Teil 4: Relationale Algebra

Literatur:

- Elmasri/Navathe:Fundamentals of Database Systems, 3. Auflage, 1999.
  Section 7.4 “Basic Relational Algebra Operations”,
  Section 7.5 “Additional Relational Algebra Operations”,
  Section 7.6 “Examples of Queries in Relational Algebra”
- Kemper/Eickler: Datenbanksysteme, 4. Auflage, 2001.
  Kapitel 3.4, “Die relationale Algebra”
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3. Auflage, 1999.
  Section 3.2: “The Relational Algebra”
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.
- Codd: A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387, 1970. Reprinted in CACM 26(1), 64–69, 1983.
  Vgl. auch: `http://www1.acm.org:81/classics/nov95/toc.html` (unvollständig)

Lernziele

Nach diesem Kapitel sollten Sie folgendes können:

- Die Operationen der relationalen Algebra aufzählen und erklären.

  Vor allem sollten Sie die fünf Basisoperationen kennen.

- Anfragen des Typs “Verbund-Selektion-Projektion” in relationaler Algebra formulieren.

  Außerdem einfache Anfragen schreiben, die auch Mengendifferenz und Vereinigung enthalten.

- Über Korrektheit und Äquivalenz von Anfragen der relationalen Algebra diskutieren.

Inhalt

Beispiel-Datenbank (1)

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

Beispiel-Datenbank (2)

- `STUDENTEN`: enthält eine Zeile für jeden Studenten.
  - ◇ `SID`: “Studenten-ID” (eindeutige Nummer).
  - ◇ `VORNAME`, `NACHNAME`: Vor- und Nachname.
  - ◇ `EMAIL`: Email-Adresse (kann NULL sein).
- `AUFGABEN`: enthält eine Zeile für jede Aufgabe.
  - ◇ `ATYP`: Typ/Kategorie der Aufgabe.

    Z.B. `'H'`: Hausaufgabe, `'Z'`: Zwischenklausur, `'E'`: Endklausur.
  - ◇ `ANR`: Aufgabennummer (innerhalb des Typs).
  - ◇ `THEMA`: Thema der Aufgabe.
  - ◇ `MAXPT`: Maximale/volle Punktzahl der Aufgabe.

Beispiel-Datenbank (3)

- `BEWERTUNGEN`: enthält eine Zeile für jede abgegebene Lösung zu einer Aufgabe.
  - ◇ `SID`: Student, der die Lösung abgegeben hat.

    Dies referenziert eine Zeile in `STUDENTEN`.
  - ◇ `ATYP`, `ANR`: Identifikation der Aufgabe.

    Zusammen identifiziert dies eine Zeile in `AUFGABEN`.
  - ◇ `PUNKTE`: Punkte, die der Student für die Lösung bekommen hat.
  - ◇ Falls es keinen Eintrag für einen Studenten und eine Aufgabe gibt: Aufgabe nicht abgegeben.

Beispiel-Datenbank (4)

Integritätsbedingungen:

- Schlüssel:
  - ◇ In `STUDENTEN` gibt es keine zwei Zeilen mit der gleichen `SID`.

    $\forall\ \texttt{STUDENTEN X},\ \texttt{STUDENTEN Y}:\ \ \texttt{X.SID} = \texttt{Y.SID} \rightarrow \texttt{X} = \texttt{Y}$
  - ◇ In `AUFGABEN` gibt es keine zwei Zeilen, die sowohl im `ATYP` als auch in `ANR` übereinstimmen.

    $\forall\ \texttt{AUFGABEN X},\ \texttt{AUFGABEN Y}:\ \ \texttt{X.ATYP} = \texttt{Y.ATYP} \wedge \texttt{X.ANR} = \texttt{Y.ANR} \rightarrow \texttt{X} = \texttt{Y}$
  - ◇ In `BEWERTUNGEN` gibt es keine zwei Zeilen, die in `SID`, `ATYP` und `ANR` übereinstimmen.

    $\forall\ \texttt{BEWERTUNGEN X},\ \texttt{BEWERTUNGEN Y}:$
    $\texttt{X.SID} = \texttt{Y.SID} \wedge \texttt{X.ATYP} = \texttt{Y.ATYP} \wedge \texttt{X.ANR} = \texttt{Y.ANR} \rightarrow \texttt{X} = \texttt{Y}$

Beispiel-Datenbank (5)

- Fremdschlüssel:
  - ◇ Jeder Wert für `SID`, der in `BEWERTUNGEN` auftaucht, existiert auch in `STUDENTEN`.

    $\forall\ \texttt{BEWERTUNGEN X}\ \exists\ \texttt{STUDENTEN Y}:\ \ \texttt{X.SID} = \texttt{Y.SID}$
  - ◇ Jede Kombination von Werten für `ATYP` und `ANR` aus `BEWERTUNGEN` existiert auch in `AUFGABEN`.

    $\forall\ \texttt{BEWERTUNGEN X}\ \exists\ \texttt{AUFGABEN Y}:\ \ \texttt{X.ATYP} = \texttt{Y.ATYP} \wedge \texttt{X.ANR} = \texttt{Y.ANR}$
- Sonstiges (Beispiel):
  - ◇ Die Punktzahl ist nicht negativ.

    $\forall\ \texttt{BEWERTUNGEN X}:\ \ \texttt{X.PUNKTE} \geq 0$

Relationales Modell: Wdh. (1)

- Eine Relation R mit Attributen $A_1, \ldots, A_n$ der Datensorten/Datentypen $D_1, \ldots, D_n$ wird interpretiert (im DB-Zustand $\mathcal{I}$) durch eine endliche Relation
$$\mathcal{I}[R] \subseteq \mathcal{I}_\mathcal{D}[D_1] \times \cdots \times \mathcal{I}_\mathcal{D}[D_n].$$
- Z.B. ist (101, 'H', 1, 10) $\in \mathcal{I}$[BEWERTUNGEN].
- Im Bereichskalkül werden die Attribute durch ihre Position identifiziert, R ist dann ein Prädikat.
- Im Tupelkalkül werden die Attribute über Namen identifiziert, formal ist R hier eine Sorte und A_i sind Zugriffsfunktionen $R \to D_i$.

Relationales Modell: Wdh. (2)

- Im Tupelkalkül werden aber auch nur solche Interpretationen betrachtet, bei denen
  - ◇ die Sorten für die Datenbank-Relationen als Relation interpretiert werden, und
  - ◇ die Zugriffsfunktionen jeweils eine Komponente des Tupels liefern.
- Insofern sind die Unterschiede rein formaler Natur.
- In der Praxis braucht man beides:
  - ◇ Attributnamen sind natürlich praktisch.
  - ◇ Es gibt aber auch eine erste, zweite, etc. Spalte.

Noch eine Formalisierung (1)

- Es gibt noch eine dritte Formalisierung:
  - ◇ Man kann Tupel als Variablenbelegung auffassen, wobei die Attribute A_i die Variablen sind (der Sorte D_i).
  - ◇ Damit ordnet jedes Tupel den Attributen einen Wert des entsprechenden Datentyps zu.

    Da Tupel mathematisch als Abbildung von $\{1, 2, \ldots, n\}$ auf die entsprechenden Komponentenwerte formalisiert sind, werden hier nur Attributpositionen durch Attributnamen ersetzt.
  - ◇ Eine Relation ist dann eine Menge von Variablenbelegungen bezüglich der Variablendeklaration $\{A_1/D_1, \ldots, A_n/D_n\}$.

Noch eine Formalisierung (2)

- Diese Formalisierung (Relation als endliche Menge von Variablenbelegungen) ist für die Relationenalgebra besonders praktisch weil:
  - ◇ Man so einen Teil der Definitionen aus der Logik mitbenutzen kann (z.B. Term, atomare Formel).
  - ◇ Die Relationenalgebra sich so auf andere Logikbasierte Datenmodelle verallgemeinern läßt.
  - ◇ Man so recht leicht einsehen kann, daß Relationenalgebra und bereichsbeschränkte Formeln gleich mächtig sind.

Inhalt

Relationale Algebra (1)

- Die relationale Algebra (RA) ist eine theoretische Anfragesprache für das relationale Modell.
- Die relationale Algebra wird in keinem kommerziellen System an der Benutzerschnittstelle verwendet.
- Varianten der RA werden aber genutzt, um Auswertungspläne für Anfragen im DBMS darzustellen.
- Kenntnis der relationalen Algebra hilft, SQL und relationale Datenbanksysteme besser zu verstehen.

  Z.B. spricht man auch bei SQL-Anfragen über "Joins" (Operation der relationalen Algebra). Explizite Joins wurden in SQL-92 eingeführt.

