5. Das relationale Daten(bank)modell

5.1 Vorbemerkungen, Einordnung, Historie

5.2 Begriffe und Eigenschaften des relationalen Modells; Abbildung E/R $\rightarrow$ relational

5.3 Sprachen für das relationale Modell: Relationenalgebra und Relationenkalkül

5.4 Die Sprache für das relationale Modell: SQL (Structured Query Language)

5.5 Anwendungsprogrammierung mit eingebettetem SQL (embedded SQL)

5.1 Vorbemerkungen, Einordnung, Historie

Entstehungsgeschichte / Ausgangspunkt

- Datenbanksysteme nach dem hierarchischen Modell und Netzwerk-Modell „**weit entfernt**“ vom **Endbenutzer** (gelegentlichen Benutzer, „casual user“):
  - Datenbank nur aus **Anwendungsprogramm** heraus ansprechbar → Programmierkenntnisse erforderlich
  - **Datenbanksprache** (DL/1, CODASYL-DDL und -DML) **schwer zu erlernen und fehlerfrei** anzuwenden
  - Nur kleiner Kreis von **Spezialisten** somit in der Lage, wirklich souverän mit hierarchischen bzw. Netzwerk-Datenbanksystemen umzugehen
  - **„Entwicklungsstau“**

- Komplizierter Umgang mit Datenbanksystem führt außerdem dazu, dass **Fehler** gemacht werden, die die Datenbank **inkonsistent** werden lassen: DBVS kann keine Inkonsistenz feststellen, es gilt aber „Zustand der modellierten Miniwelt ≠ Zustand der Datenbank“ ⇒ **fatal vor allem bei kritischen Anwendungen** (Prozessteuerung in der Industrie, aber auch Anwendungen im Finanzbereich und viele andere Beispiele möglich)

- **Entwicklungsziel für relationales Datenmodell / relationale Datenbanksysteme** (IBM Forschungslabor in San Jose CA, Ende der 1960er Jahre, **E.F. Codd** u.a.):
  - **Einfaches Datenmodell** mit mathematisch fundierter Grundlage (die der Endbenutzer nicht unbedingt kennen muss)
  - **Einfache Datenbanksprache** (auch für Ad-hoc-Anfragen) mit mathematisch fundierter Grundlage (auch hier kann die „Mathematik“ vor dem Endbenutzer verborgen werden) ⇒ **Verhalten** von Datenbanksprachanweisungen (Semantik) wohldefiniert auf math. Grundlage ⇒ Benutzer beschreibt nur noch, **was** er haben möchte (lesen, einfügen, ändern, löschen), das **wie** überlässt er dem DBS
    M.a.W.: **Deskriptive** Anfragen statt **Navigation**!!
    Was statt **wie**!!

- { **Integritäts** / **Konsistenz** } **überwachung** möglichst weitgehend durch das DBVS, nicht durch den Benutzer ⇒ Informationen über die **Semantik der Daten** dem DBVS offenlegen, so dass automatische Überprüfungen möglich werden (**DBVS** berücksichtigt bei Operationen die **Daten-inhalte** und verlässt sich nicht darauf, dass der Benutzer „die Zeiger richtig setzt“ (Bsp.: Set-Ausprägungs-Bestimmung bei CODASYL als abschreckendes Beispiel))

- **Ergebnis**
  - **Datenmodell** und **Datenbanksprache** leichter zu erlernen (auch durch „Endbenutzer“ (mit Einschränkung)) und somit **näher am Benutzer** ⇒ breiterer Benutzerkreis ⇒ Reduzierung des Entwicklungsstaus
  - Für **Datenbankzugriff** nicht eigens Anwendungsprogramme zu erstellen, sondern auch **ad-hoc** möglich
  - Inkonsistente Datenbanken unwahrscheinlicher
  - Ineffizienter Datenbankzugriff unwahrscheinlicher ⇒ wenn das DBVS für das **wie** des Zugriffs zu den Daten verantwortlich ist, **kann** es hierfür eine **optimale Ausführungsstrategie** wählen (Zugriffsplan, festgelegt durch DBVS-Komponente „optimizer“)

- **Entwicklungsweg relationaler Datenbanksysteme**
  - Erste **Prototypen** ab Mitte der 1970er:
    - System R (IBM San Jose) → später Prod. DB2 & SQL/DS
      Codd et al. (Codd nur „Grundlagen“, nicht an konkreten Systementwicklungen beteiligt)
    - Ingres (Univ. of California, Berkeley) → später Prod. Ingres sowie Prototyp / Prod. Postgres*
      PostgreSQL (Open Source)
      Stonebraker
  - Erste **Produkte** ab Ende der 1970er:
    - SQL/DS, DB2*
    - Ingres
    - Oracle*
    - Informix*
    - Sybase*, ADABAS D*, MS SQL Server, MySQL u.v.a.m.
  - Stärkere Verbreitung in der Praxis ab Mitte der 1980er und 1990er

* in Jena vorhanden (URZ, Lehrstuhl)

5.2 Begriffe und Eigenschaften des relationalen Modells

Ziele bei Definition des Modells

- Möglichst **wenige**, **einfache**, **mathematisch fundierte Konzepte** und **Begriffe**
- Beschreibung **ausschließlich logischer Aspekte** (**konzeptuelles Schema**); physische Aspekte (internes Schema) davon sauber getrennt (nicht Gegenstand des Modells)

↪ Datendarstellung in Form von **Relationen**,
Datenabfrage und –manipulation auf diesen Relationen

Gegeben seien { Wertemengen / **Wertebereiche** } (Domänen, „domains“) $D_1, D_2, ..., D_n$

Relation $\mathbf{R} \subseteq D_1 \times D_2 \times ... \times D_n \quad (n \geq 1)$

