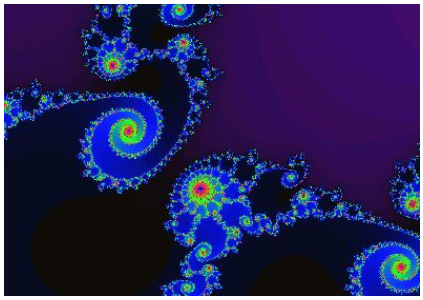
Datenbanken

8. Relationale Algebra



Fachhochschule Darmstadt
Fachbereich Informatik

Dr. Peter Nevermann

Rückblick

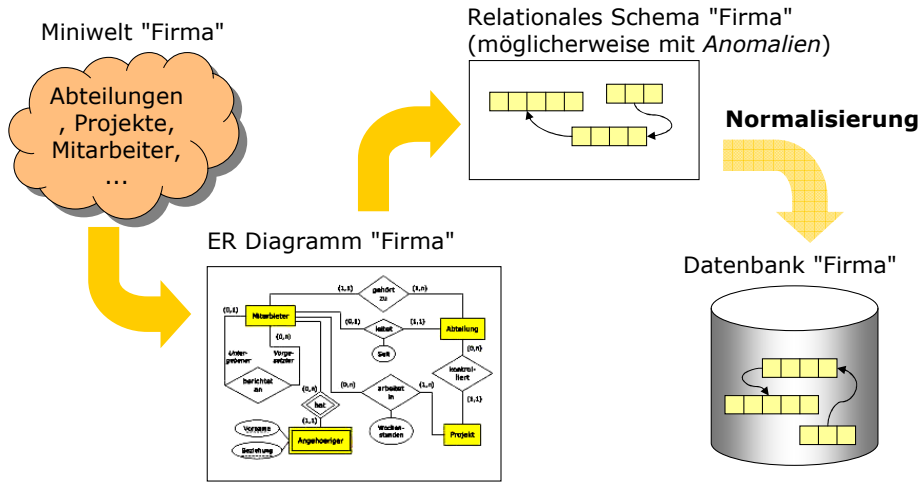
Transformation: ER-Entwurf → relationales Schema

Entity-Relationship Entwurf	Relationales Schema
Entity	Tabelle
Weak-Entity	Tabelle (FKey → Owner)
Beziehungen mit (1,1)-/(0,1)-Seite	FKey auf (1,1)-/(0,1)-Seite
Beziehungen ohne (1,1)-/(0,1)-Seite	Tabelle mit 2 FKey's
Einfaches Attribut (*)	Spalte
Mehrwertiges Attribut	Tabelle
Spezialisierungen:	
A: Eine eigene Tabelle jeweils für den Super- und jeden Subtyp	
B: Eine eigene Tabelle für jeden Subtyp	
C: Eine einzige Tabelle für die gesamte Spezialisierung	

(*) Einwertige, atomare Attribute und einfache Komponenten von einwertigen, gegliederten Attributen

Ausblick: Normalisierung

- **Finde ein geeignetes Datenbankschema für die Miniwelt "Firma"**



Wo stehen wir?

Wir haben zunächst unsere "Miniwelt" mittels ER Modellierung beschrieben.

Anschließend haben wir unseren ER Entwurf mittels bestimmter Regeln in ein relationales Schema (d.h. eine Anzahl von Tabellen-Definitionen) überführt.

Als Sprache für die Definition eines relationalen Schemas steht uns SQL DDL zur Verfügung.

Was fehlt denn jetzt noch?

Wir werden sehen, daß ein relationales Schema nicht unbedingt optimal ist. Unter Umständen weist nämlich ein Schema noch gewisse Defekte auf, die zu sogenannten *Anomalien* führen können. Durch einen Prozeß, den man *Normalisierung* nennt, können diese Defekte behoben werden.

Beruhigend ist, daß das relationale Schema, welches wir aus der ERM->RM Transformation gewonnen haben, in der Regel eine gute Qualität (d.h. keine oder nur wenige Defekte) aufweist.

Um diesen Prozeß der Normalisierung von Tabellen optimal beschreiben zu können, werden wir nun zunächst einen Formalismus einführen, den man **Relationale Algebra** oder **Relationenalgebra** (*relational algebra*) nennt.