Relationale Algebra (2)

- Eine Algebra ist eine Menge zusammen mit Operationen auf dieser Menge.
- Zum Beispiel bildet die Menge von Integers zusammen mit den Operationen $+$ und $*$ eine Algebra.
- Im Fall der relationalen Algebra ist diese Menge die Menge aller endlichen Relationen.

  Wie oben erklärt, rechnet die Relationenalgebra mit endlichen Mengen von Variablenbelegungen. Das ist aber isomorph zu Relationen.
- Eine Operation der relationalen Algebra ist $\cup$ (die Vereinigung). Relationen sind ja Mengen.

Relationale Algebra (3)

- Eine weitere Operation der RA ist die Selektion.

  Im Gegensatz zu Operationen wie $+$ für Zahlen ist die Selektion σ durch eine einfache Bedingung parametrisiert.
- Z.B. selektiert $\sigma_{\texttt{SID=101}}$ alle Tupel der Eingaberelation, die den Wert "101" in der Spalte "SID" haben:

$\sigma_{\texttt{SID=101}}$ (

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

) =

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12

Relationale Algebra (4)

- Da die Ausgabe einer Operation der relationalen Algebra wieder eine Relation ist, kann sie Eingabe für eine andere Operation der Algebra sein.

  Usw., bis das Ergebnis der Anfrage berechnet wird (dieses ist wieder eine Relation). Die relationale Algebra ist so einfach, da das relationale Modell nur ein Konstrukt enthält: die Relation.

- Eine Anfrage ist ein Term/Ausdruck in der Algebra.
- Zum Vergleich: arithmetischer Ausdruck $(x+2)*y$.
- In relationaler Algebra verknüpft man Relationen:

$$\pi_{\texttt{NACHNAME}}(\texttt{STUDENTEN} \bowtie \sigma_{\texttt{ATYP}='\texttt{Z}'}(\texttt{BEWERTUNGEN})).$$

Relationale Algebra (5)

Kleine Datenmodell-Unterschiede zu SQL:

- Nullwerte werden in der Definition der relationalen Algebra normalerweise ausgeschlossen, außer wenn Operationen wie äußerer Verbund definiert werden.

  Auch beim äußeren Verbund betrachtet man den Nullwert als einen zusätzlichen Wert zu jedem Datentyp. Die Entsprechung zu einer dreiwertigen Logik wie in SQL ist recht kompliziert.

- Die RA behandelt Relationen als Mengen, d.h. Duplikate werden automatisch eliminiert.

  In SQL sind Relationen Multimengen und können das gleiche Tupel mehrfach enthalten. Falls erforderlich, muß man die Duplikatelimination explizit verlangen (mit "`DISTINCT`"). In der relationalen Algabra muß man über Duplikate nicht nachdenken.

Relationale Algebra (6)

Bedeutung der RA für die DB-Theorie:

- Die relationale Algebra ist viel einfacher als SQL. Sie hat nur fünf Basisoperationen und kann vollständig auf einer Seite definiert werden.
- Die Relationale Algebra ist auch ein Maßstab, um die Ausdruckskraft von Anfragesprachen zu messen.
- Z.B. kann jede Anfrage, die in relationaler Algebra formuliert ist, auch in SQL formuliert werden.

  D.h. SQL ist zumindest mächtiger als die relationale Algebra. Umgekehrt kann jede Anfrage des SQL-Kerns (ohne Nullwerte, Aggregationen und Duplikate) auch in relationaler Algebra formuliert werden. Vgl. auch Folie 4-116.

Selektion (1)

- Die Operation σ_φ selektiert eine Teilmenge der Tupel einer Relation, nämlich die, die die Bedingung φ erfüllen. Die Selektion wirkt wie ein Filter auf die Eingabemenge.

  σ ist der griechische Buchstabe sigma, φ der griech. Buchstabe phi. Alle Lehrbücher verwenden σ für Selektion, aber φ ist kein Standard. In ASCII schreibt man z.B. `SEL[Bedingung](Relation)`.

- Beispiel:

$$\sigma_{A=1}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 5 \\ \hline \end{array}\right) = \begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ \hline \end{array}$$

Selektion (2)

- Die Selektionsbedingung φ ist eine atomare Formel bezüglich
  - der Datentyp-Signatur $\Sigma_{\mathcal{D}}$, und
  - der Variablendeklaration ν, die für jedes Attribut der Eingaberelation eine Variable gleichen Namens und gleicher Sorte enthält.
- Typischerweise hat die Bedingung φ die Form:

  $\langle$Term$\rangle$ $\langle$Vergleichsoperator$\rangle$ $\langle$Term$\rangle$

  Die Bedingung liefert einen Wahrheitswert (wahr oder falsch) für ein gegebenes Eingabetupel. Vergleichsoperatoren: $=$, $\neq$, $<$, $\leq$, $>$, $\geq$.

Selektion (3)

- Ein Term ("Wertausdruck") ist typischerweise
  - ein Attributname, oder
  - eine Datentypkonstante.

  Er kann für ein gegebenes Tupel zu einem Datentypelement ausgewertet werden.
- Im allgemeinen können aber alle in der Datentypsignatur $\Sigma_{\mathcal{D}}$ deklarierten Funktionen benutzt werden, um komplexe Terme zu bilden, z.B. $+$, $-$, $*$.
- Entsprechend können in der Selektionsbedingung alle Datentyp-Prädikate verwendet werden, z.B. `odd`.

Selektion (4)

- Beispiele für Bedingungen:
  - `NACHNAME = 'Weiss'`
  - `PUNKTE >= 8`
  - `PUNKTE = MAXPT` (die Eingaberelation muß beide Attribute enthalten).
- Natürlich müssen die Attribute aus der Selektionsbedingung auch in der Eingabetabelle vorkommen:

$$\sigma_{C=1}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 2 & 4 \\ \hline\end{array}\right) = \textbf{Error}$$

Selektion (5)

- $\sigma_\varphi(R)$ kann wie folgt implementiert werden:

```
Create new temporary relation T;
foreach tuple t from input relation R do
    Evaluate condition φ for tuple t;
    if true then
        insert t into T;
    fi
od;
return T;
```

- Mit anderen Datenstrukturen (B-Baum Index) kann es möglich sein, $\sigma_\varphi(R)$ zu berechnen, ohne jedes Tupel aus der Eingaberelation durchzugehen.

Erweiterte Selektion (1)

- In der grundlegenden Selektions-Operation sind nur atomare Formeln (meist Vergleiche) als Bedingungen möglich.
- Man kann aber auch die Kombination dieser atomaren Formeln durch die logischen Junktoren $\wedge$ (und), $\vee$ (oder), und $\neg$ (nicht) zulassen:

φ_1	φ_2	$\varphi_1 \wedge \varphi_2$	$\varphi_1 \vee \varphi_2$	$\neg\varphi_1$
falsch	falsch	falsch	falsch	wahr
falsch	wahr	falsch	wahr	wahr
wahr	falsch	falsch	wahr	falsch
wahr	wahr	wahr	wahr	falsch

Erweiterte Selektion (2)

- Die Selektionsbedingung muß die Auswertung für jedes Eingabetupel getrennt/einzeln zulassen.

  Somit sind "es gibt" ($\exists$) und "für alle" ($\forall$) und verschachtelte Anfragen in Selektionsbedingungen nicht erlaubt.

- Die erweiterte Form der Selektion ist nicht notwendig, da man sie immer durch die Basisoperationen der relationalen Algebra ausgedrücken kann.

  Aber sie ist bequem.

- Z.B. ist $\sigma_{\varphi_1 \wedge \varphi_2}(R)$ äquivalent zu $\sigma_{\varphi_1}(\sigma_{\varphi_2}(R))$.

  $\vee$ und $\neg$ benötigen $\cup$ (Union) und $-$ (Mengendifferenz) die auch zu den Basisoperationen der relationalen Algebra gehören (s.u.): $\sigma_{\varphi_1 \vee \varphi_2}(R)$ ist äquivalent zu $\sigma_{\varphi_1}(R) \cup \sigma_{\varphi_2}(R)$ und $\sigma_{\neg\varphi}(R)$ ist äquivalent zu $R - \sigma_{\varphi}(R)$.

Übung

Schreiben Sie folgende Anfragen in relationaler Algebra:

- Welche Aufgaben behandeln "SQL"?

  Geben Sie die ganze Zeile der Tabelle aus. Die Projektion (zur Eliminierung von Spalten) wird unten behandelt.

