenz	→	Hypothese	*Interpretation von Fakten*
Situation	→	Aktion	*situatives Verhalten*
WENN ...	→	DANN ...	*informelle Umschreibung*
linke Seite	→	rechte Seite	*völlig unverbindliche Sicht*

Werden Regeln zur Auswertung über einer Datenbasis formuliert, ergeben sich zwei grundsätzliche Interpretationsweisen:

"Wenn Prämisse in Datenbasis erfüllt, dann füge Konklusion der Datenbasis zu"

"Wenn Bedingung in Datenbasis erfüllt, dann führe Aktion aus"

85

Das Regelsystem OPS5

OPS5 ("Official Production System, Version 5")
- **zu Beginn der 80er Jahre am CMU entwickelt**
- **Implementierungssprache für erste XPS von DEC (XCON, XSEL u.a.)**

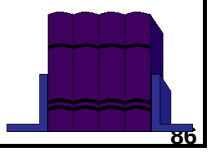
CLIPS
- **Reimplementierung von OPS5 in C für die NASA**
- **als Freeware erhältlich**

JESS
- **Reimplementierung von OPS5 in Java**
- **als Freeware erhältlich**
- **auf CD-ROM als Beigabe zum Buch** →

M. Watson:

Intelligent Java Applications

Morgan Kaufmann 1997



86

Regeln in OPS5

Syntax einer Regel in OPS5:

<rule>::=	**[P <rule-name> <antecedent> --> <consequent>]**
<antecedent>::=	**{<condition>}**
<condition> ::=	**<pattern> | - <pattern>**
<pattern> ::=	**[<object> {^<attribute> <value>}]**
<consequent> ::=	**{<action>}**
<action> ::=	**[MAKE <object> {^<attribute> <value>}] |** **[MODIFY <pattern-number> {^<attribute> <value>}]** **[REMOVE <pattern-number>] |** **[WRITE {<value>}]**

Beispiel: Finde Personen mit gleichem Wohnort

[P Finde-Familie [Person^Name <x1> ^Wohnort <y>]
[Person^Name <x2> ^Wohnort <y>] --> ...]

Variable

- **gleiche Variablennamen erfordern gleiche Werte**
- **Suche nach möglichen Instantiierungen aller Variablen kann sehr aufwendig werden**

87

Regelauswahl

Wenn im Auswahl-Anwendungszyklus mehrere Regeln feuerbereit sind, kann eine Auswahlstrategie verwendet werden (zur "Konfliktauflösung"):

• alte Fakten (Ziele) bevorzugen	*Breitensuche*
• neue Fakten (Ziele) bevorzugen	*Tiefensuche*
• spezielle Regel vor allgemeiner	*spezieller = mehr Bedingungen*
• Priorisierung von Regeln	*z.B. durch Speicherfolge (PROLOG)*
• Entscheidung durch Metaregeln	*Regeln für die Regelauswahl*

Für ein regelbasiertes System ist grundsätzlich keine im Einzelnen kontrollierbare Ablaufsteuerung vorgesehen. Der Entwickler soll von einzelnen Schritten des Inferenzprozesses abstrahieren.

=> Paradigma der datengetriebenen Verarbeitung

88

Implementierung eines Regelsystems

Naive Implementierung des Auswahl-Anwendungszyklus führt im allgemeinen zu unakzeptablem Laufzeitverhalten (Überprüfung des gesamten Datenbestandes auf Regelanwendbarkeit).

Effizienzsteigerung durch:

1. Einschränkung der zu überprüfenden Daten

Ergebnisse des letzten Mustervergleichs speichern und nur veränderte Datenelemente daraufhin prüfen, ob neue Regelanwendungen möglich oder alte nicht mehr möglich sind.

2. Einschränkung der zu überprüfenden Prämissen

Haben mehrere Regeln Prämissen gemeinsam, brauchen diese nur einmal evaluiert zu werden.

Der Rete-Algorithmus realisiert diese Effizienzsteigerungen durch Verwalten feuerbereiter Daten und Prämissen in einer Netzstruktur.

(Details s. Expertensystem-Literatur)

89

Regelbasierte Konfigurierung von Rechneranlagen mit XCON

Vertriebsmitarbeiter nimmt Kundenwünsche auf, XCON prüft auf Konsistenz, ergänzt und parametriert Komponenten.

- **Inbetriebnahme 1982 mit 1000 Regeln**
- **Software-Tod nach 1988 mit mehr als 10000 Regeln**

Paraphrase einer XCON-Regel:

```
ASSIGN-POWER-SUPPLY-1
IF:   THE MOST CURRENT ACTIVE CONTEXT IS ASSIGNING A POWER SUPPLY
      AND AN SBI MODULE OF ANY TYPE HAS BEEN PUT IN A CABINET
      AND THE POSITION IT OCCUPIES IN THE CABINET IS KNOWN
      AND THERE IS SPACE IN THE CABINET FOR A POWER SUPPLY
      AND THERE IS NO AVAILABLE POWER SUPPLY
      AND THE VOLTAGE AND FREQUENCY OF THE COMPONENTS IS KNOWN
THEN: FIND A POWER SUPPLY OF THAT VOLTAGE AND FREQUENCY
      AND ADD IT TO THE ORDER
```

90

"Intelligente" Datenbanken durch Regelsysteme?