Das relationale Modell

- **Begriffe**
  - Wertebereich (*domain*)
    - Name
    - Datentyp
    - Format
  - Relationales Schema
    - Name
    - Liste von Attributen
  - Attribut
    - Name
    - Wertebereich
  - Relation
    - Tupel
    - NULL Werte

$R (A_1, A_2, ..., A_n)$

$dom (A)$

$t \in r (R)$
$t = <v_1, v_2, ..., v_n>$
$v_i = t.A_i = t[A_i]$

Datenbanken I | 4 | SS 05, 03.06.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Wertebereich

•Ein *Wertebereich* (*domain*) ist eine Menge atomarer(*) Werte.

•Ein Wertebereich kann durch folgende Informationen beschrieben werden:

- •einen Namen (Personenname, Telefonnummer, Geldbetrag, Geburtsdatum, ...)
- •einen Datentyp (String, Ganzzahl, Gleitkommazahl, Datum, Boolesch, ...)
- •ein Format (YYYY-MM-DD für Datum, +009 000009 000000000009 für Telefonnummer, € 00.000.009,99 für Geldbetrag, ...)

(*) Ob ein Wert wie Name oder Adresse atomar ist oder nicht, ergibt sich aus den Anforderungen der Miniwelt.

Relationales Schema und Attribut

•Ein *relationales Schema* (oder kurz *Schema*) ist durch einen Namen und einer Liste von *Attributen* gegeben.

•Schreibweise:
$R (A_1, A_2, ..., A_n)$

•Beispiel:
MITARBEITER(personal-nr, nachname, vorname, geburtsdatum)

•Ein *Attribut* ist durch seinen Namen und der Zuordnung eines Wertebereichs gegeben.

•Schreibweise für den Wertebereich eines Attributs A ist: dom (A)

•Natürlich können Wertebereiche mehrfach innerhalb eines Schemas verwendet werden. So können z.B. im Schema "mitarbeiter" die Attribute "telefon-privat" und "telefon-büro" vorkommen, welche beide den Wertebereich Telefonnummer haben.

•Man kann auch sagen, daß ein Attribut der Name einer *Rolle* ist, die ein Wertebereich innerhalb eines Schemas spielt.

•Der Name eines Attributs muß innerhalb eines Schemas eindeutig sein, kann aber in unterschiedlichen Schemas mehrfach vorkommen. Dann kann es Sinn machen, den Attributnamen in der Form 'R.A' zu qualifizieren, z.B. MITARBEITER.nachname.

Tupel, Relation, NULL Werte

•Ein Tupel t zu einem Schema $R(A_1, A_2, ..., A_n)$ (oder kurz *R-Tupel*) besteht aus einer Liste von Werten $<v_1, v_2, ..., v_n>$, wobei $v_i \in dom(A_i)$ oder v_i ist NULL gilt. NULL ist nicht ein Wert wie jeder andere sondern hat eine besondere Semantik: "kein Wert" oder "Wert unbekannt".

•Für den Wert eines Tupels in einem Attribut gibt es auch folgende Schreibweisen:
$v_i = t [A_i] = t.A_i$

•Für eine Teilliste $S = (A_{i1}, A_{i2}, ..., A_{ik})$ von Attributen eines Schemas $R(A_1, A_2, ..., A_n)$ kann man entsprechend Subtupel $s = <v_{i1}, v_{i2}, ..., v_{ik}>$ bilden. Dazu gibt es auch folgende Schreibweisen:
$s = t [S] = t.S = t [A_{i1}, A_{i2}, ..., A_{ik}] = t.(A_{i1}, A_{i2}, ..., A_{ik})$

•Ein *relationaler Zustand* (oder kurz *Relation*) r(R) zu einem Schema $R(A_1, A_2, ..., A_n)$ ist eine Menge von R-Tupeln.

•Da ein Schema R einen Begriff der Miniwelt beschreibt, kann man wieder sagen, daß das Schema die *Intension*, die Relation die *Extension* des Begriffes ist. Das Schema ist relativ statisch (ändert sich selten oder nie), die Relation ist relativ dynamisch (kann sich sehr häufig ändern).

•Es ist üblich große Buchstaben für Schemas und kleine Buchstaben für Relationen zu verwenden. Allerdings passiert es auch schnell, daß beides durcheinander gerät und man aus dem Kontext die korrekte Bedeutung herauslesen muß. In diesem Skript soll "R" beides, das Schema und die Relation bezeichnen dürfen, $R(A_1, A_2, ..., A_n)$ steht aber nur für das Schema.