- Geben Sie alle Bewertungen für Hausaufgabe 1 aus, bei denen weniger als 10 Punkte erzielt wurden.

Dies bezieht sich auf das Schema auf Folie 4-4:

- `STUDENTEN(`<u>`SID`</u>`, VORNAME, NACHNAME, EMAIL`^{o}`)`
- `AUFGABEN(`<u>`ATYP`</u>`, `<u>`ANR`</u>`, THEMA, MAXPT)`
- `BEWERTUNGEN(`<u>`SID`</u>`→STUDENTEN, (`<u>`ATYP`</u>`, `<u>`ANR`</u>`)→AUFGABEN, PUNKTE)`

Projektion (1)

- Die Projektion π wird typischerweise benutzt, um Attribute (Spalten) aus der Eingaberelation zu eliminieren (so wie die Selektion Zeilen eliminiert).

  π ist der griechische Buchstabe "pi".
  In Datenbanken bedeutet dies immer "Projektion", und nicht 3.14....
  Einige Autoren verwenden ein großes Π zur Unterscheidung.
  In ASCII kann man z.B. schreiben: `PROJ[Spalten](Relation)`.

- Beispiel:

$$\pi_{A,C}\left(\begin{array}{|c|c|c|}\hline A & B & C \\ \hline 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 3 & 9 \\ \hline\end{array}$$

Projektion (2)

- Im allgemeinen hat die Projektion die Form

$$\pi_{B_1 \leftarrow t_1, \ldots, B_k \leftarrow t_k}(R),$$

wobei die B_i Attributnamen sind (die nicht in der Eingabe vorzukommen brauchen), und die t_i Terme bezüglich
  - ◇ der Datentyp-Signatur $\Sigma_{\mathcal{D}}$, und
  - ◇ der Variablendeklaration ν, die für jedes Attribut der Eingaberelation eine Variable gleichen Namens und gleicher Sorte enthält.
- $B_i \neq B_j$ für $i \neq j$ (keine doppelten Attribute).

Projektion (3)

- Die Projektion erzeugt für jedes Eingabetupel ein Ausgabetupel.

  Falls jedoch zwei Eingabetupel zum gleichen Ausgabetupel führen, wird das Duplikat eliminiert.
- Sei $\mathcal{A}$ die Variablenbelegung, die einem Eingabetupel $(A_1 \colon d_1, \ldots, A_n \colon d_n)$ entspricht.
- Dann erzeugt die Projektion

$$\pi_{B_1 \leftarrow t_1, \ldots, B_k \leftarrow t_k}(R),$$

daraus das Tupel

$$(B_1 \colon \langle \mathcal{I}_{\mathcal{D}}, \mathcal{A} \rangle [t_1], \ldots, B_k \colon \langle \mathcal{I}_{\mathcal{D}}, \mathcal{A} \rangle [t_k]).$$

Projektion (4)

- Die Projektion wendet eine Abbildung auf jedes Eingabetupel an und liefert die Menge der Funktionswerte (Ausgabetupel).

  Wie "`map`" in funktionalen Sprachen, z.B. Lisp.
- Jedes Eingabetupel wird lokal auf ein Ausgabetupel abgebildet.

  D.h. es sind nur Funktionen erlaubt, deren Ausgabe nur von jeweils einem Eingabetupel abhängt. Man kann also mit der Projektion nicht Werte verschiedener Eingabetupel zu einem Ausgabetupel zusammenfassen (siehe aber kartesisches Produkt unten). Ansonsten sind sehr allgemeine Tupel-zu-Tupel Funktionen erlaubt.

Projektion (5)

- Normalerweise gibt es ein Ausgabetupel für jedes Eingabetupel. Falls jedoch zwei Eingabetupel zu dem gleichen Ausgabetupel führen, wird das Duplikat eliminiert.

  DBMS verwenden eine explizite Ausgabeeliminierung, wenn benötigt. Aber in der Theorie sind Relationen Mengen.
- Beispiel:

$$\pi_B \left(\begin{array}{|c|c|} \hline A & B \\ \hline 1 & 4 \\ 2 & 5 \\ 3 & 4 \\ \hline \end{array} \right) = \begin{array}{|c|} \hline B \\ \hline 4 \\ 5 \\ \hline \end{array}$$

Projektion (6)

- $\pi_{B_1 \leftarrow t_1, \ldots, B_k \leftarrow t_k}(R)$:

Create new temporary relation T;
foreach $(A_1{:}\,d_1, \ldots, A_n{:}\,d_n)$ **in** R **do**
    for $i = 1$ **to** k **do**
        Let d'_i be the value of t_i for the
            input tuple $(A_1{:}\,d_1, \ldots, A_n{:}\,d_n)$
            /* e.g. if $t_i = A_j$ then $d'_i = d_j$ */
    insert $(B_1{:}\,d'_1, \ldots, B_k{:}\,d'_k)$ **into** T;
od;
return T;

Diese Programmskizze setzt voraus, daß "**insert**" Duplikate eliminiert.

Projektion (7)

- Falls ein Ergebnisterm selbst ein Attributname ist, und man diesen Attributnamen auch im Ausgabetupel verwenden will, ist es nicht nötig, explizit einen neuen Namen anzugeben.
- $\pi_{A_{i_1}, \ldots, A_{i_k}}(R)$ ist also eine Abkürzung für

$$\pi_{A_{i_1} \leftarrow A_{i_1}, \ldots, A_{i_k} \leftarrow A_{i_k}}(R).$$

- Da es die typische Anwendung der Projektion ist, überflüssige Attribute/Spalten "wegzuprojizieren", ist das eine sehr nützliche Abkürzung.

Projektion (8)

Weitere Beispiele:

- Umbenennung einer Spalte (von SID in NR):

$$\pi_{\texttt{NR} \leftarrow \texttt{SID},\ \texttt{VORNAME},\ \texttt{NACHNAME}}(\texttt{STUDENTEN}).$$

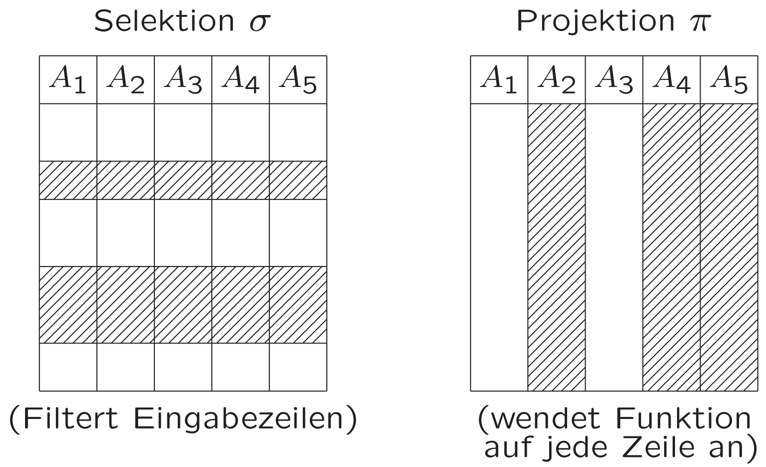
- Berechnung eines neuen Spaltenwertes durch eine Datentyp-Operation (String-Kontatenation):

$$\pi_{\texttt{SID},\ \texttt{NAME} \leftarrow \texttt{VORNAME || ' ' || NACHNAME}}(\texttt{STUDENTEN}).$$

- Erstellung einer neuen Spalte mit konstantem Wert (nützlich als Eingabe für eine Vereinigung mit einer Teilrelation mit einem anderen Wert):

$$\pi_{\texttt{SID},\ \texttt{VORNAME},\ \texttt{NACHNAME},\ \texttt{NOTE} \leftarrow \texttt{1}}(\texttt{STUDENTEN}).$$

Zusammenfassung



Operationen schachteln (1)

- Da das Ergebnis einer Operation der relationalen Algebra wieder eine Relation ist, kann man es als Eingabe für eine weitere Operation verwenden.
- Zum Beispiel, wenn man die abgegebenen Übungen von Student 102 ausgeben möchte:

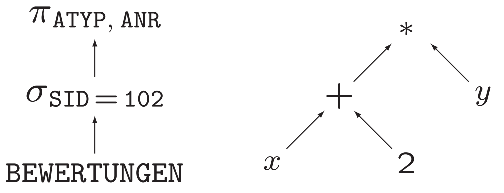
$$\pi_{\texttt{ATYP, ANR}}(\sigma_{\texttt{SID}=102}(\texttt{BEWERTUNGEN}))$$

- Man kann ein Zwischenergebnis in einer temporären Relation speichern (andere Sicht: Makro-Definition).

$$\texttt{S102} := \sigma_{\texttt{SID}=102}(\texttt{BEWERTUNGEN});$$
$$\pi_{\texttt{ATYP, ANR}}(\texttt{S102})$$

Operationen schachteln (2)

- Ausdrücke der relationalen Algebra können klarer werden, wenn man sie in Operatorbäumen darstellt.