Eine Relation (**Menge!**) **R** ist also eine Teilmenge des kartesischen Produkts der Domänen $D_1, D_2, ..., D_n$

n = **Stelligkeit** (**Grad**) der Relation

Ein Element $\mathbf{r} = (d_1, d_2, ..., d_n)$ mit $d_i \epsilon D_i$

wird als **Tupel** von R bezeichnet, d_i als die i-te **Komponente** des Tupels

Beispiel

D_1 = {rot, blau, grün}

D_2 = {0,1}

$D_1 \times D_2$ = {(rot,0), (rot,1), (blau,0), (blau,1), (grün,0), (grün,1)}

R_1 = {(rot,0), (grün,1)}

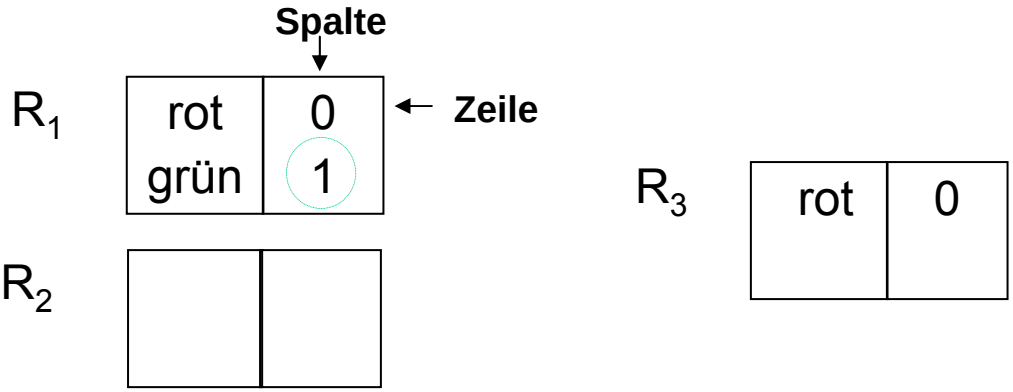
R_2 = { }

R_3 = {(rot,0)}

↳ sind mögliche (2-stellige) Relationen

Relationen und Tabellen

Relationen können als **Tabellen** dargestellt („aufgefasst“) werden



Tupel = **Tabellenzeile** („rows“)

Komponente des Tupels = **Wert in Tabellenspalte** („columns“)

Relation vs. Relationsschema

- Bisher wurden Relationen (**Mengen** von Tupeln) betrachtet
- Aus **Datenbanksicht** ist aber auch das zugehörige **Schema** von Interesse, das einen **Relationstyp** beschreibt $\Rightarrow$ Abstraktion von konkreter Relation zum Relationstyp

- **Relationsschema** besteht aus:
  - **Schemaname**
  - Menge von **Domänen**(namen) $D_1, D_2, ..., D_n$
  - Menge von **Attributnamen** $A_1, A_2, ..., A_n$ (**= Benennung der Tabellenspalten** bei tabellarischer Darstellung), wobei für die Attribut**werte** a_i (Tabellen**einträge**) $a_i \epsilon D_i$ gelten muss

  [- Zusätzlichen **Integritätsbedingungen** (s.u.)]

- Bsp.1: **Domänen**: D_1, D_2 wie oben: D_1 = {rot, blau, grün}
  D_2 = {0,1}

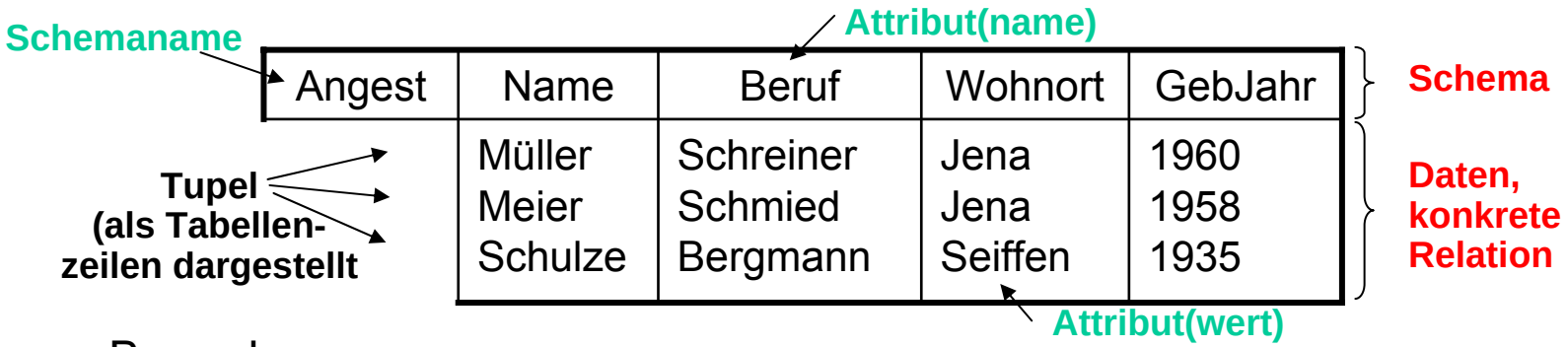
  Attributnamen: Farbe, Wert

  Schemaname: FarbTabelle

  Integritätsbedingung: Farbe rot darf nicht mit Wert 1 zusammen in einem Tupel vorkommen

Kurzschreibweise auch : FarbTabelle (Farbe, Wert) wenn die
Attribut**menge**
zugehörigen Domänen/Integritätsbedingungen separat beschrieben sind

- Bsp.2: Relationsschema Angest (Name, Beruf, Wohnort, GebJahr) (+ Domänen/Integritätsbedingungen) als Darstellung von **Schema + Daten**



- Bemerkungen:
  - **Menge** von Relationsschemata (mit sich unterscheidenden Schemanamen) + zusätzliche Integritätsbedingungen (relationsschemaübergreifend) ergibt **Datenbankschema**
  - Begrifflich wird zwischen Relationsschema und Relation nicht immer strikt unterschieden; wenn von **Relation** die Rede ist, wird meist darunter auch das Schema subsumiert (d.h. Daten + Schema oder auch nur Schema)

- Dürfen in einer Relation aufgrund ihrer Schemadefinition nur **atomare Werte** auftreten, d.h. u.a. keine Wiederholungsgruppen, so sagen wir, dass die Relation in **1. Normalform** ist

Das relationale Modell, wie von Codd definiert und in DBVS-Produkten realisiert, verlangt „im Prinzip“ 1. Normalform!!!