•Beispiele Schema:
- R oder $R(A_1, A_2, ..., A_n)$,
- S oder $S(B_1, B_2, ..., B_m)$,
- mitarbeiter oder MITARBEITER oder MITARBEITER(personal-nr, nachname, vorname, geburtsdatum)

Beispiele Relation:
- R oder r(R) oder r,
- S oder s(S) oder s,
- mitarbeiter oder MITARBEITER

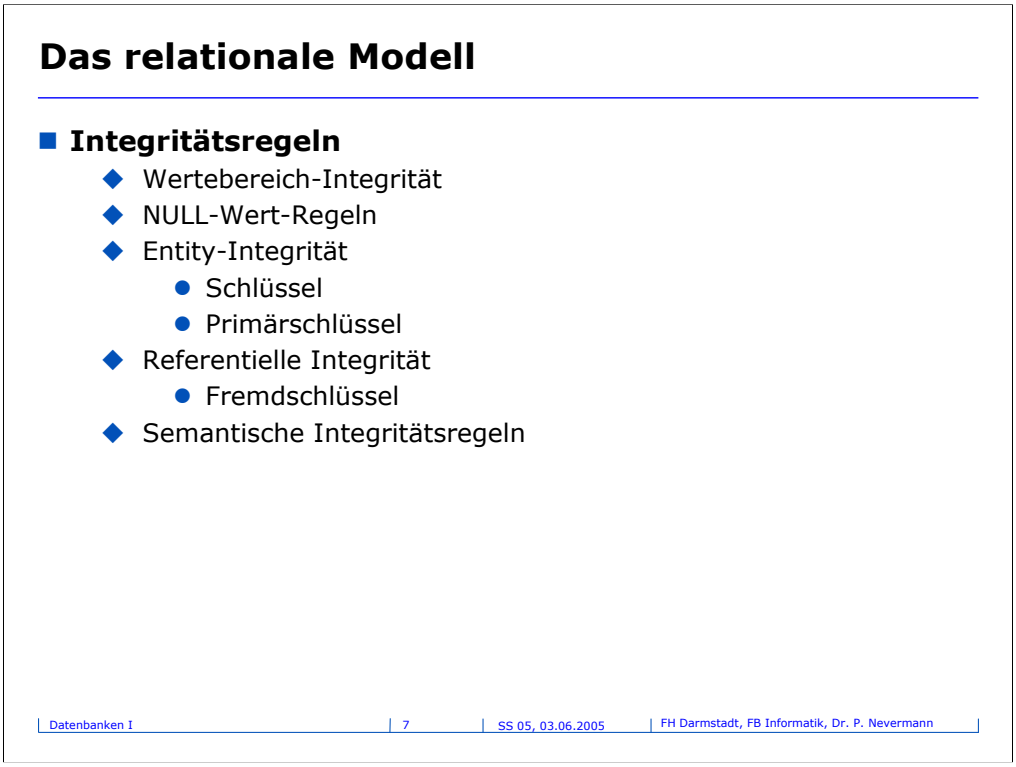
Relationales Datenbankschema, relationale Datenbank

•Eine *relationale Datenbank* (oder kurz *Datenbank*) besteht aus einer Menge von zusammengehörigen Relationen. Eine Datenbank ist ein Datenbestand.

•Ein *relationales Datenbankschema* (oder kurz *Datenbankschema*) besteht aus einer Menge von zusammengehörigen relationalen Schemas zusammen mit einer Menge von *Integritätsregeln* [was das ist kommt sofort ...]

SQL DDL und Datenbankschemas

Mit SQL DDL haben wir eine standardisierte Datendefinitionssprache kennengelernt, mit der man relationale Datenbankschemas für ein DBMS verständlich und interpretierbar beschreiben kann.



Integritätsregeln sind ein Bestandteil des Datenbankschemas und beschreiben die Bedingungen die zu erfüllen und zu überwachen sind, damit die Datenbank (der Datenbestand) konsistent bleibt.

Wertebereich-Integrität

•Wertebereich-Integrität ist durch die Zuordnung eines Wertebereichs zu jedem Attribut gegeben. Somit ist geregelt, welche Werte für ein Attribut zulässig sind.

• Z.B. darf ein Attribut "geb-datum" mit dem Wertebereich "Datum" nicht eine Telefonnummer als Wert bekommen, bzw. ein Attribut "geschlecht" mit Wertebereich {'W', 'M'} darf nicht den Wert 'X' haben.