- Zum Vergleich ist rechts der Operatorbaum des arithmetischen Ausdrucks $(x+2)*y$ dargestellt.

Zwischenergebnisse fließen entlang der Linien von unten nach oben.

Basis-Operanden

- Die Blätter eines Operatorbaumes sind
  - Namen von Datenbankrelationen
  - konstante Relationen (explizite Tupelaufzählung).
- Ein Relationsname R ist ein legaler Ausdruck der relationalen Algebra. Sein Wert ist die gesamte, unter diesem Namen gespeicherte, Relation.

  Die relationale Algebra fordert nicht, daß jede Anfrage eine Projektion enthalten muß. Eine Projektion auf alle Attribute ist überflüssig.
- Eine konstante Relation mit einem Tupel wird in folgener Form geschrieben: $\{(A_1{:}\, c_1, \ldots, A_n{:}\, c_n)\}$.

  Für mehrere Tupel verwende man $\cup$. Die c_i sind Konstanten.

Übungen (1)

Schreiben Sie folgende Anfrage in relationaler Algebra:

- Geben Sie die E-Mail-Adresse von Lisa Weiss aus.

  Schreiben Sie die Anfrage als Baum und verschachtelt mit Klammern.

Dies bezieht sich auf das Schema auf Folie 4-4:

- `STUDENTEN(`<u>`SID`</u>`, VORNAME, NACHNAME, EMAIL`^{o}`)`
- `AUFGABEN(`<u>`ATYP`</u>`, `<u>`ANR`</u>`, THEMA, MAXPT)`
- `BEWERTUNGEN(`<u>`SID`</u>`→STUDENTEN, (`<u>`ATYP`</u>`, `<u>`ANR`</u>`)→AUFGABEN, PUNKTE)`

Übungen (2)

- Welche der folgenden Ausdrücke der relationalen Algebra sind syntaktisch korrekt?
  Was bedeuten sie?
  - ☐ STUDENTEN.
  - ☐ $\sigma_{\texttt{MAXPT} \neq 10}(\texttt{AUFGABEN})$.
  - ☐ $\pi_{\texttt{VORNAME}}(\pi_{\texttt{NACHNAME}}(\texttt{STUDENTEN}))$.
  - ☐ $\sigma_{\texttt{PUNKTE} \leq 5}(\sigma_{\texttt{PUNKTE} \geq 1}(\texttt{BEWERTUNGEN}))$.
  - ☐ $\sigma_{\texttt{PUNKTE}}(\pi_{\texttt{PUNKTE} = 10}(\texttt{BEWERTUNGEN}))$.

Inhalt

Kartesisches Produkt (1)

- Häufig müssen Ausgabetupel berechnet werden, die aus mehreren Eingabetupeln abgeleitet sind.

  Für σ und π gilt, daß jedes Ausgabetupel von einem einzigen Eingabetupel abgeleitet ist. Da π Duplikate eliminiert, ist es möglich, daß das gleiche Ausgabetupel von zwei verschiedenen Eingabetupeln abgeleitet wird, aber dann ist eins der beiden überflüssig.
- Dies leistet das "kartesische Produkt" $\times$.

  Vgl. "kartesische Koordinaten". Man nennt es auch "Kreuzprodukt".
- $R \times S$ verbindet ("klebt zusammen") jedes Tupel von R mit jedem Tupel von S.

  In ASCII schreibt man "`R PRODUCT S`" oder "`R x S`" für $R \times S$. Falls nötig, setzt man $(\ldots)$ um die Eingaberelationen.

Kartesisches Produkt (2)

- Beispiel:

A	B
1	2
3	4

$\times$

C	D
6	7
8	9

$=$

A	B	C	D
1	2	6	7
1	2	8	9
3	4	6	7
3	4	8	9

- Da Attributnamen innerhalb eines Tupels eindeutig sein müssen, kann man das kartesische Produkt nur anwenden, wenn R und S keine gemeinsamen Attribute haben.
- Das ist keine echte Einschränkung. Man kann die Attribute notfalls erst umbenennen (π).

Kartesisches Produkt (3)

- Ist $t = (A_1\colon a_1, \ldots, A_n\colon a_n)$, $u = (B_1\colon b_1, \ldots, B_m\colon b_m)$, so sei $t \circ u = (A_1\colon a_1, \ldots, A_n\colon a_n, B_1\colon b_1, \ldots, B_m\colon b_m)$.
- Das kartesische Produkt $R \times S$ kann mit zwei verschachtelten Schleifen berechnet werden:

```
Create new temporary relation T;
foreach tuple t in R do
    foreach tuple u in S do
        insert t ∘ u into T;
    od;
od;
return T;
```


Kartesisches Produkt (4)

- Die Relation R enthalte n Tupel, und die Relation S enthalte m Tupel, dann enthält $R \times S$ $n * m$ Tupel.
- Das kartesische Produkt allein ist selten sinnvoll, da es zu einem "Aufblähen" der Relationsgröße führt.
- Das Problem ist, daß $R \times S$ jedes Tupel von R mit jedem Tupel von S verbindet. Normalerweise will man aber nur bestimmte Paare von Tupeln verknüpfen.
- Somit ist das kartesische Produkt nur als Eingabe für eine folgende Selektion sinnvoll (→ Verbund).

Kartesisches Produkt (5)

- Einige Autoren definieren $\times$ so, daß doppelte Attribute automatisch umbenannt werden:
  - ◇ Z.B. für Relationen $R(A, B)$ und $S(B, C)$ hat das Produkt $R \times S$ die Attribute $(R.A, R.B, S.B, S.C)$.
  - ◇ Sie erlauben außerdem, den Präfix wegzulassen, wenn der Rest eindeutig ist, z.B. A und C.
- **In dieser Vorlesung ist das nicht gestattet!**

  Die formale Definition ist einfacher: Was passiert z.B. mit $((R \cup S) \times T)$ und mit $R \times R$? Normalerweise geben Autoren, die solche Namen zulassen, keine formale Definition. In dieser Vorlesung kann man den Umbenennungsoperator (siehe unten) verwenden, um Attributnamen der Form $R.A$ einzuführen, aber dann kann A allein nicht verwendet werden: Jedes Attribut hat genau einen Namen.

Umbenennung

- Ein Operator $\rho_R(S)$, der "$R.$" vor alle Attributnamen stellt, ist manchmal nützlich:

$$\rho_R\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 2 \\ 3 & 4 \\ \hline \end{array}\right) = \begin{array}{|r|r|}\hline R.A & R.B \\ \hline 1 & 2 \\ 3 & 4 \\ \hline \end{array}$$

- Dies ist nur eine Abkürzung für eine Anwendung der Projektion: $\pi_{R.A \leftarrow A,\, R.B \leftarrow B}(S)$.
- Sonst enthalten Attributnamen in der relationalen Algebra nicht automatisch den Relationsnamen.

  Eine explizite, manuelle Umbenennung mit dem Operator $\rho_R(S)$ ist einfach zu formalisieren, eine automatische Umbennung nicht.

Verbund/Join (1)

- Da die Kombination von kartesischem Produkt und Selektion so verbreitet ist, wurde dafür ein spezielles Symbol eingeführt:

  $R \underset{A=B}{\bowtie} S$ ist eine Abkürzung für $\sigma_{A=B}(R \times S)$.

- Diese Operation nennt man "Verbund" (Join): Sie wird verwendet, um zwei Tabellen (d.h. ihre Tupel) zu verbinden.

  In ASCII schreibt man z.B. "`R JOIN[A=B] S`".

- Der Verbund ist eine der wichtigsten und nützlichsten Operationen der relationalen Algebra.

  Gleich nach der Selektion.

Verbund/Join (2)

STUDENTEN ⋈ BEWERTUNGEN						
SID	VORNAME	NACHNAME	EMAIL	ATYP	ANR	PUNKTE
101	Lisa	Weiss	···	H	1	10
101	Lisa	Weiss	···	H	2	8
101	Lisa	Weiss	···	Z	1	12
102	Michael	Grau	NULL	H	1	9
102	Michael	Grau	NULL	H	2	9
102	Michael	Grau	NULL	Z	1	10
103	Daniel	Sommer	···	H	1	5
103	Daniel	Sommer	···	Z	1	7

- Die Studentin Iris Winter taucht nicht auf, da sie keine Hausaufgabe abgegeben und nicht an einer Klausur teilgenommen hat.
- Oben ist der natürliche Verbund der beiden Tabellen gezeigt. Im folgenden wird aber zunächst der Standard-Verbund erklärt.