Vorteile:

- **Regeln passen zu objektorientierter Datensicht**
- **Regeln bieten intuitive Formulierungsmöglichkeit**
- **Regeln können vielseitig eingesetzt werden**
- **keine Ablaufsteuerung erforderlich**
- **inkrementell zu entwickeln**

Nachteile:

- **Regeln können inkonsistent sein**
- **Regeln können inkonsistente Daten erzeugen**
- **keine klare Semantik**
- **große Regelmengen (> 1000) sind unübersichtlich**

91

Deduktionen mit DATALOG

DATALOG ermöglicht deduktive Erweiterung einer relationen Datenbasis.

- **Repräsentation von Fakten und Regeln auf der Basis eines eingeschränkten Prädikatenkalküls**
- **ähnlich wie PROLOG**
- **präzise Definition von *extensionaler* und *intensionaler* DB**

Extensionale Datenbasis (EDB):

"Faktenbasis", explizit gespeicherte Tupel von Relationen (= Prädikate)

Deduktionskomponente

DATALOG-Programm, besteht im wesentlichen aus Ableitungsregeln

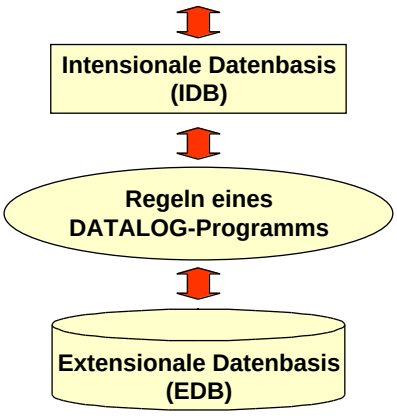
Intensionale Datenbasis (IDB):

Virtuelle Relationen, werden mit Deduktionskomponente abgeleitet

92

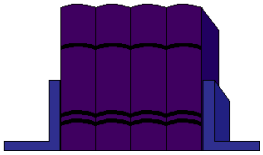
Grobstruktur einer deduktiven Datenbank

Benutzersicht auf IDB + EDB



Literatur:

A. Kemper, A. Eickler: Datenbanksysteme, 3. Auflage, Oldenbourg, 1999



93

Notation in DATALOG

Bestandteile eines Prädikatenkalküls ohne Funktionssymbole:

Variable:	**Zeichenketten, die mit Großbuchstaben beginnen**
Konstante:	**Zeichenketten, die mit Kleinbuchstaben beginnen oder Zahlen**
Prädikate:	**Zeichenketten, die mit Kleinbuchstaben beginnen**
Term:	**Konstante oder Variable**
Atom:	**n-stelliges Prädikat angewandt auf n Terme** z.B. p(X, a, Y)
Grundatom:	**alle Argumente sind Konstante**
Literal:	**Atom oder negiertes Atom**
Klausel:	**Liste von Literalen** z.B. { ¬p(X,a), q(Y,a) } **dasselbe im Prädikatenkalkül:** ∀X ∀Y (¬ p(X,a) ∨ q(Y,a))
Hornklausel:	**enthält höchstens 1 positives Literal**

94

Regeln in DATALOG

DATALOG-Regel:

- **Hornklausel mit *genau einem positiven Literal* und *mindestens einem negativen* Literal**
- **Argumente des positiven Literals sind *Variable*, die mindestens einmal in den negativen Literalen vorkommen müssen**

Beispiel:

R: { ¬parent(X), father(X), ¬male(X) }

Umformung im Prädikatenkalkül:

$\forall X\ (\neg \text{parent}(X) \vee \text{father}(X) \vee \neg \text{male}(X))$

$\forall X\ (\neg (\text{parent}(X) \wedge \text{male}(X)) \vee \text{father}(X))$

$\forall X\ ((\text{parent}(X) \wedge \text{male}(X)) \Rightarrow \text{father}(X))$ **Regel als Implikation**

Vergleiche mit PROLOG:

{ p(...), $\neg p_1(...)$, ... , $\neg p_k(...)$ } in PROLOG: p(...) :- $p_1(...)$, ... , $p_k(...)$

95

Fakten, Anfragen, Programme

Fakten:

positive Literale z.B. parent(otto, anna), brother(emil, anna)

entsprechen Tupeln von Relationen

Anfrage (Ziel, goal):

negative Klausel z.B. ¬father(otto, X)

vergleiche mit PROLOG: ?- father(otto, X)

DATALOG-Anfragen werden i.A. aus Benutzeranfragen an die DB generiert

DATALOG-Programm:

Fakten und DATALOG-Regeln mit der Eigenschaft:

- **Fakten sind Teil der EDB**
- **Prädikate der Regelköpfe sind Prädikate der IDB**