↳ mengenwertige bzw. strukturierte Attribute (im E/R-Verständnis) seien hier also verboten „vorerst“ → sog. objektrelationale DBMS lassen sie zu (mehr oder weniger)

Unsere Attribute typischerweise (1. Normalform!)

Integer, Real, Decimal, CHAR, STRING

→ atomare Attribute

Schlüssel

- Relationen sind **Mengen**, es dürfen also in einer Relation keine identischen Tupel auftauchen

Müller	Schreiner	Jena	1960
Müller	Schreiner	Jena	1960

unzulässig

„reine Lehre“

- Relationenmodell verlangt für jede Relation Vorhandensein eines **Schlüssels**, d.h. einer **identifizierenden Attributkombination**, und dieser Schlüssel „muss“ **minimal** sein, d.h. bei Weglassen eines Attributs aus der Kombination geht Schlüsseleigenschaft verloren
- Bsp.: Relation Angest (Name, Beruf, Wohnort, GebJahr)
  Was ist Schlüssel??
  $\Rightarrow$ i.d.R. Angest (**PNR**, Name, Beruf, Wohnort, GebJahr) erforderlich
- Falls für eine Relation mehrere Schlüssel(kandidaten) existieren, muss einer ausgewählt werden als **Primärschlüssel**
- Falls eine Attributkombination einer Relation R1 in einer Relation R2 (Primär-)Schlüsseleigenschaft besitzt, können wir sie in R1 als **Fremdschlüssel** bezeichnen mit Bezug auf den (Primär-)Schlüssel in R2

- Bsp.: Angest (PNR, Name, Beruf, Wohnort, GebJahr, ANR)

Ein Fremdschlüssel ist (i.d.R.) **kein** Schlüssel

Abteilung (ANR, AOrt, Mgr, Budget)

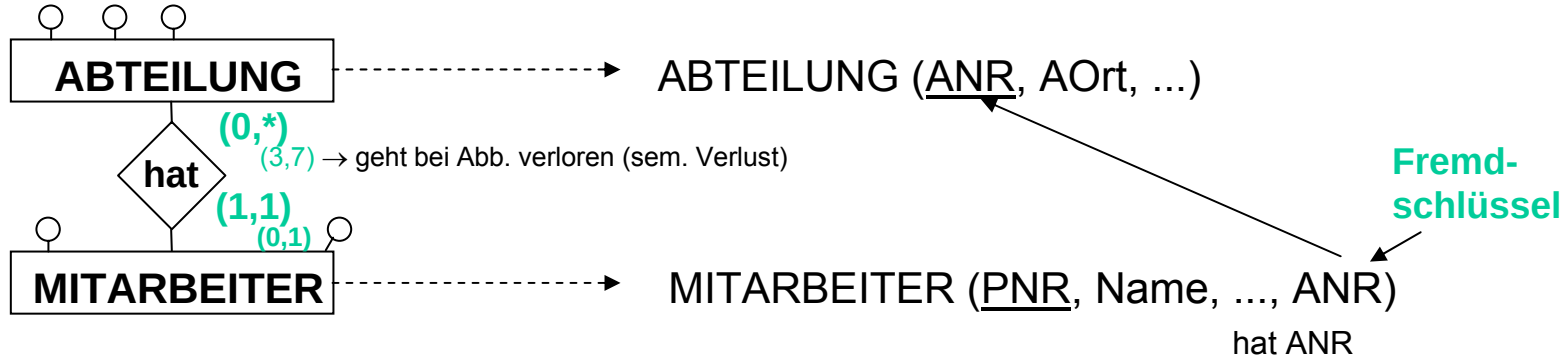
⇒ ANR kann als Fremdschlüssel in Angest bezüglich des Primärschlüssels ANR in Abteilung **vereinbart** werden
→ **per DDL-Anweisung**

- Bedeutung **Konsequenz der Fremdschlüsseldefinition**/Forderung **(sog. referentielle Integrität, „referential integrity“)** (erläutert am obigen Beispiel): Für ANR in Angest dürfen nur Werte auftreten, die auch als ANR-Wert in Abteilung vorkommen (entspricht: Waisenkinder sind verboten!)
  - ↪ jedenfalls von der „reinen Lehre“ her, in realen relationalen Datenbanksystemen – und auch in der (SQL-)Sprachnorm – gibt es Möglichkeiten, sie zuzulassen
- Eine Relation **muss** einen Primärschlüssel besitzen und **darf** beliebig viele Fremdschlüssel besitzen
- **Namensidentität** zwischen der Attributkombination (Fremdschlüssel) in R1 und der ((Primär-)Schlüssel) in R2 ist nicht vorgeschrieben, lediglich „Kompatibilität“ der Wertebereich [möglichst Identität anzuraten]