Verbund/Join (3)

- $R \underset{A=B}{\bowtie} S$ kann ausgewertet werden wie $\sigma_{A=B}(R \times S)$:

```
Create new temporary relation T;
foreach tuple t in R do
    foreach tuple u in S do
        if t.A = u.B then
            insert t ∘ u into T;
        fi;
    od;
od;
return T;
```


Verbund/Join (4)

- Die obige Prozedur nennt man "nested loop join".
- Man beachte, daß das Zwischenergebnis von $R \times S$ nicht explizit gespeichert wird.

  Tatsächlich versucht ein DBMS auch sonst, keine Zwischenergebnisse zu materialisieren (soweit möglich). Jeder Algebraoperator berechnet nur jeweils ein Tupel ("Pipeline-Auswertung"). Dadurch ist der Nested Loop Join ohnehin das gleiche wie $\times$ gefolgt von σ.

- Es wurden viele verschiedene Algorithmen zur Berechnung des Verbunds entwickelt.

  Weil der Verbund eine wichtige und relativ teure Operation ist. Z.B. "merge join", "index join", "hash join". Siehe Vorlesung "DB II B".

Verbund/Join (5)

- Die Verbundbedingung muß nicht die Form $A = B$ haben (obwohl dies am häufigsten vorkommt). Es kann eine beliebige Bedingung sein, z.B. auch $A < B$.

  Ein Verbund mit einer Bedingung der Form $A = B$ (oder $A_1 = B_1 \wedge \cdots \wedge A_n = B_n$) nennt man "equijoin".

- Eine typische Anwendung ist es, Tupel entsprechend einem Fremdschlüssel zu verknüpfen, z.B.

$$\mathtt{BEWERTUNGEN} \underset{\mathtt{SID=SID'}}{\bowtie} \pi_{\mathtt{SID'} \leftarrow \mathtt{SID},\, \mathtt{NACHNAME},\, \mathtt{EMAIL}}(\mathtt{STUDENTEN})$$

  Die Umbenennung des Attributs "SID" ist notwendig, da das kartesische Produkt disjunkte Attributnamen verlangt. Aber der unten erklärte natürliche Verbund macht dies wieder überflüssig.

Verbund/Join (6)

- Der Verbund kombiniert Tupel und agiert als Filter: Er eliminiert Tupel ohne Verbundpartner. (Fremdschlüssel sichern Existenz von Verbundpartnern!)

A	B
1	2
3	4

$\underset{B=C}{\bowtie}$

C	D
4	5
6	7

$=$

A	B	C	D
3	4	4	5

- Ein "Semijoin" ($\ltimes$, $\rtimes$) agiert nur als Filter.

  Er entspricht einem Verbund und anschließender Projektion auf die Attribute der linken (linker SJ) oder rechten Relation (rechter SJ).

- Ein "äußerer Verbund" (vgl. Ende dieses Kapitels) agiert nicht als Filter: Er erhält alle Eingabetupel.

Natürlicher Verbund (1)

- Eine weitere nützliche Abkürzung ist der "natürliche Verbund" (natural join) $\bowtie$.

  Anstelle dieses Symbols kann man auch "*" verwenden.

- Er verknüpft Tupel, die die gleichen Werte bei Attributen mit gleichem Namen haben.

  Während kartesisches Produkt und allgemeiner Verbund verlangen, daß die Attribute beider Relationen verschiedene Namen haben, verwendet der natürliche Verbund die gleichen Namen für die Bedingung.

A	B
1	2
3	4

$\bowtie$

B	C
4	5
4	8
6	7

$=$

A	B	C
3	4	5
3	4	8

Natürlicher Verbund (2)

- Der natürliche Verbund der beiden Relationen
  - ◇ $R(A_1, \ldots, A_n, B_1, \ldots, B_k)$ und
  - ◇ $S(B_1, \ldots, B_k, C_1, \ldots, C_m)$

  produziert im DB-Zustand $\mathcal{I}$ alle Tupel der Form

$$(a_1, \ldots, a_n, b_1, \ldots, b_k, c_1, \ldots, c_m),$$

  wobei
  - ◇ $(a_1, \ldots, a_n, b_1, \ldots, b_k) \in \mathcal{I}[R]$ und
  - ◇ $(b_1, \ldots, b_k, c_1, \ldots, c_m) \in \mathcal{I}[S]$.

Natürlicher Verbund (3)

- Der natürliche Verbund entspricht nicht nur dem kartesischen Produkt gefolgt von einer Selektion, sondern
  - ◇ benennt jeweils eine Kopie der gemeinsamen Attribute vor dem kartesischen Produkt um, und
  - ◇ verwendet eine Projektion, um diese doppelten Attribute am Schluß zu eliminieren.
- Seien z.B. $R(A, B)$ und $S(B, C)$ gegeben. Dann ist $R \bowtie S$ eine Abkürzung für

$$\pi_{A,B,C}(\sigma_{B=B'}(R \times \pi_{B' \leftarrow B,\, C}(S))).$$

Bemerkung zum DB-Entwurf

- Zur Unterstützung des natürlichen Verbunds sollte man Attributen zweier Relationen, die typischerweise verbunden werden, den gleichen Namen geben.
- Auch wenn die Anfragesprache keinen natürlichen Verbund beinhaltet, ist das gute Dokumentation.

  Man kann beim Datenbankentwurf anwendungsspezifische Datentypen einführen (sogenannte Domains, z.B. `KundenNr` statt `int`). Wenn man die Namen dieser Domains als Spalten-/Attributnamen verwendet, passt der natürliche Verbund meist automatisch.
- Umgekehrt sollte man Attribute, die nicht sinnvoll mit einem Verbund verknüpft (d.h. verglichen) werden können, auch unterschiedlich nennen.

Algebraische Gesetze (1)

- Definition: Zwei RA-Ausdrücke Q_1 und Q_2 heißen äquivalent genau dann, wenn ihr Ergebnisschema gleich ist, und für alle DB-Zustände $\mathcal{I}$ gilt:

$$\mathcal{I}[Q_1] = \mathcal{I}[Q_2].$$

  Man schreibt dann $Q_1 = Q_2$.

  Es gibt tatsächlich zwei Begriffe von Äquivalenz, abhängig davon, ob man alle strukturell möglichen Zustände betrachtet, oder nur die, die den Integritätsbedingungen genügen. Die erste Alternative ist eine stärkere Forderung. Bei der zweiten Alternative erhält man mehr äquivalente Anfragen. Die folgenden Aussagen gelten in beiden Fällen.
- Äquivalente Anfragen liefern also immer das gleiche Ergebnis.

Algebraische Gesetze (2)

- Der Verbund erfüllt das Assoziativitätsgesetz:

$$(R \bowtie S) \bowtie T \;=\; R \bowtie (S \bowtie T).$$

- Somit werden die Klammern nicht benötigt:

$$R \bowtie S \bowtie T.$$

- Der Verbund ist nicht ganz kommutativ: Die Reihenfolge der Spalten ist unterschiedlich.

  Fasst man Tupel als Variablenbelegung auf, gibt es keine Reihenfolge.
- Falls jedoch danach eine Projektion folgt, ist das egal (man kann π auch für diesen Zweck einführen):

$$\pi_{\ldots}(R \bowtie S) \;=\; \pi_{\ldots}(S \bowtie R).$$

Algebraische Gesetze (3)

- Vom Anfrageoptimierer eines relationalen DBMS werden weitere algebraische Gesetze verwendet.
- Z.B. wenn sich φ nur auf S bezieht, dann gilt

  $$\sigma_\varphi(R \bowtie S) \;=\; R \bowtie \sigma_\varphi(S).$$

  Die rechte Seite kann meist effizienter ausgewertet werden (abhängig von Relationsgröße, Indexen).
- Für diese Vorlesung ist Effizienz nicht so wichtig.

  Der Anfrageoptimierer formt eine gegebene Anfrage automatisch in eine effizientere Variante um, so daß sich der Nutzer darum nicht kümmern muß. Für eine Lösung, die immer das richtige Ergebnis liefert, und die nicht unnötig kompliziert ist (z.B. π auf alle Spalten), wird es die volle Punktzahl geben.