96

Beispiel eines DATALOG-Programms

```
anc(X, Y) :- par(X, Y).
anc(X, Y) :- anc(X, Z), par(Z, Y).
anc(adam, X) :- person(X).
person(X) :- lives(X, Y).

par(otto, anna).
par(berta, otto).
par(berta, emil).
lives(otto, hamburg).
lives(anna, kiel).

"Wer sind die Vorfahren von Anna?"
?- anc(X, anna).

Antwort:
{anc(otto, anna), anc(adam, anna), anc(berta, anna) }
```

anc und person sind Prädikate (Relationen), die nur in der IDB auftreten

par und lives sind Prädikate (Relationen) der EDB

97

Beantworten von Anfragen

Antwortgenerierung mit einem logischen Beweisverfahren.

<u>Beweisidee</u>:

Behaupte Unerfüllbarkeit der Anfrage und finde Gegenbeispiele.

z.B. Anfrage ?- anc(X, anna) wird interpretiert als X (¬anc(X, anna))

Resolution (Robinson 1965) ist ein vollständiges und korrektes Verfahren zum Ableiten von Widersprüchen in Klauselmengen.

<u>Prinzip</u>: Verbinde zwei Klauseln, die dasselbe Literal, aber mit verschiedenem Vorzeichen enthalten

Beispiel:

{ ¬anc(X, anna) }	***Klausel 1***
{ anc(X, Y), ¬par(X, Y) }	***Klausel 2***
{ ¬par(X, anna) }	***Resolvente***

Klauseln müssen durch geeignete Substitutionen unifiziert werden

Dahinter steckt die allgemeine Schlußregel:
((A => B) ∧ (B => C)) => (A => C)

98

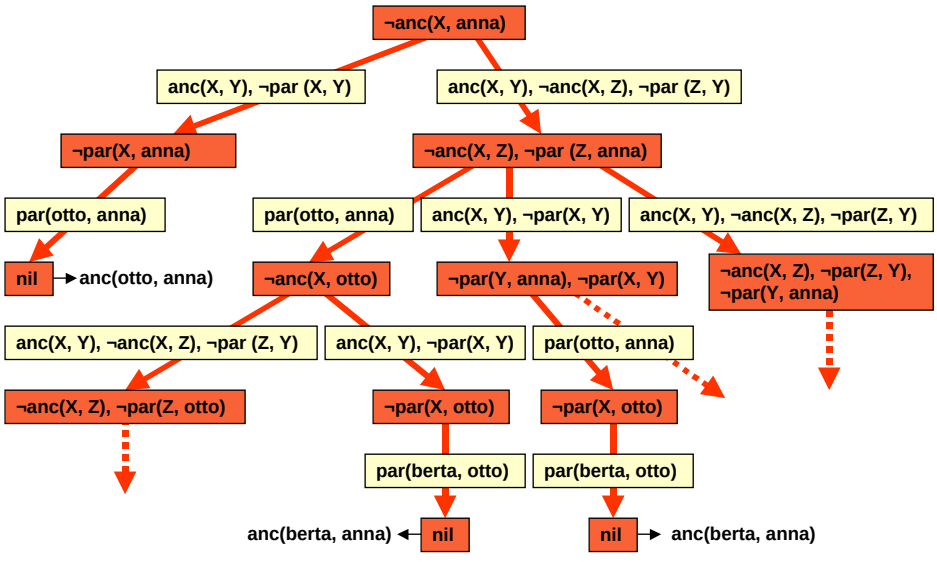
Rückwärtsverkettung zur Beweissuche

1. Gegeben eine Klauselmenge K und eine Anfrageklausel A∉K. Resolviere A mit allen passenden Klauseln aus K. Resolvente sind Nachfolger von A.
2. Ist ein Nachfolger leer (Widerspruch), so ist er ein Blatt des Suchbaums. Führe alle Substitutionen von der Wurzel bis zum Blatt an den Variablen der Anfrageklausel durch. Füge die so entstandene Grundklausel der Ergebnismenge zu.
3. Wiederhole Schritt 1 mit allen nichtleeren Nachfolgern bis zu einer maximalen Tiefe T.

Eine effektive Tiefenbeschränkung T läßt sich bei rekursiven Regeln ohne Einbuße der Vollständigkeit und Korrektheit des Verfahrens nicht immer angeben (s. Beispiel). Hier ist T ~ Zahl der Tupel in EDB.

99

Beispiel für Resolutionsbeweis



100

DATALOG-Implementierungen

Für effektive Berechnungen der IDB erforderlich:

- **Vermeiden "unsicherer" Regeln**
  Regelköpfe müssen instantiierbar sein
- **Berechnung eines auf die Anfrage zugeschnittenen Teils der IDB**
  kann durch Rückwärtsverkettung bei Regelanwendungen erreicht werden
- **Regelanwendung in der besten Reihenfolge**
  Sortierung der Regeln und Tupel entsprechend Abhängigkeitsgraph
- **Besondere Maßnahmen bei Rekursivität**
  Vermeiden mehrfacher Ableitungen (hier nicht im Detail behandelt)

101

Abhängigkeitsgraph

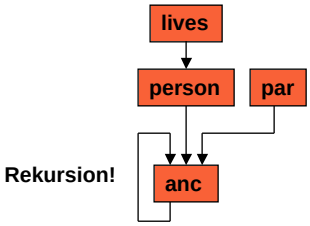
Ein Abhängigkeitsgraph macht explizit, welche Relation aus welcher Relation berechnet werden kann.