Abbildung E/R-Modell → relational

1. nicht rekursive 1:n-Beziehungen

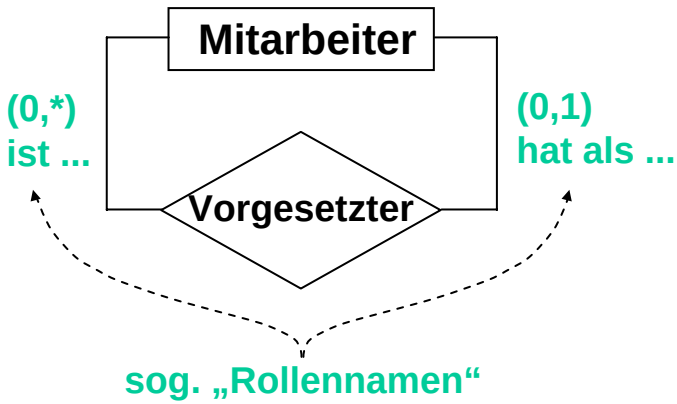
E/R-Diagramm **Relational**



- Überführung Entity-Typ → Relation(sschema)
- ANR in Mitarbeiter ist **Fremdschlüssel** mit Bezug auf den Primärschlüssel von Abteilung ⇒ **repräsentiert Beziehungstyp** „hat“ aus dem E/R-Diagramm (Überführung des Beziehungstyps in eine eigenständige Relation somit nicht erforderlich)
- (Atomare) Attribute eines Entity-Typs werden unmittelbar in Attribute der jeweils entsprechenden Relation übernommen; **Vorsicht bei Schlüsselüberführung** (Minimalitätsforderung des relationalen Modells in bezug auf Schlüssel)
- „Beliebige Kardinalitäten“ (z.B. (3,7)) nicht direkt im relationalen Modell darstellbar

2. rekursive 1:n-Beziehung

E/R-Diagramm



Weiter IB:
In der MgrPNR-Spalte von Mitarbeiter darf nur **einmal** NULL stehen (beim Big Boss)

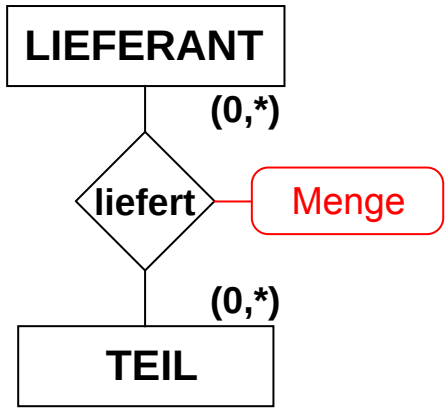
Relational

Mitarbeiter (PNR, Name, ..., **MgrPNR**)

- MgrPNR ist Fremdschlüssel in Mitarbeiter mit Bezug auf den Primärschlüssel der gleichen Relation (ist erlaubt!!)
- Wie stellen wir als Tupel den „obersten Boss“ dar? Alternativen:
  - MgrPNR und PNR besitzen gleichen Wert (nicht empfehlenswert!) **Trick**
  - MgrPNR ist „undefiniert“ (spezieller, dem DBVS bekannter Nullwert) **NULL**

3. nichtrekursive n:m-Beziehung(en)

E/R-Diagramm

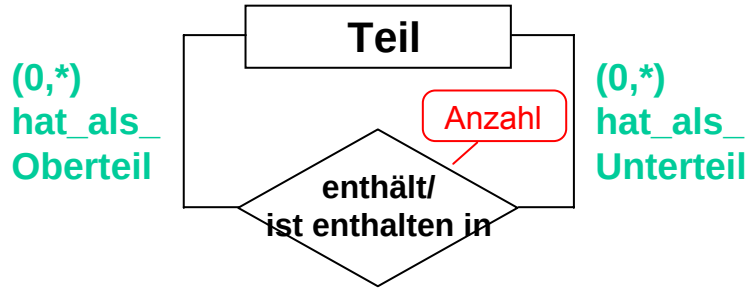


Relational

Lieferant (<u>LieferNr</u>, Name, Ort)
Teil (<u>TeileNr</u>, Bezeichnung, ...)
Liefert (<u>TeileNr</u>, <u>LieferNr</u>, Menge)

- Beziehungstyp wird in **eigenständige Relation** überführt
- **(Primär-)Schlüssel** dieser Relation: Attribut**kombination** der Schlüssel der Relationen Lieferant und Teil
- TeileNr in Liefert ist Fremdschlüssel mit Bezug auf Primärschlüssel von Teil
- LieferNr in Liefert ist Fremdschlüssel mit Bezug auf Primärschlüssel von Lieferant

4. rekursive n:m-Beziehung(en)



„Stückliste“

vgl. Gozinto-Graphen auf Folie 156

Teil möge sowohl für ein Einzelteil als auch für eine Baugruppe stehen

Relational

Teil (TeileNr, Bezeichnung, ...)

Struktur (OberteilNr, UnterteilNr, Anzahl)

- Beziehungstyp wird in **eigenständige Relation** überführt
- **(Primär-)Schlüssel** setzt sich aus Attributen OberteilNr und UnterteilNr zusammen
- Ein Struktur-Tupel pro Kante im Gozinto-Graphen
- OberteilNr und UnterteilNr sind jeweils einzeln Fremdschlüssel in Struktur mit Bezug auf Primärschlüssel von Teil

Beispiele (relationale Tabellen) zu den Abbildungen E/R-Modell → relational

1. nicht rekursive 1:n-Beziehungen (Folie 180)

Abteilung	ANR	AOrt	. . .
	3815	Jena	. . .
	3952	Weimar	. . .
	4717	Erfurt	. . .
	. . .	. . .	. . .

referentielle Integrität:
Werte dienen als Verweise

(kein „Pointer“ wie etwa in Netzwerk-Datenbanken)

Mitarbeiter	PNR	Name	. . .	ANR
	2837	Meier	. . .	3815
	1113	Meyer	. . .	3952
	1548	Maier	. . .	4717
	. . .	. . .	. . .	. . .