Algebraische Gesetze (4)

- Obige Äquivalenzen gelten auch für das kartesische Produkt anstelle des Verbundes:
  - ◇ $(Q_1 \times Q_2) \times Q_3 \;=\; Q_1 \times (Q_2 \times Q_3)$.
  - ◇ Ist A ein Attribut im Schema von Q_1, dann ist $\sigma_{A=d}(Q_1 \times Q_2) \;=\; (\sigma_{A=d}(Q_1)) \times Q_2$.
- Die Reihenfolge aufeinander folgender Selektionen ist egal: $\sigma_{\varphi_1}(\sigma_{\varphi_2}(Q)) \;=\; \sigma_{\varphi_2}(\sigma_{\varphi_1}(Q))$.
- Zwei Projektionen direkt nacheinander sind überflüssig: $\pi_{A_1,\ldots,A_n}(\pi_{B_1,\ldots,B_m}(Q)) \;=\; \pi_{A_1,\ldots,A_n}(Q)$.

Algebraische Gesetze (5)

- Satz: Die Äquivalenz von RA-Ausdrücken ist unentscheidbar.

  D.h. man kann kein Programm schreiben, das zwei beliebige RA-Ausdrücke Q_1 und Q_2 einliest, und "ja" oder "nein" ausgibt, abhängig davon, ob Q_1 und Q_2 äquivalent sind, und das garantiert nach endlicher Rechenzeit anhält. Für spezielle Paare von Q_1 und Q_2 ist es natürlich häufig möglich, festzustellen, ob sie äquivalent sind, oder nicht. Es wird aber immer Fälle geben, die ein Programm nicht behandeln kann.

  Dies ist auch eine Aussage über die Mächtigkeit der Relationenalgebra.

  Ein Anfrageoptimierer verwendet die bekannten Äquivalenzen (u.a. Umordnung von Verbunden, Aufspalten und Verschieben von Selektionen), um eine Reihe von äquivalenten Anfragen zu erzeugen, und schätzt dann die "Kosten" (Ausführungsdauer). Er kann aber niemals alle äquivalenten Formulierungen erzeugen, das Ergebnis muß also nicht optimal sein.

Häufiges Anfragemuster (1)

- Folgende Anfragestruktur ist sehr häufig:

  $$\pi_{A_1,\ldots,A_k}(\sigma_\varphi(R_1 \bowtie \cdots \bowtie R_n)).$$

  - ◇ Erst verknüpft man alle Tabellen, die für die Anfrage benötigt werden, mit einem Verbund.
  - ◇ Dann selektiert man die relevanten Tupel.

    Die Selektionsbedingung kann sich auf die Attribute von allen Tabellen beziehen, die im ersten Schritt verknüpft wurden.
  - ◇ Als drittes projiziert man auf die Attribute, die ausgeben werden sollen.

Häufiges Anfragemuster (2)

- Muster sind oft nützlich.

  Damit können typische Anfragen schnell formuliert werden.

- Aber die Operationen der relationalen Algebra können auf jede Weise kombiniert werden. Man muß sich nicht notwendig an dieses Muster halten.

  Z.B. wenn keine Projektion oder Selektion benötigt wird, wäre es falsch, die Anfrage zu verkomplizieren, indem man eine Projektion auf alle Attribute oder eine Selektion mit einer stets wahren Bedingung verwendet (beides entspricht einfach einer Identitätsabbildung).

- Dagegen sind in SQL die Schlüsselworte `SELECT` und `FROM` Pflicht, und die Reihenfolge muß immer sein:

  `SELECT ... FROM ... WHERE ...`

Häufiges Anfragemuster (3)

- $\pi_{A_1,\ldots,A_m}(\sigma_\varphi(R_1 \bowtie \cdots \bowtie R_n))$ schreibt man im Tupelkalkül:

  $$\{\overline{A_1},\ldots,\overline{A_m}\ [R_1\ X_1,\ldots,R_n\ X_n] \mid \overline{\varphi} \wedge G\}$$

  wobei

  ◇ $\overline{A_i} := X_j.A_i$, falls R_j das Attribut A_i hat.

    Gibt es mehrere R_j mit Attribut A_i, so kann ein beliebiges gewählt werden: Aufgrund der Join-Bedingung G sind alle $X_j.A_i$ gleich.

  ◇ $\overline{\varphi}$ entsprechend aus φ hervorgeht, indem den Attributnamen Tupelvariablen hinzugefügt werden.

  ◇ G ist die Joinbedingung, siehe nächste Folie.

Häufiges Anfragemuster (4)

- In obiger Anfrage ist die Verbund-Bedingung G die Konjunktion über $X_j.B = X_k.B$ für alle R_j, R_k, B, so daß R_j und R_k das gemeinsame Attribut B haben.

  Natürlich kann man durch die Transitivität von "=" implizierte Bedingungen weglassen. Wenn z.B. R_1, R_2, R_3 ein gemeinsames Attribut B haben, reicht $X_1.B = X_2.B \wedge X_2.B = X_3.B$.

- Es ist ein häufiger Fehler, eine Verbundbedingung zu vergessen.

  Dann erhält man ein kartesisches Produkt, welches zu falschen Antworten und häufig zu sehr großen Anfrageergebnissen führt.

Häufiges Anfragemuster (5)

- $\pi_{A_1,\ldots,A_m}(\sigma_\varphi(R_1 \bowtie \cdots \bowtie R_n))$ kann in den Bereichskalkül übersetzt werden, indem man für jedes Attribut eine Variable gleichen Namens einführt:

  $$\{A_1,\ldots,A_m\ [s_1\ B_1,\ldots,s_k\ B_k] \mid \varphi \wedge G\}$$

  wobei

  ◇ $B_1,\ldots,B_k$ alle in $R_1,\ldots,R_n$ vorkommenden Attribute sind, und B_i jeweils die Sorte s_i hat.

  ◇ G die Konjunktion der Formeln $R_i(B_{j_1},\ldots,B_{j_l})$ für $i = 1,\ldots,n$ ist, dabei seien $B_{j_1},\ldots,B_{j_l}$ die Attribute von R_i.

Häufiges Anfragemuster (6)

- Um eine Anfrage zu formulieren, sollte man zunächst über die benötigten Tabellen nachdenken:
  - ◇ Normalerweise nennt die in natürlicher Sprache gestellte Anfrage gewisse Attribute.

    Es ist auch möglich, daß Datenwerte genannt werden, die gewissen Attributen entsprechen (z.B. Studentennamen).
  - ◇ Jedes solche Attribut benötigt zumindest eine Relation, die dieses Attribut enthält.

    So daß das Attribut in der Selektionsbedingung oder der Projektionsliste verwendet werden kann.

Häufiges Anfragemuster (7)

- Anfrageformulierung, fortgesetzt:
  - ◇ Schließlich benötigt man manchmal Zwischenrelationen, um einen Verbund sinnvoll zu machen.
  - ◇ Z.B. seien $R(A, B)$, $S(B, C)$, $T(C, D)$ gegeben und die Attribute A und D werden gebraucht. Dann wäre $R \bowtie T$ nicht korrekt. Warum?
  - ◇ Korrekt ist hier der Verbund $R \bowtie S \bowtie T$.
  - ◇ Eine Zeichnung der Fremdschlüsselbeziehungen zwischen den Tabellen ist oft hilfreich (entspricht typischen Verknüpfungen mittels Verbund).

Übung

Schreiben Sie folgende Anfragen in relationaler Algebra:

- Geben Sie alle Hausaufgabenergebnisse von Lisa Weiss aus (Übungsnummer und Punkte).
- Wer hat auf eine Hausaufgabe volle Punktzahl (Vorname, Nachname und Aufgabennummer)?

Dies bezieht sich auf das Schema auf Folie 4-4:

- STUDENTEN(<u>SID</u>, VORNAME, NACHNAME, EMAIL^o)
- AUFGABEN(<u>ATYP</u>, <u>ANR</u>, THEMA, MAXPT)
- BEWERTUNGEN(<u>SID</u>→STUDENTEN, (<u>ATYP</u>, <u>ANR</u>)→AUFGABEN, PUNKTE)

Selbstverbund (1)

- Manchmal ist es notwendig, sich auf mehr als ein Tupel einer Relation gleichzeitig zu beziehen.
- Z.B. Wer hat auf irgendeine Übung mindestens so viele Punkte wie die Studentin 101?
- In diesem Fall benötigt man zwei Tupel der Relation BEWERTUNGEN, um ein Ergebnistupel zu berechnen:
  - ◇ Ein Tupel für die Studentin 101.
  - ◇ Ein Tupel der gleichen Übung, bei dem PUNKTE mindestens so groß wie in dem ersten Tupel ist.