Er besteht aus den Prädikaten des DATALOG-Programms als *Knoten* und den durch Regeln möglichen Berechnungen als *Kanten*:

$p(...) :- q_1(...), ... ,q_n(...)$ ergibt n Kanten $q_i(...) \longrightarrow p$, i = 1 ... n

Ein DATALOG-Programm ist genau dann *rekursiv*, wenn der Abhängigkeitsgraph *Zyklen* hat.

<u>Beispiel</u>:




```
anc(X, Y) :- par(X, Y).
anc(X, Y) :- anc(X, Z), par(Z, Y).
anc(adam, X) :- person(X).
person(X) :- lives(X, Y).
```

102

Vergleich PROLOG - DATALOG

PROLOG	DATALOG
Tiefensuche	Breitensuche
tupelorientiert	mengenorientiert
sensitiv auf Reihenfolge	nicht sensitiv auf Reihenfolge
Spezialprädikate	keine Spezialprädikate
Funktionssymbole	keine Funktionssymbole

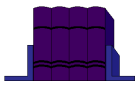
- **PROLOG ist nicht nur Deduktionsmaschine sondern auch Programmiersprache.**
- **DATALOG bietet zuverlässige deduktive Datenbankerweiterung.**

103

Datenmodellierung mit Beschreibungslogiken

Eine Beschreibungslogik (Description Logic) ist ein Formalismus zur Wissensrepräsentation:

- **objektorientiert**
- **formale Semantik**
- **Teilmenge des Prädikatenkalküls**
- **Konzepte und Fakten**



Einführende Darstellung von KL-ONE in P. Struß (Hrsg.): Wissensrepräsentation, Oldenbourg, 1991, S. 103 - 122

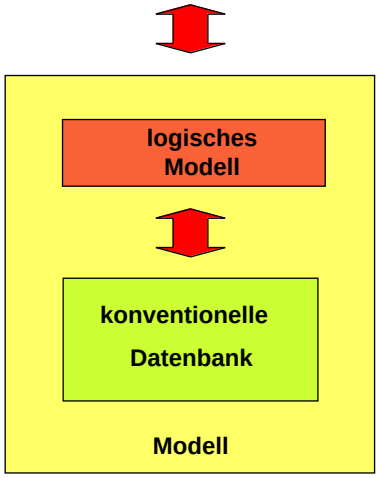
- **mächtige Inferenzdienste**
  - Konsistenztest — ***Ist eine Beschreibung erfüllbar?***
  - Subsumptionstest — ***Impliziert ein Konzept eine gegebene Beschreibung?***
  - Klassifikation — ***Einordnen einer gegebenen Beschreibung in eine Konzepthierarchie***
  - Abstraktion — *Verallgemeinern,* ***Gemeinsamkeiten mehrerer Beschreibungen***

104

"Modell" als intelligente Datenbank

Logisches Modell ermöglicht

- **zusätzliche Dienste**
- **Garantien für Korrektheit und Vollständigkeit von Diensten**
- **Wiederverwendbarkeit von Problemlösungskomponenten**



105

Autohandel mithilfe von Beschreibungslogik

Anfrage:

"Gesucht Auto mit Ledersitzen, mehr als 90 PS, Einspritzmotor, 25000 - 30000 DM"

Sitzbezug:	**Leder**
Leistung:	**> 90 PS**
Motortyp:	**Einspritzer**
Preis:	**45000.-**

Welche Klasse kommt in Frage?

Klasse:	**Daimler-Sport**
...	
Sitzbezug:	**{Leder, Stoff}**
Leistung:	**150 PS**
Motortyp:	**Einspritzer**
Preis:	**53000.-**
Hersteller:	**Daimler**
Farbe:	**{silber, weiß, rot}**
...	

Mögliche Subsumer der Anfragebeschreibung stellen gewünschte Antwort dar.

Klassifikation erfordert *hinreichende* Bedingungen dafür, daß Anfragebeschreibung Klassenbeschreibung subsumiert.

106

Wichtige Beschreibungslogiken

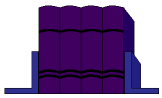
Beschreibungslogiken sind aktuelles Forschungsthema:

- **Ausdruckskraft der Konzeptsprache**
- **Entscheidbarkeit und Traktabilität der Inferenzdienste**
- **Einbeziehen konkreter Domänen (z.B. Zahlen, Raum und Zeit)**

Es gibt mehrere experimentelle und kommerzielle Entwicklungen von Beschreibungslogiken, darunter:

- KL-ONE Erster Entwurf einer Beschreibungslogik 1985
- CLASSIC Kommerzielle Implementierung von AT&T
- LOOM Experimentelles System der USC
- RACE Experimentelles System am FB Informatik der Uni HH

Die folgende Darstellung verwendet RACE als Beispiel.