NULL-Wert-Regeln

•NULL-Wert-Regeln geben an, ob NULL-Werte für ein Attribut erlaubt sind oder nicht.

•SQL DDL: NOT NULL

Identifizierende Attributlisten, Schlüssel, Primärschlüssel

•Eine Teilliste S = (A_{i1}, A_{i2}, ..., A_{ik}) von Attributen eines Schemas R(A_1, A_2, ..., A_n) bezeichnet man als *identifizierend*, wenn in jeder Relation r(R) gilt:
t_1 [S] ≠ t_2 [S], für alle t_1, t_2 ∈ r(R) mit t_1 ≠ t_2

•S = (A_1, A_2, ..., A_n), die vollständige Attributliste, ist trivialerweise identifizierend

•Ein *Schlüssel* (oder *Schlüsselkandidat*) von R ist eine identifizierende Attributliste S, die minimal ist, d.h. man kann kein Attribut aus S entfernen, ohne daß die Eigenschaft von S identifizierend zu sein verloren geht.

•In einer Relation kann es mehrere Schlüssel geben. Einen davon kann man als Primärschlüssel (*primary key*) auswählen.

•Schreibweise: die Attribute die den Primärschlüssel ausmachen kann man unterstreichen, z.B. R_1 (<u>A</u>, B, C, D) heißt S = (A) ist der Primärschlüssel, R_2 (U, <u>V</u>, <u>W</u>, X, Y, Z) bedeutet S= (V, W) ist der Primärschlüssel.

•SQL DDL: PRIMARY KEY

Entity-Integrität

•Diese Regel besagt, daß NULL-Werte im Primärschlüssel einer Relation niemals vorkommen dürfen.

Fremdschlüssel und referentielle Integrität

•Gegeben seien zwei Schemas R_1 (A_1, A_2, ..., A_n) und R_2 (B_1, B_2, ..., B_m). PK = (B_{j1}, B_{j2}, ..., B_{jp}) sei der Primärschlüssel von R2.
Eine Attributliste FK = (A_{i1}, A_{i2}, ..., A_{iq}) von R_1 heißt *Fremdschlüssel* auf R_2 falls:
a) FK und PK dieselbe Länge haben, d.h. p = q,
b) FK und PK dieselben Wertebereiche haben, d.h. dom (A_{ix}) = dom (B_{jx}) für x = 1, ..., p

•*Referentielle Integrität* ist gegeben, wenn alle Fremdschlüssel noch folgende Regel erfüllen
c) für jedes R_1-Tupel t_1 gilt t_1 [FK] ist NULL, oder es gibt ein R_2-Tupel t_2 mit t_1 [FK] = t_2 [PK] (man sagt dann auch t_1 referenziert t_2).

•SQL DDL: FOREIGN KEY ... REFERENCES ... ON UPDATE ... ON DELETE

Semantische Integritätsregeln

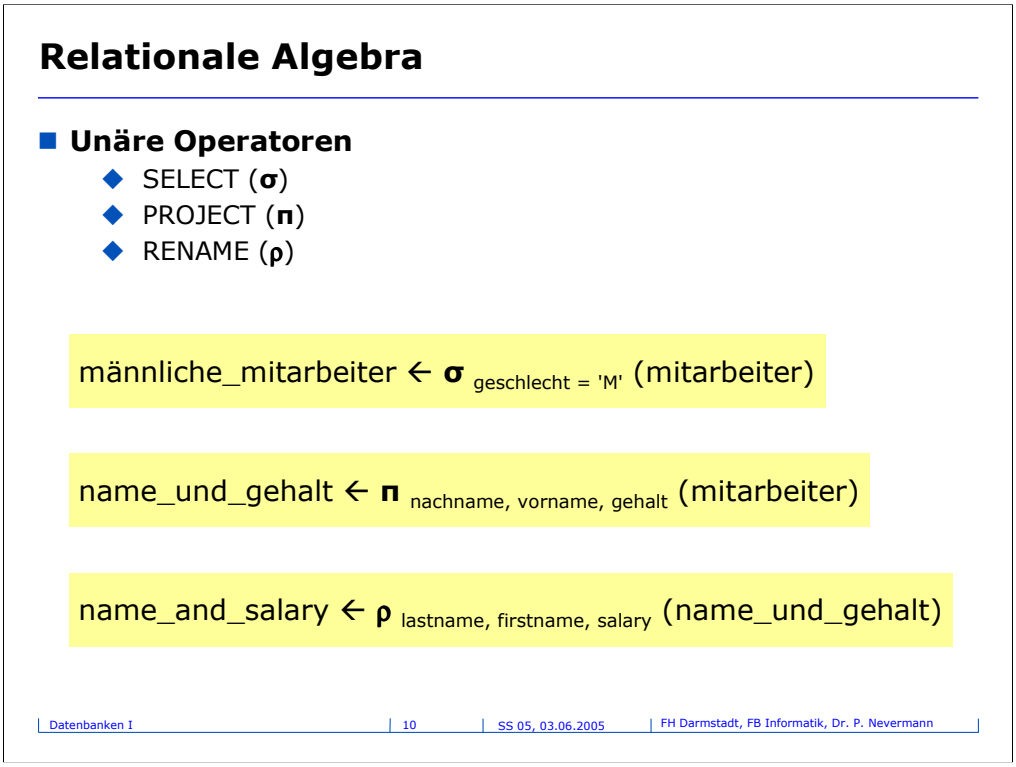
•Semantische Integritätsregeln sind an die Bedeutung der Attribute in der Miniwelt gekoppelt