Verweise als
log. Konstrukt

- ANR in Mitarbeiter ist **Fremdschlüssel** mit Bezug auf den Primärschlüssel (ANR) von Abteilung
  ⇒ als **Attributwerte für ANR in Mitarbeiter** dürfen **nur** Werte auftreten, die auch als Attributwerte für ANR in Abteilung existieren
- **Falls** Mitarbeiter erlaubt sein sollen, die **keiner** Abteilung angehören (**Entwurfsentscheidung!**), darf für ANR in Mitarbeiter vereinbart werden, dass **Nullwert („undefiniert“)** gestattet ist

- Beim **Nullwert („undefiniert“)** handelt es sich **nicht** um den „normalen“ Wert 0, sondern um einen speziell reservierten Wert mit der (dem DBVS bekannten) **Semantik „undefinierter Wert“**

Mitarbeiter	PNR	Name	. . .	ANR
	2837	Meier	. . .	3952
	1113	Meyer	. . .	**undef**
	1548	Maier	. . .	4717
	2964	**undef**	. . .	3815

undef = NULL

Annahmen hier: Für die Attribute **ANR** und **Name** in Mitarbeiter sei die **Zulässigkeit von Nullwerten** vereinbart worden
= Administratorentscheidung

Für den Primärschlüssel sind Nullwerte stets verboten!!

d.h. auch ´in Teilen´ eines mehrattributigen Primärschlüssels sind sie verboten
↳ NULL-Werte würden gegen identifizierende Eigenschaft des Schlüssels verstoßen ↯

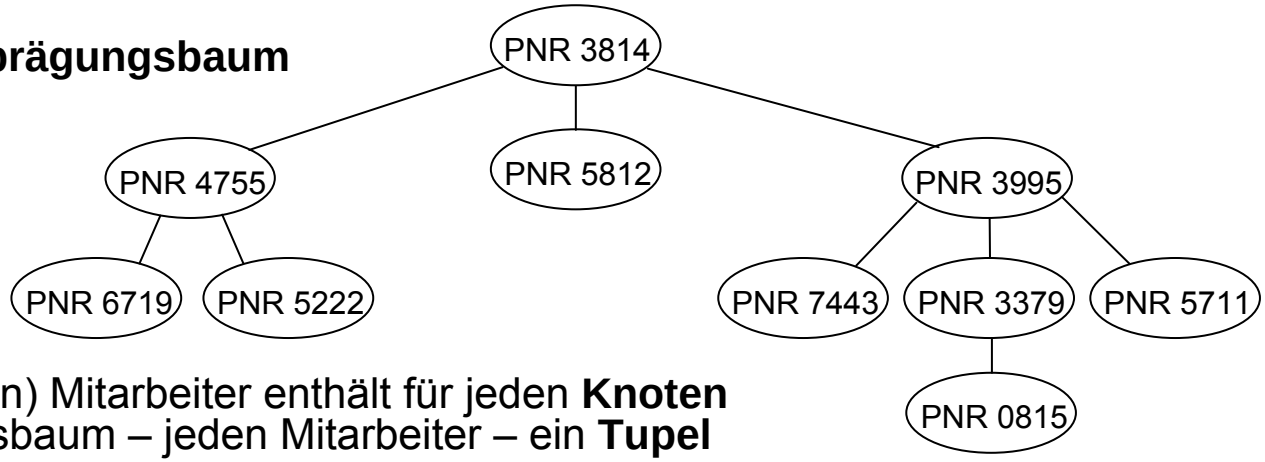
2. rekursive 1:n-Beziehung (Folie 181)

Mitarbeiter	PNR	Name	. . .	MgrPNR
	3814	Schmidt	. . .	**undef**
	4755	Dagobert	. . .	3814
	5812	Graf	. . .	3814
	6719	Schneider	. . .	4755
	. . .	. . .	. . .	. . .

Reihenfolge der Tabellen**zeilen** unerheblich (Menge!)

[Spaltenreihenfolge ebenfalls unerheblich]

steht für einen **Ausprägungsbaum** der Gestalt:



- Tabelle (Relation) Mitarbeiter enthält für jeden **Knoten** im Ausprägungsbaum – jeden Mitarbeiter – ein **Tupel**
- **Kanten** im Ausprägungsbaum werden durch die **Primärschlüssel-Fremdschlüssel-Beziehungen** in der Tabelle dargestellt
- **Nullwert („undefiniert“)** für Attribut MgrPNR erlaubt!
- Welche **zusätzlichen Integritätsbedingungen** gibt es?
  - Zyklenfreiheit
  - **Genau ein undef** in Spalte MgrPNR gestattet (wirklich?)

3. nichtrekursive n:m-Beziehungen (Folie 182)

Lieferant	LieferNr	Name	Ort	. . .
	3612	Jenoptik	Jena	. . .
	2525	Zeiss	Jena	. . .
	5888	Rodenstock	Stuttgart	. . .
	. . .	. . .	. . .	. . .

Teil	TeileNr	Bezeichnung	. . .
	374	Glasauge	. . .
	812	Linse	. . .
	111	Okular	. . .
	. . .	. . .	. . .

Liefert	TeileNr	LieferNr	Menge
	374	3612	7
	374	5888	5
	111	2525	8
	812	2525	30
	812	3612	30
	. . .	. . .	. . .

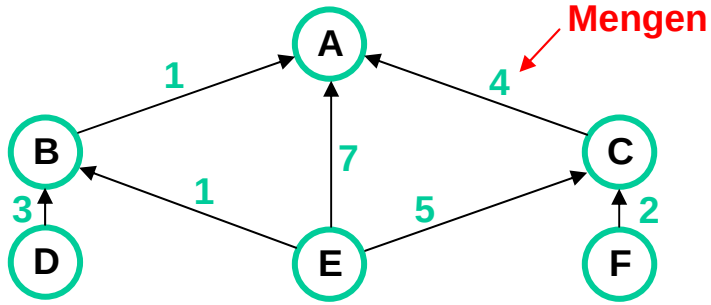
- In der Spalte **TeileNr** von **Liefert** dürfen nur Teilenummern vorkommen, die auch in **Teil** vorhanden sind
- In der Spalte **LieferNr** von **Liefert** dürfen nur Lieferantennummern vorkommen, die auch in **Liefert** vorhanden sind

Konsequenz der 2 Primärschlüssel-Fremdschlüssel-Beziehungen

- **Nullwerte**problematik hier durch Definition ausgeschlossen (warum?)