V. Haarslev, R. Möller, A.-Y. Turhan: RACE User´s Guide and Reference Manual Version 1.1, FBI-Mitteilung 269

Primitive und definierte Konzepte

Konzepte einer Beschreibungslogik beschreiben Objektmengen (vergl. Entitätsmengen, Klassen) durch Konzeptausdrücke, mit denen Eigenschaften der Objekte festgelegt werden.

Grundbausteine sind primitive oder definierte Konzepte.

Primitive Konzepte: **Konzept => Eigenschaften**

Eigenschaften sind *notwendige* Bedingungen für Zugehörigkeit eines Objektes zum Konzept

Definierte Konzepte: **Konzept <=> Eigenschaften**

Eigenschaften sind *notwendige und hinreichende* Bedingungen für Zugehörigkeit eines Objektes zum Konzept

Primitives Konzept "person":
(implies person (and human (some has-gender (or female male))))

Definiertes Konzept "parent":
(equivalent parent (and person (some has-child person)))

Konzeptbeschreibungen

Konzeptterme

C = Konzeptterm, CN = Konzeptname, RN = Rollenname, n = natürliche Zahl

C ⟶	**CN**	**|**	***Konzeptname***
	top	**|**	***allgemeinstes Konzept***
	bottom	**|**	***speziellstes (inkonsistentes) Konzept***
	(not C)	**|**	***Negation***
	(and C_1 ... C_n)	**|**	***Konjunktion***
	(or C_1 ... C_n)	**|**	***Disjunktion***
	(some RN C)	**|**	***existentielle Restriktion von Rollenfüllern***
	(all RN C)	**|**	***Werterestriktion von Rollenfüllern***
	(at-least n RN)	**|**	***Kardinalitätsrestriktion für Rollenfüller***
	(at-most n RN)	**|**	***Kardinalitätsrestriktion für Rollenfüller***
	(exactly n RN)		***Kardinalitätsrestriktion für Rollenfüller***

Konzeptaxiome

(define-primitive-concept CN C)	***Konzeptname subsumiert Konzeptausdruck***
(define-concept CN C)	***Konzeptname und Konzeptausdruck sind gleich***
(implies C_1 C_2)	***allgemeiner Konzepteinschluß***
(equivalent C_1 C_2)	***Gleichheit zweier Konzepte***
(disjoint C_1 C_2)	***Disjunktheit (keine gemeinsamen Instanzen)***

109

Rollen

Rollen beschreiben zweistellige Beziehungen zwischen Objekten.

Attribute (features)	***funktionale Rollen mit höchstens einem Rollenfüller pro Individuum***
Transitive Rollen	***Rollen sind transitiv geschlossen , d.h. für transitive Rolle r gilt $r(i_1, i_2) \wedge r(i_2, i_3) \Rightarrow r(i_1, i_3)$***
Rollenhierarchien	***Angabe von Ober- und Unterrollen***
Domänenrestriktion	***simuliert in RACE mit*** **(implies (some RN *top*) C)**
Bereichsrestriktion	***simuliert in RACE mit*** **(implies *top* (all RN C))**

In RACE gibt es nur primitive Rollen, d.h. Rollenzugehörigkeit kann i.A. nicht erschlossen werden.

Ausnahmen: Transitive Rollen, Schlüsse in Rollenhierarchien

110

Beispiel einer T-Box

```
(signature      :atomic-concepts   (person human female male woman man parent
                                    mother father grandmother aunt uncle sister brother)
                :roles             ((has-child :parent has-descendant)
                                    (has-descendant :transitive t)
                                    (has-sibling)
                                    (has-sister :parent has-sibling)
                                    (has-brother :parent has-sibling)
                                    (has-gender :feature t)))
```

Signatur der T-Box

```
(implies *top* (all has-child person))
(implies (some has-child *top*) parent)
(implies (some has-sibling *top*) (or brother sister))
(implies *top* (all has-sibling (or sister brother)))
(implies *top* (all has-sister (some has-gender female)))
(implies *top* (all has-brother (some has-gender male)))
```

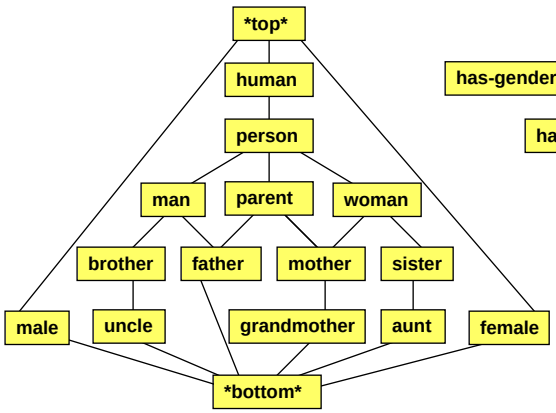
Domänen- und Wertebereichs-einschränkungen für Rollen