Selbstverbund (2)

- Dies erfordert eine Verallgemeinerung des obigen Anfragemusters, in dem zwei Kopien einer Relation verknüpft werden (mindestens eins muß vorher umbenannt werden).

  $$S := \rho_{\mathtt{X}}(\mathtt{BEWERTUNGEN}) \underset{\substack{\mathtt{X.ATYP} = \mathtt{Y.ATYP} \\ \wedge\, \mathtt{X.ANR} = \mathtt{Y.ANR}}}{\bowtie} \rho_{\mathtt{Y}}(\mathtt{BEWERTUNGEN});$$

  $$\pi_{\mathtt{X.SID}}(\sigma_{\mathtt{X.PUNKTE} \geq \mathtt{Y.PUNKTE} \,\wedge\, \mathtt{Y.SID}=\mathtt{101}}(S))$$

- Diese Verbunde einer Tabelle mit sich selbst werden manchmal "Selbstverbund" (self join) genannt.

Inhalt

Mengenoperationen (1)

- Da Relationen Mengen (von Tupeln) sind, können auch die gewöhnlichen Mengenoperationen $\cup$, $\cap$, $-$ auf Relationen angewandt werden.
- Hierbei müssen beide Eingaberelationen das gleiche Schema haben.

  Z.B. ist es nicht möglich, die zwei Relationen $R(A)$ und $S(B, C)$ zu vereinigen, da es kein gemeinsames Schema für die Ausgaberelation gibt.

- $R \cup S$ enthält alle Tupel, die in R, in S, oder in beiden Relationen enthalten sind (Vereinigung).

Mengenoperationen (2)

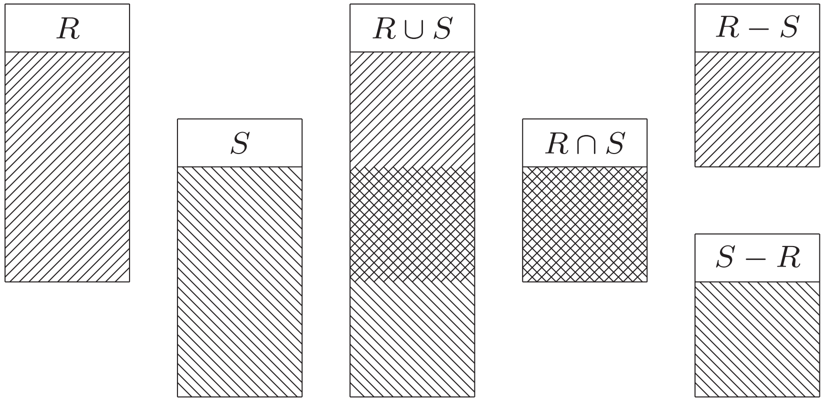
- $R - S$ enthält alle Tupel, die in R, aber nicht in S sind (Mengendifferenz/Set Difference).
- $R \cap S$ enthält Tupel, die in beiden, R und S, sind (Durchschnitt/Intersection).
- Der Durchschnitt ist (wie der Verbund) eine abgeleitete Operation: Er kann durch $-$ ausgedrückt werden:

  $$R \cap S = R - (R - S).$$

- Übung: Beweisen Sie diese Gleichung.

  Z.B. Zeichnen Sie ein Venn Diagramm.

Mengenoperationen (3)



Mengenoperationen (4)

- $R \cup S$ kann wie folgt implementiert werden:

```
Create new temporary relation T;
foreach tuple t in R do
    insert t into T;
od;
foreach tuple t in S do
    insert t into T;
od;
return T;
```

- **insert** muß dabei eventuell Duplikate eliminieren.

  In SQL gibt es `UNION` (mit Duplikatelimination) und `UNION ALL` (ohne Duplikatelimination, läuft schneller).

Mengenoperationen (5)

- $R - S$ kann wie folgt implementiert werden:

```
Create new temporary relation T;
foreach tuple t in R do
    Remove := false;
    foreach tuple u in S do
        if u = t then
            Remove := true;
    od;
    if not Remove then
        insert t into T;
od;
return T;
```


Vereinigung (1)

- Ohne $\cup$ kann jede Ausgabespalte nur Werte einer einzigen Spalte der gespeicherten Tabellen enthalten.

  Oder eine einzelne Konstante. Wenn Datentypoperationen in der Projektion erlaubt sind, kann der Ergebniswert mit einer einzelnen Formel aus verschiedenen Eingabespalten berechnet werden, aber selbst dies ist noch kein "Union"-Verhalten.

- Beispiel: Neben registrierten Studenten, die Hausaufgaben abgeben und Prüfungen machen, gibt es auch Gasthörer, die sich nur die Vorlesung anhören:

  GASTHÖRER(<u>VORNAME</u>, <u>NACHNAME</u>, EMAIL^o).

- Aufgabe: Erstellung einer Liste der Email-Adressen von Studenten und Gasthörern in einer Anfrage.

Vereinigung (2)

- Mit $\cup$ ist dies einfach:

$$\pi_{\texttt{EMAIL}}(\texttt{STUDENTEN}) \cup \pi_{\texttt{EMAIL}}(\texttt{GASTHÖRER}).$$

- Diese Anfrage kann nicht ohne $\cup$ formuliert werden.
- Eine weitere typische Anwendung von $\cup$ ist die Fallunterscheidung:

$$\texttt{ZPT} := \pi_{\texttt{SID,PUNKTE}}(\sigma_{\texttt{ATYP='Z'} \wedge \texttt{ANR=1}}(\texttt{BEWERTUNGEN}));$$

$$\begin{array}{l} \pi_{\texttt{SID, NOTE}\leftarrow\texttt{'1'}}(\sigma_{\texttt{PUNKTE} \geq 12}(\texttt{ZPT})) \\ \cup\, \pi_{\texttt{SID, NOTE}\leftarrow\texttt{'2'}}(\sigma_{\texttt{PUNKTE} \geq 10 \wedge \texttt{PUNKTE} < 12}(\texttt{ZPT})) \\ \cup\, \pi_{\texttt{SID, NOTE}\leftarrow\texttt{'3'}}(\sigma_{\texttt{PUNKTE} \geq 7 \wedge \texttt{PUNKTE} < 10}(\texttt{ZPT})) \\ \cup\, \pi_{\texttt{SID, NOTE}\leftarrow\texttt{'5'}}(\sigma_{\texttt{PUNKTE} < 7}(\texttt{ZPT})) \end{array}$$

Mengendifferenz (1)

- Die Operatoren σ, π, $\times$, $\bowtie$, $\cup$ haben ein monotones Verhalten, z.B.

$$R \subseteq S \implies \sigma_\varphi(R) \subseteq \sigma_\varphi(S)$$

- Damit folgt, daß sich auch jede Anfrage Q, die obige Operatoren verwendet, monoton verhält:
  - ◇ Sei $\mathcal{I}_1$ ein DB-Zustand, und resultiere $\mathcal{I}_2$ aus $\mathcal{I}_1$ durch Einfügen von einem oder mehreren Tupeln.
  - ◇ Dann ist jedes als Antwort auf Q in $\mathcal{I}_1$ enthaltene Tupel t, auch als Antwort auf Q in $\mathcal{I}_2$ enthalten.

    D.h. korrekte Antworten bleiben auch nach Einfügungen gültig.

Mengendifferenz (2)

- Definition: Ein DB-Zustand $\mathcal{I}_1$ ist ist kleinergleich einem DB-Zustand $\mathcal{I}_2$, geschrieben $\mathcal{I}_1 \subseteq \mathcal{I}_2$, gdw. $\mathcal{I}_1(R) \subseteq \mathcal{I}_2(R)$ für alle Relationen R im Schema.
- Definition: Eine Anfrage Q heißt monoton gdw. für alle DB-Zustände $\mathcal{I}_1, \mathcal{I}_2$ gilt:

$$\mathcal{I}_1 \subseteq \mathcal{I}_2 \implies \mathcal{I}_1[Q] \subseteq \mathcal{I}_2[Q].$$

- Satz: Kommt der Mengendifferenz-Operator $-$ in einem RA-Ausdruck Q nicht vor, so ist Q monoton.

  Übung: Beweisen Sie den Satz durch Induktion über die Struktur von Q ("strukturelle Induktion").

Mengendifferenz (3)

- Wenn sich die gewünschte Anfrage nichtmonoton verhält, so folgt, daß man die Mengendifferenz "−" verwenden muß. Beispiele solcher Anfragen:
  - ◇ Welcher Student hat noch keine Übung gelöst?
  - ◇ Wer hat die meisten Punkte auf Hausaufgabe 1?
  - ◇ Wer hat alle Übungen in der Datenbank gelöst?
- Übung: Geben Sie für jede dieser Fragen ein Antworttupel aus dem Beispielzustand an (vgl. nächste Folie) und für jede solche Antwort ein Tupel, das durch Einfügung diese Antwort ungültig macht.