4. rekursive n:m-Beziehungen (Folie 183)

DAG



Gozinto-Graph
(Stücklistendarstellung)

[vgl. Darstellung und Diskussion in Zshg. mit Netzwerk-Datenmodell auf Folien 156]

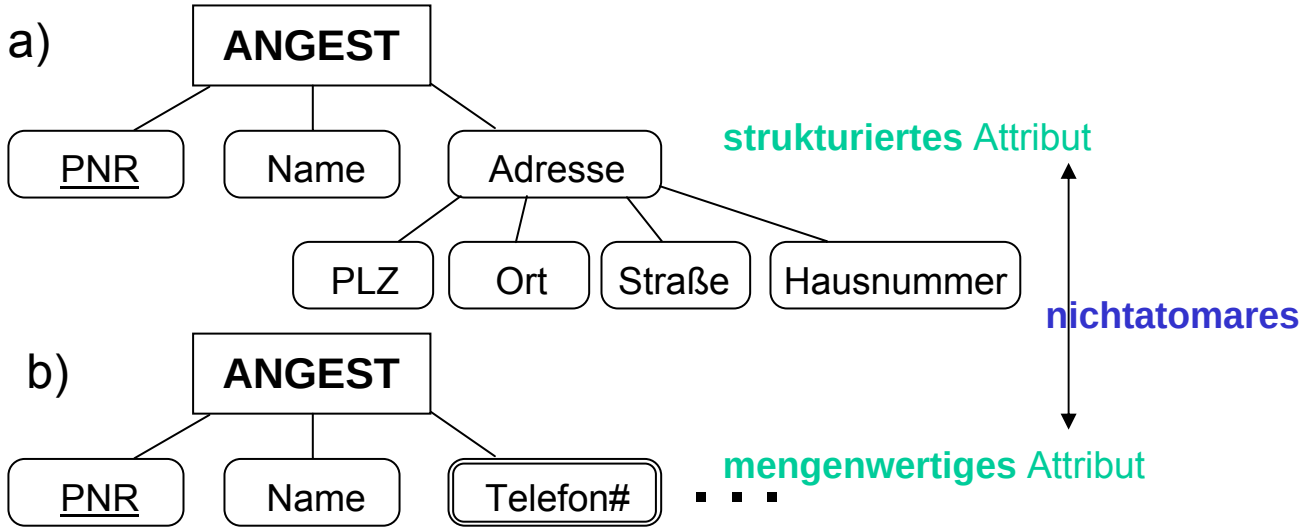
Teil	TeileNr	Bezeichnung	. . .
	A	Getriebe	. . .
	B	Gehäuse	. . .
	C	Welle	. . .
	D	Schraube	. . .
	E	Schraube	. . .
	F	Kugellager	. . .

Liefert	Ober-teilNr	Unter-teilNr	Anzahl
	A	B	1
	A	C	4
	A	E	7
	B	D	3
	B	E	1
	C	E	5
	C	F	2

- In den Spalten **OberteilNr** und **UnterteilNr** von **Struktur** dürfen nur Teilenummern vorkommen, die auch in **Teil** vorhanden sind
  ⇒ **Konsequenz der 2 Primärschlüssel-Fremdschlüssel-Beziehungen**
- **Nullwerte**problematik hier durch Definition ausgeschlossen (wie bei 3. oben)
- **Zusätzliche Integritätsbedingung**? → Zyklenfreiheit wegen "A"

Umgang mit nichtatomaren Attributen im E/R-Modell bei der Abbildung auf Relationenmodell (wegen 1 NF)

Beispiele: a)



beides nicht direkt („1:1“) auf relationales Modell 1. Normalform abbildbar

Lösungen:

b) **Zwei Tabellen**

Fremdschlüssel

Angest	PNR	Name	. . .
	3814	Schmidt	. . .
	4755	Dagobert	. . .
	5812	Graf	. . .
	. . .	. . .	. . .

Telefon	PNR	TelNr
	3814	31350
	3814	31051
	4755	31189
	4755	24753
	4755	24752

- PNR in Telefon ist **Fremdschlüssel** mit Bezug auf den Primärschlüssel (PNR) von Angest

- **Nullwert**problematik hier durch Definitionen ausgeschlossen
- **Zusätzliche Integritätsbedingung** denkbar:
  kein „Telefonnummern-Sharing“, d.h. zwei Angestellte besitzen nie die gleiche Telefonnummer
  ⇒ TelNr in Telefon zum Schlüssel(kandidaten) machen, garantiert Werteeindeutigkeit! (Primärschlüssel kann PNR mit TelNr bleiben)?
  → **nein, „eigentlich nicht“** d.h. von der reinen Lehre verboten
- Anderer Modellierungsvorschlag: **EINE Tabelle**

Angest´	PNR	TelNr	Name	. . .
	3814	31350	Schmidt	. . .
	3814	31051	Schmidt	. . .
	4755	31189	Dagobert	. . .
	4755	24753	Dagobert	. . .
	4755	24752	Dagobert	. . .
	. . .	. . .	. . .	. . .
	4317	**NULL**	**Hugo**	↯


nein!