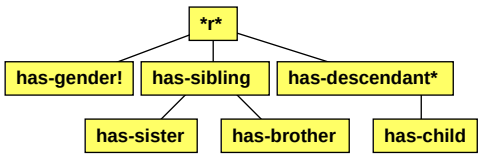
```
(implies person (and human (some has-gender (or female male))))
(disjoint female male)
(implies woman (and person (some has-gender female)))
(implies man (and person (some has-gender male)))
(equivalent parent (and person (some has-child person)))
(equivalent mother (and woman parent))
(equivalent father (and man parent))
(equivalent grandmother (and mother (some has-child (some has-child person))))
(equivalent aunt (and woman (some has-sibling parent)))
(equivalent uncle (and man (some has-sibling parent)))
(equivalent brother (and man (some has-sibling person)))
(equivalent sister (and woman (some has-sibling person)))
```

Konzepte

111

Konzept- und Rollenhierarchie





r	universelle Rolle
!	Attribut (feature)
*	transitive Rolle

112

T-Box-Inferenzen

Kernstück aller T-Box-Inferenzen ist der Test eines Konzeptausdrucks auf Inkonsistenz:

C $\stackrel{?}{\models}$ *bottom*

Beispiel: (and (at-least 1 has-child) (at-most 0 has-child)) $\models$ *bottom*

Damit können andere Inferenzdienste aufgebaut werden, z.B.

- **Subsumption**
  (implies C1 C2) <=> (and C1 (not C2)) $\models$ *bottom*
- **Klassifikation eines Konzeptausdrucks**
  Suche in der Konzepthierarchie top-down nach den speziellsten Konzepten, die den Konzeptausdruck subsumieren

T-Box-Inferenzen werden durch einen *Beweiser* realisiert.
(Ableiten eines logischen Ausdrucks aus Axiomen heißt Beweisen.)

113

Formale Semantik von Konzeptausdrücken

D	**Menge aller möglichen Objekte einer Domäne**
$\mathbf{E[C] \subseteq D}$	**Extension eines Konzeptausdrucks C (repräsentiert Bedeutung von C)**
$\mathbf{E[RN] \subseteq D \times D}$	**Extension einer Rolle RN (repräsentiert Bedeutung von RN)**

Formale Semantik von Konzeptoperatoren wird durch Mengenoperationen definiert:

E[*top*] = D

E[*bottom*] = { }

E[(and C_1 ... C_n)] = $E[C_1] \cap ... \cap E[C_n]$

E[(or C_1 ... C_n)] = $E[C_1] \cup ... \cup E[C_n]$

E[(all RN C)] = $\{x \mid \forall (x, y) \in E[RN] \Rightarrow y \in E[C]\}$

E[(some RN C)] = $\{x \mid \exists (x, y) \in E[RN] \wedge y \in E[C]\}$

114

Beschreibung einer A-Box

T-Box = terminologisches Wissen (Konzepte und Rollen)

A-Box = assertionales Wissen (Fakten)

Eine A-Box enthält:

- **Konzeptassertionen** **(instance IN C)**
  Individuum IN ist Instanz eines Konzeptausdrucks C
- **Rollenassertionen** **(related IN_1 IN_2 RN)**
  Individuum IN_1 steht zu Individuum IN_2 in Beziehung RN

- **Eine A-Box bezieht sich immer auf eine T-Box.**
- **In einer A-Box werden eindeutige Bezeichner verlangt (unique name assumption).**
- **Die Fakten einer A-Box werden als unvollständig angenommen (OWA).**

  OWA = Open World Assumption
  (neue Fakten können dazukommen, daher weniger Inferenzen möglich)

  CWA = Closed World Assumption
  (Es können keine neue Fakten dazukommen)

A-Box-Inferenzen

A-Box-Inferenzen = Ableiten von Aussagen über A-Box-Individuen

Typische Anfragemöglichkeiten:

- **Konsistenztest** ***Ist die A-Box konsistent?***
- **Retrieval** ***Welche Instanzen befriedigen einen Konzeptausdruck?***
- **Klassifikation** ***Welche (speziellsten) atomaren Konzepte beschreiben eine Instanz?***

A-Box-Konsistenztest ist anders (und in der Regel komplizierter) als T-Box-Konsistenztest.

A-Box konsistent <=> es gibt "Modell" für T-Box und A-Box
(s. Modellbegriff in der Logik, in F2 behandelt)

Realisierung aller A-Box-Inferenzen auf der Basis des A-Box-Konsistenztests.