•Beispiele:
Gehalt des Mitarbeiters darf nicht Gehalt des Vorgesetzten überschreiten
Wochenstundenzahlen eines Mitarbeiters über alle Projekte summiert muß ≤ 40 Stunden sein

•SQL DDL: CHECK(...), CREATE ASSERTION ..., Trigger

Zustands- und Zustandsübergangs-Bedingungen

Alle soweit vorgestellten Integritätsregeln sind Zustandsbedingungen, weil sie Aussagen zum Zustand einer Datenbank machen. Darüberhinaus kann es auch Regeln geben, die Aussagen über Zustandsübergänge machen, z.B. das Gehalt eines Mitarbeiters darf im Laufe der Zeit nicht abfallen.



Die Operatoren der relationalen Algebra (oder Relationenalgebra) operieren auf der Menge der Relationen einer Datenbank und produzieren neue Relationen. SQL DML ist eine standardisierte Sprache mit der die meisten Ausdrücke der relationalen Algebra für das DBMS verständlich und interpretierbar ausgedrückt werden können.

SELECT Operator (σ)

•Der SELECT Operator filtert aus einer Relation eine Tupel-Teilmenge aufgrund einer Bedingung aus. Die Ergebnisrelation hat dasselbe Schema (gleiche Attributliste) wie die Ausgangsrelation.

•Für die Formulierung der Bedingung stehen, wie bei SQL, die Vergleichsoperatoren (=, <, ≤, >, ≥, ≠), sowie die logischen Operatoren (NOT, AND, OR) zur Verfügung. Die Operatoren <, ≤, >, ≥ können allgemein auf geordnete Wertebereiche angewendet werden (Zahlen, Datum, String gemäß einer vorgegebenen Sortierfolge [*collation*]).

•Schreibweise:

$R' \leftarrow \sigma_{<bedingung>} (R)$

•Beispiel:
männliche_mitarbeiter ← $\sigma_{geschlecht = 'M'}$ (mitarbeiter)
Äquivalenter SQL-Ausdruck:
SELECT * FROM mitarbeiter
WHERE geschlecht = 'M';

•Die Verkettung des SELECT-Operators ist kommutativ:
$\sigma_{c1} (\sigma_{c2} (R)) = \sigma_{c2} (\sigma_{c1} (R)) = \sigma_{c1 \text{ AND } c2} (R)$

PROJECT Operator (π)

•Der PROJECT Operator reduziert eine Relation auf Teiltupel gemäß einer vorgegebenen Attribut-Teilliste.

•Die Ergebnisrelation ist auch bei diesem Operator eine Menge, enthält also (anders als bei SQL) keine Duplikate.