Probleme hierbei:

- Mitarbeiter ohne Telefon können in Angest´ nicht aufgenommen werden!!
- Redundanz in Tabellenspalte Name!! Lösung: höheren Normalf. (2ff.) Änderungsaufwand, Integritätsgefährdung

- Nachteil der Verwendung **zweier** Tabellen (Angest, Telefon)
  ⇒ Eine Datenbankanfrage, die z.B. **Name und TelNr** sehen möchte, muss auf **beide** Tabellen zugreifen

a) **Eine Tabelle**

... durch „Herausnehmen“ der Modellierungsebene Adresse und „Hochziehen“ der darunterliegenden atomaren Attribute

Angest	PNR	Name	PLZ	Ort	Straße	Hausnr.	. . .
	3814	Schmidt	73428	Stammheim	Freiheit	47	. . .
	4755	Dagobert	13121	Berlin	Moabit	17	. . .
	5812	Graf	68131	Mannheim	Am Knast	7	. . .
	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Nachteil:

sem. Verlust

- **Zusammengehörigkeit** von PLZ-Ort-Straße-Hausnummer (nämlich als Adresse) nicht mehr erkennbar
- Adresse in Datenbankanfrage nicht mehr **einfach als ganzes** ansprechbar („Gib mir Adresse von Dagobert“)

Alternative:

Angest´	PNR	Name	. . .
	3814	Schmidt	. . .
	4755	Dagobert	. . .
	5812	Graf	. . .
	. . .	. . .	. . .

Adressen	PNR	PLZ	. . .
	3814	73428	. . .
	4755	13121	. . .
	5812	68131	. . .
	. . .	. . .	. . .

- Eine Anfrage gegen die Adresstabelle liefert Adresse erweitert um die PNR („Gib mir alles aus Tabelle Adressen für PNR=4755“)

5.3 Sprachen für das relationale Modell: Relationenalgebra und Relationenkalkül

Vorbemerkungen:

- Die Sprachen für das relationale Modell (Relationenalgebra, -kalkül und auch SQL (5.4)) sind **nicht navigierend und satzorientiert**, sondern **mengenorientiert und deskriptiv**
- Es werden **Mengen von Tupeln** gelesen, eingefügt, geändert, gelöscht mit **einer DML-Anweisung** $\Rightarrow$ größere **Sprachmächtigkeit** als bei HDM/NDM, potentiell auch **Performance-Gewinn** möglich (nur **eine** (mächtige) Sprachanweisung zu verarbeiten vom DBVS vs. **viele** (einfache) Anweisungen beim HDM/NDM)
  Optimizer des DBMS ist „gefordert“ – und wie!!!
- Spezifikation von **WAS** statt des **WIE**
- **Basisoperationen des relationalen Modells**
  - **Selektion**: Auswahl von Tupeln (mit gewissen Eigenschaften) aus einer Relation $\rightarrow$ **Zeilen**auswahl aus Tabelle
  - **Projektion**: Streichen von Spalten aus einer Tabelle im Zuge der Anfrageausführung $\leftrightarrow$ Auswählen von Spalten, die übrigbleiben sollen $\rightarrow$ **Spalten**auswahl aus Tabelle

- **Verbund (Join):** Verknüpfung von Tabellen aufgrund von Attributwert-Beziehungen ⇒ Tupel aus i.d.R. verschiedenen Tabellen werden zu einem Tupel „**konkateniert**“
- **Mengenvereinigung, -differenz, -durchschnitt**: Operationen der Mengenlehre ausgeführt auf verschiedenen Relationen **gleicher Struktur** (wg. Homogenität)

Beispiele für diese Basisoperationen

Beispiel-tabellen:

Angest	ANGNR	Name	Wohnort	Beruf	AbtNr
	112	Müller	Erfurt	Ingenieur	3
	205	Winter	Zwickau	Programmierer	3
	117	Rüllich	Weimar	Hundezüchter	5
	198	Schumann	Jena	Kaufmann	4
	. . .	. . .	. . .	. . .	. . .

Projekt	PRONR	PName	PBeschr	PLeiter
	27	Pkw2000	xyz...	205
	16	Wankel99	xyz...	117
	84	Trabbxx	xyz...	117
	. . .	. . .	. . .	. . .

Mitarbeit	PRONR	ANGNR	Prozent
	27	112	100
	27	198	70
	16	198	30
	. . .	. . .	. . .

Beispielanfragen:

- „**Finde die Wohnorte aller Angestellten, die Programmierer sind**“
  Lösung:

  a) **Selektion** der Tupel der Angest-Relation mit
  Beruf = "Programmierer"

  gefolgt von

  b) **Projektion** auf Spalte Wohnort

  Hinweis: Operationsergebnis muss wieder eine **Relation** (Menge!) sein ⇒ Projektion muss evtl. Duplikate (mehrere Programmierer mit identischem Wohnort) automatisch eliminieren

  ⇒ **Abgeschlossenheit** der relationalen Operationen

- „**Finde die Namen aller Angestellten, die am Projekt 27 mitarbeiten**“

  1. Lösungsvariante

  a) Verbinde die Relationen Mitarbeit und Angest zu einer neuen temporären Relation (Join), indem Angest-Tupel und Mitarbeit-Tupel **mit gleicher** Angestellten**nummer** „konkateniert“ werden

gefolgt von

b) Auf diesem Zwischenergebnis: **Selektion** der Tupel mit PRONR = 27 und **Projektion** auf Spalte Name (Duplikate zu eliminieren?)

2. Lösungsvariante

a) **Selektion** der Tupel der Mitarbeit-Relation mit PRONR = 27 und **Projektion** auf Spalte ANGNR (Duplikate zu eliminieren?)

gefolgt von

b) Nimm dieses Zwischenergebnis und verbinde es mit der Relation Angest (**Join**), indem Tupel **mit gleicher Angestelltennummer** „konkateniert" werden, und führe auf der neuen Relation anschließend eine **Projektion** auf Spalte Name durch (Duplikate zu eliminieren?)

Was die Beispielanfragen bereits zeigen?

1. Es gibt oftmals **verschiedene Möglichkeiten**, **eine Datenbankanfrage** mit Hilfe der relationalen Basisoperationen **zu formulieren**. Dies betrifft teils die **Reihenfolge** der Verwendung der Basisoperationen, teils auch die **Häufig-**

keit der Benutzung einer Basisoperation (Bsp.: Projektion in Schritt a (2. Lösungsvariante oben) erforderlich??)