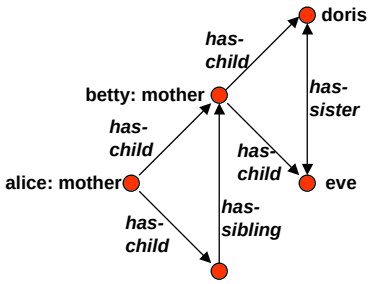
Befragen einer A-Box

Inhalt der A-Box

(instance alice mother)
(related alice betty has-child)
(related alice charles has-child)

(instance betty mother)
(related betty doris has-child)
(related betty eve has-child)

(instance charles brother)
(related charles betty has-sibling)
(instance charles (at-most 1 has-sibling))
(related doris eve has-sister)
(related eve doris has-sister)



Fragen und Antworten

(individual-instance? doris woman)
T

Ist doris *Instanz vom Konzept* woman*?*

(individual-types eve)
((sister) (woman) (person) (human) (*top*))

Von welchen atomaren Konzepten ist eve *Instanz?*

(individual-fillers alice has-descendant)
(doris eve charles betty)

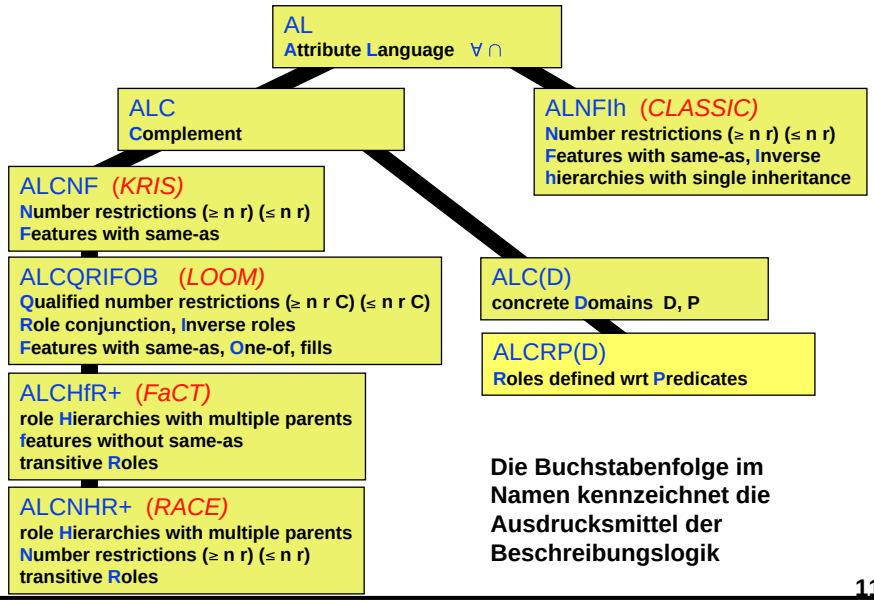
Was sind die descendants von eve*?*

(concept-instances sister)
(doris betty eve)

Welche Instanzen hat das Konzept sister*?*

117

Familie der Beschreibungslogiken



Die Buchstabenfolge im Namen kennzeichnet die Ausdrucksmittel der Beschreibungslogik

118

Abstraktionen mit Beschreibungslogiken

Abstraktion = Fortlassen von Eigenschaften, Verallgemeinern

Beispiele:

- **Übergeordnetes Konzept in Konzepthierarchie**
  Mann, Frau → Person
- **verallgemeinerter Konzeptausdruck**
  (and (some has-occupation professor) (at-least 3 has-child))

  (and (some has-occupation beamter) (at-least 1 has-child))
- **subsumierender Konzeptausdruck für mehrere Instanzen**
  1. **bestimme zugehörige Konzepte für Instanzen (Objektklassifikation)**
  2. **berechne kleinstes gemeinsames Oberkonzept (Least Common Subsumer LCS)**
     - **für RACE: triviale Lösung durch (or C_1 ... C_n)**
     - **für Beschreibungslogiken ohne ODER: spezieller Abstraktionsoperator LCS**

Anwendungsbeispiel: Auswahl von TV-Sendungen

ARD	ZDF	RTL	SAT.1
20.15	20.15	20.15	20.00
Fußball-WM	China heute	Galactica	Dragonheart
21.45	21.15	21.35	21.00
Sissi	Wetten, daß...	Braveheart	Stirb langsam 2
22.30	22.00	22.45	22.15
Tagesthemen	Heute	Sexshow	Rolling Stones
23.00	22.30	23.30	23.00
The Rock	Terminator 2	Speed	Alien