•Schreibweise:
R' ← $\pi_{<attributliste>} (R)$

•Beispiel:
name_und_gehalt ← $\pi_{nachname, vorname, gehalt}$ (mitarbeiter)
Äquivalenter SQL-Ausdruck:
SELECT DISTINCT nachname, vorname, gehalt FROM mitarbeiter;

•Die Verkettung des PROJECT-Operators ist nicht kommutativ:
$\pi_{S1} (\pi_{S2} (R)) = \pi_{S1} (R)$ [der Ausdruck ist nur dann gültig, wenn S1 von S2 umfaßt wird]

Ausdrücke in denen σ– und π–Operator kombiniert werden

•Beispiel:
name_und_gehalt_männlicher_mitarbeiter ← $\pi_{nachname, vorname, gehalt} (\sigma_{geschlecht = 'M'}$ (mitarbeiter))
Äquivalenter SQL-Ausdruck:
SELECT DISTINCT nachname, vorname, gehalt FROM mitarbeiter
WHERE geschlecht = 'M';

•Alternative Schreibweise mit zwei Ausdrücken:
männlicher_mitarbeiter ← $\sigma_{geschlecht = 'M'}$ (mitarbeiter)
name_und_gehalt_männlicher_mitarbeiter ← $\pi_{nachname, vorname, gehalt}$ (männlicher_mitarbeiter)

RENAME Operator (ρ)

•Der RENAME-Operator dient zum Umbenennen von Relationen und/oder Attributen. Anzahl und Wertebereiche der Attribute sowie Anzahl der Tupel bleibt durch diesen Operator unverändert.

•Schreibweisen:

$\rho_{S}(R)$	[R wird in S umbenannt]
$\rho_{B1, B2, ..., Bm}(R)$	[R's Attribute werden umbenannt]
$\rho_{S(B1, B2, ..., Bm)}(R)$	[R's Attribute und R selbst werden umbenannt]

•Alternative Schreibweisen ohne ρ -Operator:

$S \leftarrow R$	[R wird in S umbenannt]
$R(B_1, B_2, ..., B_m) \leftarrow R$	[R's Attribute werden umbenannt]
$S(B_1, B_2, ..., B_m) \leftarrow R$	[R's Attribute und R selbst werden umbenannt]

•Beispiel [beide Ausdrücke sind äquivalent]:

$name\_und\_gehalt \leftarrow \rho_{nname, vname, gehalt}\ \pi_{nachname, vorname, gehalt}(mitarbeiter)$

$name\_und\_gehalt(nname, vname, gehalt) \leftarrow \pi_{nachname, vorname, gehalt}(mitarbeiter)$

Äquivalenter SQL-Ausdruck:

```
SELECT DISTINCT nachname AS nname, vorname AS vname, gehalt FROM mitarbeiter;
```

Relationale Algebra

- **Binäre Operatoren**
  - UNION
  - INTERSECTION
  - DIFFERENCE
  - CROSS-PRODUCT
  - JOIN (oder INNER-JOIN)
  - OUTER-JOIN
  - NATURAL-JOIN

$R_1 \cup R_2$ $R_1 \times R_2$
$R_1 \cap R_2$ $R_1 \bowtie R_2$
$R_1 - R_2$ $R_1 * R_2$

Eine relationale Variante der Mengen-Operatoren sind ebenfalls Bestandteil der Relationenalgebra.

UNION-, INTERSECTION- und DIFFERENZ-Operatoren ($\cup$, $\cap$, $-$)

•Der mengentheoretischen Operatoren $\cup$, $\cap$, $-$ bilden jeweils die Vereinigung, den Durchschnitt und die Differenz von zwei Relationen. Dabei müssen die beteiligten Relationen R_1 (A_1, A_2, ..., A_n) und R_2 (B_1, B_2, ..., B_m) *tupel-kompatibel* sein, d.h.
a) die Anzahl der Attribute muß übereinstimmen ($n = m$)
b) die Wertebereiche müssen übereinstimmen ($dom(A_i) = dom(B_i)$, für alle $i = 1...n$)

•Die Namen der Attribute in der Ergebnisrelation werden vom ersten Operanden übernommen (und können mittels RENAME-Operator umbenannt werden).

•Schreibweise:
$R' \leftarrow R_1 \cup R_2$
$R' \leftarrow R_1 \cap R_2$
$R' \leftarrow R_1 - R_2$

•Beispiel:
result ← $\sigma_{\text{geschlecht = 'M'}}$ (mitarbeiter) ∪ $\sigma_{\text{geschlecht = 'W'}}$ (mitarbeiter)
Äquivalenter SQL-Ausdruck:

```
SELECT * FROM mitarbeiter WHERE geschlecht = 'M'
UNION
SELECT * FROM mitarbeiter WHERE geschlecht = 'W';
```

•Die Operatoren ∪ und ∩ sind kommutativ, der Differenz-Operator ist es nicht.