2. Es gibt darunter oftmals **Möglichkeiten**, die bezüglich der **Ausführungskosten** (Anfragebearbeitung durch das DBVS) **günstiger** aussehen als andere Möglichkeiten (Bsp.: **Erst** Selektion und **dann** Join (2. Lösungsvariante) **erscheint günstiger** als umgekehrte Reihenfolge (1. Lösungsvariante)) $\Rightarrow$ davon sollte man sich als Benutzer bei der Anfrageformulierung aber **nicht** leiten lassen, sondern eine Anfrage so formulieren, dass sie leicht verständlich ist[1]

$\Rightarrow$ es ist Aufgabe des rel. **DBVS**, eine Anfrage ggf. intern so zu **transformieren** (unsichtbar für den Benutzer), dass die Ausführungskosten minimal sind!!!

↳ Optimizer (Ausführungskostenvorhersage)

Ein (DBMS-)Optimizer berechnet die **zu erwartenden**[2] Ausführungskosten für verschiedene Lösungsvarianten und wählt die **Kostenminimale** LV aus. „Er bemüht sich redlich.“ ☺

[1] dies ist zumindest die „reine Lehre“ ...

[2] Schätzwerte

Kriterien für Anfragesprachen

(entnommen aus Heuer/Saake, S. 225)

- **Ad-hoc-Formulierung**: Der Benutzer soll eine **Anfrage** formulieren können, **ohne** ein **vollständiges Programm** schreiben zu müssen
- **Mengenorientiert**: **Jede** Operation soll auf **Mengen** von Daten „gleichzeitig“ arbeiten, nicht navigierend nur auf einzelnen Elementen (´one tuple at a time´)
- **Deskriptivität**: Der Benutzer soll formulieren „**Was** will ich haben?“ **und nicht** „**Wie** komme ich an das, was ich haben will?“
  [Hinweis: Auch bei der Formulierung des WAS* kommt man meist nicht daran vorbei, dies in einzelnen **Schritten** auszudrücken, etwa aus **Basisoperationen** zusammenzusetzen. Dies unterscheidet sich jedoch **wesentlich** von der prozeduralen/navigierenden Art, in der man das WIE bei hierarchischen/Netzwerk-Datenbanksystemen auszudrücken gezwungen ist!!]
- **Abgeschlossenheit**: Das Anfrageergebnis ist wieder eine **Relation** und kann wieder als Eingabe für die nächste Anfrage verwendet werden

* mit relationalen Anfragesprachen

- **Adäquatheit**: Alle Konstrukte des zugrundeliegenden Datenmodells werden unterstützt

- **Orthogonalität**: Sprachkonstrukte sind in ähnlichen Situationen auch ähnlich anwendbar [„Sprachkonstrukte sind miteinander weitgehend frei kombinierbar.“] Beispiel (Relationenalgebra): In dem Selektionsausdruck SL_F R (vgl. Folie 192) darf statt R auch wieder ein Selektionsausdruck stehen (SL_{F1} (SL_{F2} R)) → mag selbstverständlich erscheinen, ist es in der Realität (SQL, s.u.!!) aber leider nicht. Orthogonalität setzt Abgeschlossenheit voraus (als notwendige Bedingung): **Orthogonalität erleichtert den Umgang mit einer Sprache ganz wesentlich und erleichtert auch die Implementierung** und Optimierung (weniger/keine Ausnahmefälle der Art, dass Konstrukt x nicht mit Konstrukt y verträglich ist)!!

Relation

- **Optimierbarkeit**: Die Sprache besteht aus wenigen Operationen, für die es **Optimierungsregeln** gibt [Möglichkeit, einen komplexen, zusammengesetzten Sprachausdruck (**Anfrage**) so **umzubauen** in einen **semantisch äquivalenten** Ausdruck, dass die Ausführungskosten

(-zeit) minimal werden. **Optimierbarkeit** wird **wesentlich gefördert durch**

- **formale Sprachdefinition** (dann lassen sich Umbauvorschriften formal beschreiben und auch die semantische Äquivalenz vor/nach dem Umbau lässt sich beweisen)
- **Orthogonalität** (damit der Umbau auch wieder einen gültigen Sprachausdruck erzeugt)]

- **Effizienz**: Jede Operation ist **effizient** ausführbar (im Relationenmodell hat jede Operation eine Komplexität $\leq 0(n^2)$, n Anzahl der Tupel einer Relation) [wobei Ausführungskosten $0(n^2)$ natürlich möglichst vermieden werden sollten $\rightarrow$ z.B. 0(n) oder 0 (n log n), aber immerhin: keine Operation mit exponentiellem Kostenzuwachs bei steigendem n]
- **Sicherheit**: Keine Anfrage, die syntaktisch korrekt ist, darf in eine Endlosschleife geraten oder ein unendliches Ergebnis liefern

- **Eingeschränktheit**: Die Anfragesprache darf **keine komplette Programmiersprache** sein. Diese Eigenschaft folgt aus Sicherheit , Optimierbarkeit [Fähigkeit, die Ausführungskosten einer Anfrage vorherzusagen!] und Effizienz
- **Vollständigkeit**: Die Sprache muss mindestens die Anfragen einer „Standardsprache“ (à la Relationenalgebra, -kalkül (s.u.)) ausdrücken können

Relationenalgebra und –kalkül **erfüllen obige Kriterien**.

In der Praxis verwendete* Anfragesprachen, wie heutiges SQL, **erfüllen die Kriterien weitgehend**. Teilweise bewegen sie sich aber auch bewusst etwas davon weg (z.B. bzgl. der **Eingeschränktheit**), teilweise weisen sie Schwächen aufgrund ihrer Historie (Aufwärtskompatibilität von alten zu neuen Sprachversionen) auf (z.B. bzgl. der **Orthogonalität**).

* relationale