Benutzer wählt Beispiele aus

Braveheart
Stirb langsam 2
Terminator 2

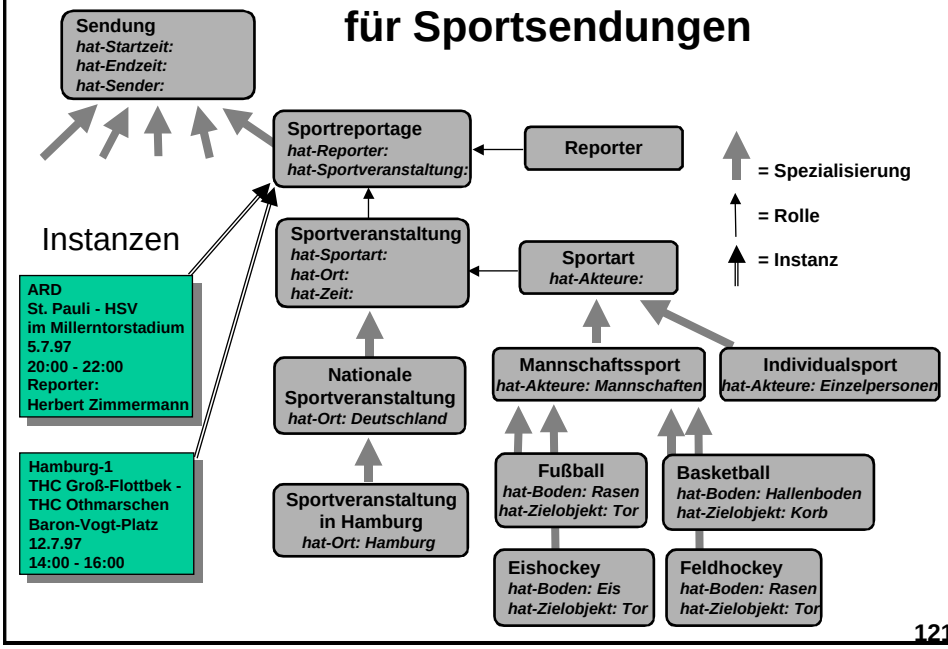
System bestimmt inhaltliche Gemein-samkeiten

- Action/Horror
- Kino-Highlights
- bekannte Schauspieler
- Filme neueren Datums

beispiel-basierte Programm-auswahl

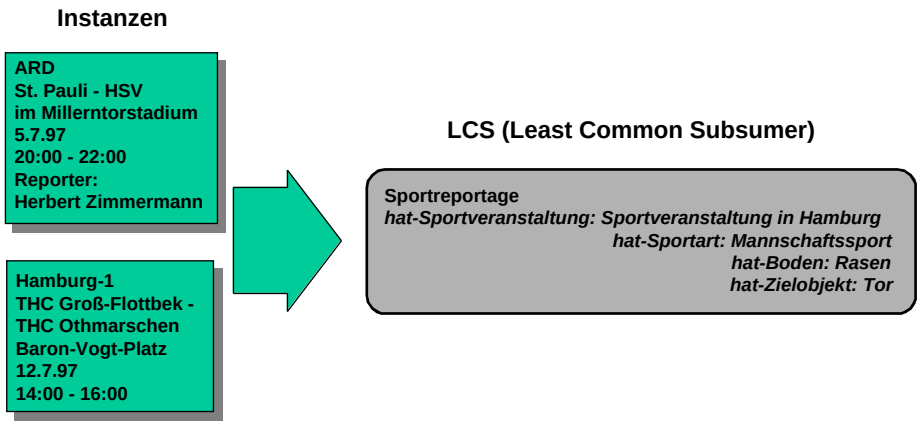
ARD	N3	RTL	PRO 7
20.15	20.15	20.15	20.00
Schatzinsel	Eiskunstlauf	Goldfinger	Psycho II
21.45	21.00	21.30	21.00
Lindenstraße	Sterbehilfe	Dallas	Deep Impact
22.30	22.00	22.15	22.15
Tagesthemen	Extra 3	Titanic	Killerwale
23.00	22.30	23.30	23.00
Armageddon	Achterbahn	Robocop	Arabella

Beispiel: Begriffssystem für Sportsendungen



121

Kleinster gemeinsamer Oberbegriff von 2 Instanzen



122

Problematische Anfragen an einen relationalen Datenbestand

Studenten, deren Semesterzahl größer als die Regelstudienzeit ihres Studienfaches ist.

MatrNr	*Sem*	*Fach*
124711	6	Informatik
132457	4	BWL
132457	23	Germanistik
142346	29	Informatik

Fach	*RegelSem*
Informatik	9
BWL	8
Germanistik	10
Physik	10

(implies matrnr (and zahl (some has-sem zahl) (some has-fach fach)))
(implies fach (and (or informatik bwl germanistik physik) (some has-regelsem zahl))))

(related 124711 6 has-sem)
(related 124711 informatik has-fach)
...
(related informatik 9 has-regelsem)
...
(concept-instances (and matrnr (somep has-sem has-fach.has-regelsem >)))

Feature-Kette (in RACE nicht vorhanden)

Prädikat über konkreter Domäne (in RACE nicht vorhanden)

123

Logische Datenmodelle mit Beschreibungslogiken

Beschreibungslogiken bieten:

- differenzierte logische Beschreibung von Datenobjekten
- mächtige Inferenzdienste
- Korrektheits- und Vollständigkeitsgarantien

Beschreibungslogiken unterliegen Beschränkungen:

- zweistellige Beziehungen (im Vergleich zu n-stelligen Relationen)
- beschränkte Ausdruckskraft bei Anfragen
- aufwendige Berechnungsverfahren (je nach Ausdrucksstärke)
- erst experimenteller Umgang mit konkreten Domänen (z.B. Zahlen)

124