CROSS-PRODUCT-Operator (×)

•Der mengentheoretische Kreuzprodukt-Operator bildet die beteiligten Relationen R_1 (A_1, A_2, ..., A_n) und R_2 (B_1, B_2, ..., B_m) (*Operanden*) in eine Relation S (A_1, A_2, ..., A_n, B_1, B_2, ..., B_m) ab, und enthält alle Kombinationen von R_1- und R_2-Tupeln.

•Schreibweise:
S ← $R_1 \times R_2$

•Beispiel:
result ← mitarbeiter × abteilung
Äquivalenter SQL-Ausdruck:

```
SELECT * FROM mitarbeiter, abteilung;
```

•In der Regel ist man natürlich selten am vollen Kreuzprodukt interessiert, sondern selektiert sich daraus bestimmte (sinnvolle) Tupel aus, wie z.B.
result ← $\sigma_{\text{mitarbeiter.abtnr = abteilung.anr}}$ (mitarbeiter × abteilung)
Äquivalenter SQL-Ausdruck:

```
SELECT * FROM mitarbeiter, abteilung
WHERE mitarbeiter.abtnr = abteilung.anr;
```

JOIN-Operator (⋈)

•Der JOIN-Operator baut auf dem Kreuzprodukt auf und bietet eine Abkürzung für den Ausdruck $\sigma_{\text{<join-bedingung>}}$ (R × S) an.

•Schreibweise:
result ← R ⋈ $_{\text{<join-bedingung>}}$ S
[was, wie gesagt, äquivalent zu $\sigma_{\text{<join-bedingung>}}$ (R × S) ist]
Die Join-Bedingung hat dabei die Form:
A_{i1} θ B_{i1} AND A_{i2} θ B_{i2} AND ... A_{ik} θ B_{ik} mit θ ∈ {=, <, ≤, >, ≥, ≠}, A_{ix} Attribut von R, B_{ix} Attribut von S und beide Attribute A_{ix} und B_{ix} haben denselben Wertebereich.

•Aufgrund dieser Schreibweise nennt man diesen Join auch *Theta-Join*.

•Wenn $\theta \in \{=\}$ für alle Vergleiche der Join-Bedingung gilt, spricht man von einem *Equi-Join*. Man beachte, daß das Ergebnis eines Equi-Joins immer Paare von identischen "Spalten" enthält (wenn ich kurz mal auf die Tabellen-Denkweise wechseln darf), d.h. Paare von Attributen (die Attribute der Join-Bedingung nämlich) mit gleichen Werten in allen Tupeln.

•Die doppelten/überflüßigen Attribute in der Ergebnisrelation kann man natürlich durch Anwendung des PROJECT-Operators entfernen.

NATURAL-JOIN-Operator ($\bowtie$ oder *)

•Der NATURAL-JOIN-Operator führt ein Equi-Join aus, bei dem die Attribute der Join-Bedingung genau diejenigen Attribute sind, welche in beiden Relationen denselben Namen und Wertebereich haben. Außerdem enthält die Ergebnisrelation keine doppelten Attribute.

•Schreibweise:
result ← R $\bowtie$ S (oder manchmal auch result ← R * S)
[was, wie gesagt, äquivalent zu $\pi_{\text{<attributliste>}}$ (R $\bowtie_{\text{<join-bedingung>}}$ S) ist, wobei in <attributliste> die doppelten Attribute fehlen]

•Im Extremfall, daß R und S keine gleichnamigen Attribute haben, gilt R $\bowtie$ S = R $\times$ S. Im anderen Extremfall, daß R und S äquivalente Schemas haben (d.h. alle Attribute korrespondieren im Namen und Wertebereich), gilt R $\bowtie$ S = R $\cap$ S. Keiner von beiden Fällen ist von besonderer Bedeutung.

•In den meisten Fällen sind die Attributlisten über die ein Join gebildet wird Primärschlüssel in der einen und Fremdschlüssel in der anderen Relation, d.h. in den meisten Fällen *realisiert* ein Join eine Beziehung zwischen zwei Entities.

OUTER-JOIN-Operator (=$\bowtie$ oder $\bowtie$= oder =$\bowtie$=)

•Vom OUTER-JOIN gibt es drei Varianten: LEFT-OUTER-JOIN, RIGHT-OUTER-JOIN und FULL-OUTER-JOIN.

•Schreibweise:
result ← R =$\bowtie_{\text{<join-bedingung>}}$ S LEFT-OUTER-JOIN
result ← R $\bowtie$=$_{\text{<join-bedingung>}}$ S RIGHT-OUTER-JOIN
result ← R =$\bowtie$=$_{\text{<join-bedingung>}}$ S FULL-OUTER-JOIN

•Der LEFT-OUTER-JOIN ($R =\bowtie_{<join\text{-}bedingung>} S$) unterscheidet sich vom gewöhnlichen JOIN ($R \bowtie_{<join\text{-}bedingung>} S$) darin, daß von der "linken" der beiden beteiligten Relationen (d.h. R) auch noch diejenigen Tupel in die Ergebnisrelation gelangen, welche nicht der Join-Bedingung genügen. Diese werden im "S-Teil" des Ergebnistupels mit NULL aufgefüllt.

•Analog ist der RIGHT- und der FULL-OUTER-JOIN zu definieren (beim FULL-OUTER-JOIN hat man die restlichen Tupel beider Relationen hinzu (jeweils mit NULL aufgefüllt).

Relationenalgebra Erweiterungen

Manche Operationen, die wir aus SQL DML bereits kennen, lassen sich mit Relationenalgebra in dem Umfang wie bisher vorgestellt nicht ausdrücken. Dazu gehören Aggregatfunktionen und Gruppierungen. Deshalb wurde die Relationenalgebra um solche Operation noch erweitert.

Aggregatfunktionen und Gruppierung ($\Im$ - "*script F*")

•Schreibweise:

result $\leftarrow {}_{<liste\text{-}der\text{-}gruppierungsattribute>} \Im_{<liste\text{-}der\text{-}aggregatfunktionen>} (R)$

•Die Liste der Aggregatfunktionen ist von der Form: $f1(a_1), f2(a_2), ..., fn(a_n)$, wobei $a_1, ..., a_n$ Attribute der Relation R sind.

•Die Ergebnisrelation besteht aus den Gruppierungsattributen und zusätzlich einem Attribut je Aggregatfunktion.

•Beispiel:

durchschnittsgehaelter_pro_abteilung $\leftarrow \rho_{abtnr,\ durchschnittsgehalt} ({}_{abtnr}\Im_{AVG(gehalt)} (mitarbeiter))$

(Die zusätzliche Anwendung des RENAME-Operators in dem Beispiel soll nur die Struktur der Ergebnisrelation klarer machen.)

Aufgaben

Formulieren Sie folgende SQL Abfragen mit Ausdrücken der Relationenalgebra:

•SELECT m.nachname, m.vorname, m.gehalt, a.abtname AS abteilung
FROM (mitarbeiter AS m JOIN abteilung AS a ON m.abtnr = a.anr)
WHERE m.gehalt > 60000 AND a.abtname <> 'Verwaltung';

•SELECT abtnr AS abteilung, max(gehalt) AS maximalgehalt, avg(gehalt) AS durchschnittsgehalt
FROM mitarbeiter
GROUP BY abtnr;

•SELECT nachname, vorname
FROM mitarbeiter
WHERE abtnr = 1 AND geschlecht = 'M'
UNION
SELECT nachname, vorname
FROM mitarbeiter
WHERE abtnr = 2 AND geschlecht = 'W';

Übersetzen Sie folgende Relationenalgebra-Ausdrücke nach SQL:

•name_und_gehalt $\leftarrow \rho_{\text{name, gehalt, abteilung}} (\pi_{\text{nachname, gehalt, abtname}} (\text{mitarbeiter} \bowtie_{\text{mitarbeiter.abtnr = abteilung.anr}} \text{abteilung})$

•verwaltungsdamen $\leftarrow \pi_{\text{pnr, nachname, vorname}} (\sigma_{\text{<abtname = 'Verwaltung'>}} (\text{mitarbeiter} \bowtie_{\text{mitarbeiter.abtnr = abteilung.anr}} \text{abteilung}) \cap \sigma_{\text{geschlecht = 'W'}} (\text{mitarbeiter})$

Ausblick

- **Normalisierung**
  - Minimierung von Redundanzen
  - Vermeidung von übermäßig vielen NULL Werten
  - Vermeidung von INSERT, UPDATE und DELETE Anomalien