Mengendifferenz (4)

STUDENTEN			
<u>SID</u>	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

AUFGABEN			
<u>ATYP</u>	<u>ANR</u>	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

BEWERTUNGEN			
<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

Mengendifferenz (5)

- Z.B. welcher Student hat noch keine Übung gelöst?

  $\texttt{KEINE\_LÖS} := \pi_{\texttt{SID}}(\texttt{STUDENTEN}) - \pi_{\texttt{SID}}(\texttt{BEWERTUNGEN});$

  $\pi_{\texttt{VORNAME, NACHNAME}}(\texttt{STUDENTEN} \bowtie \texttt{KEINE\_LÖS})$

- Übung: Wo ist der Fehler in dieser Anfrage?

  $\pi_{\texttt{SID, VORNAME, NACHNAME}}(\texttt{STUDENTEN}) - \pi_{\texttt{SID}}(\texttt{BEWERTUNGEN})$

- Ist dies die richtige Lösung?

  $\pi_{\texttt{NACHNAME}}(\texttt{STUDENTEN} \underset{\texttt{SID}\neq\texttt{SID2}}{\bowtie} \pi_{\texttt{SID2}\leftarrow\texttt{SID}}(\texttt{BEWERTUNGEN}))$

Mengendifferenz (6)

- Wenn man $-$ verwendet, ist der Anti-Join ein typisches Muster.
- Z.B. seien $R(A, B)$ und $S(B, C)$ gegeben, so können die Tupel von R, die keinen Verbundpartner in S haben, wie folgt berechnet werden:

  $$R \bowtie (\pi_B(R) - \pi_B(S)).$$

- Folgendes ist äquivalent: $R - \pi_{A,B}(R \bowtie S)$.

  Man muß beachten, daß die Mengendifferenz gleiche Schemata auf beiden Seiten erfordert. Dazu benötigt man Projektion und Verbund.

- Es ist kein Symbol für den Anti-Join verbreitet, man könnte $R \,\overline{\ltimes}\, S$ verwenden.

Mengendifferenz (7)

- Man beachte, daß zur Anwendung der Mengendifferenz $R - S$ nicht (!) $S \subseteq R$ gelten muß.

  In einer Klausur enthielten viele Lösungen unnötige Komplikationen, die auf dieses Missverständnis zurückzuführen sein könnten.

- Z.B. berechnet diese Anfrage die SIDs der Studenten, die HA 2, aber nicht HA 1 gelöst haben:

  $$\begin{array}{rl} & \pi_{\texttt{SID}}(\sigma_{\texttt{ATYP='H'} \wedge \texttt{ANR=2}}(\texttt{BEWERTUNGEN})) \\ - & \pi_{\texttt{SID}}(\sigma_{\texttt{ATYP='H'} \wedge \texttt{ANR=1}}(\texttt{BEWERTUNGEN})) \end{array}$$

- Es ist hierbei kein Problem, daß es auch Studenten geben kann, die Hausaufgabe 1, aber nicht Hausaufgabe 2 gelöst haben.

Übungen

Schreiben Sie folgende Anfragen in relationaler Algebra:

- Wer hat die meisten Punkte für Hausaufgabe 1?

  Hinweis: Berechnen Sie zunächst die Studenten, die nicht die meisten Punkte haben, d.h. für die es einen Studenten mit mehr Punkten gibt. Verwenden Sie dann die Mengendifferenz.

- Wer hat alle Übungen der DB gelöst?

Dies bezieht sich auf das Schema auf Folie 4-4:

- STUDENTEN(<u>SID</u>, VORNAME, NACHNAME, EMAIL^{o})
- AUFGABEN(<u>ATYP</u>, <u>ANR</u>, THEMA, MAXPT)
- BEWERTUNGEN(<u>SID</u>→STUDENTEN, (<u>ATYP</u>, <u>ANR</u>)→AUFGABEN, PUNKTE)

Vereinigung vs. Verbund

- Zwei alternative Darstellungen der Punkte für HA und Zwischen- und Endklausur der Studenten sind:

Resultate_1

STUDENT	H	Z	E
Jim Ford	95	60	75
Ann Lloyd	80	90	95

Resultate_2

STUDENT	ATYP	PROZENT
Jim Ford	H	95
Jim Ford	Z	60
Jim Ford	E	75
Ann Lloyd	H	80
Ann Lloyd	Z	90
Ann Lloyd	E	95

- Geben Sie Algebraausdrücke an, um die Tabellen ineinander zu überführen.

Zusammenfassung

Die fünf Basisoperationen der relationalen Algebra sind:

- σ_φ: Selektion
- $\pi_{A_1,\ldots,A_k}$: Projektion
- $\times$: Kartesisches Produkt
- $\cup$: Vereinigung
- $-$: Mengendifferenz

Abgeleitete Operationen sind:
Der allgemeine Verbund $\underset{\varphi}{\bowtie}$, der natürliche Verbund $\bowtie$, der Umbenennungsoperator ρ, der Durchschnitt $\cap$.

Inhalt

Äußerer Verbund (1)

- Der gewöhnliche Verbund (“innerer Verbund”) eliminiert Tupel ohne Verbundpartner:

A	B
a_1	b_1
a_2	b_2

$\bowtie$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_2	b_2	c_2

- Der linke äußere Verbund stellt sicher, daß Tupel der linken Tabelle auch im Ergebnis vorhanden sind:

A	B
a_1	b_1
a_2	b_2

$⟕$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_1	b_1	
a_2	b_2	c_2

Zeilen der linken Seite werden, falls notwendig, mit “Null” aufgefüllt.

Äußerer Verbund (2)

- Der rechte äußere Verbund erhält die Tupel der rechten Tabelle:

A	B
a_1	b_1
a_2	b_2

$⟖$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_2	b_2	c_2
	b_3	c_3

- Der volle äußere Verbund eliminiert gar kein Tupel:

A	B
a_1	b_1
a_2	b_2

$⟗$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_1	b_1	
a_2	b_2	c_2
	b_3	c_3

Äußerer Verbund (3)

$R \underset{A=B}{⟕} S$:

```
Create new temporary relation T;
foreach tuple t in R do
    HasJoinPartner := false;
    foreach tuple u in S do
        if t.A = u.B then
            insert t ∘ u into T;
            HasJoinPartner := true;
        fi;
    od;
    if not HasJoinPartner then
        insert t ∘ (null, ..., null) into T;
od;
return T;
```


Äußerer Verbund (4)

- Z.B. Studenten mit ihren Hausaufgabenergebnissen, Studenten ohne Hausaufgabenergebnis werden mit Null-Werten aufgeführt:

$\texttt{STUDENTEN} ⟕ \pi_{\texttt{SID,ANR,PUNKTE}}(\sigma_{\texttt{ATYP='H'}}(\texttt{BEWERTUNGEN}))$

SID	VORNAME	NACHNAME	EMAIL	ANR	PUNKTE
101	Lisa	Weiss	···	1	10
101	Lisa	Weiss	···	2	8
102	Michael	Grau	NULL	1	9
102	Michael	Grau	NULL	2	9
103	Daniel	Sommer	···	1	5
104	Iris	Winter	···	NULL	NULL

Äußerer Verbund (5)

- Übung: Gibt es irgendeinen Unterschied zwischen
  - ◇ `STUDENTEN` ⋈ `BEWERTUNGEN` und
  - ◇ `STUDENTEN` ⟕ `BEWERTUNGEN`?
- Der äußere Verbund ist vor allem mit Aggregationsfunktionen (z.B. `count`, `sum`) sinnvoll, siehe unten.

  Aggregationsfunktionen werden hier nur für SQL eingeführt.
- Mit einer Selektion auf das Ergebnis des äußeren Verbunds, kann man ihn wie eine Mengendifferenz verwenden: Aber das ist fragwürdiger Stil.

  Früher notwendig in MySQL (hat Unteranfragen erst ab Version 5.1). Auch heute noch soll die Lösung mit Outer Join schneller sein.

Äußerer Verbund (6)

- Der äußere Verbund ist eine abgeleitete Operation (wie $\bowtie$, $\cap$), d.h. man kann ihn mit den fünf Basisoperationen der relationalen Algebra darstellen.
- Z.B. seien $R(A, B)$ und $S(B, C)$ zwei Relationen.
- Dann ist der linke äußere Verbund R ⟕ S eine Abkürzung für

$$R \bowtie S \quad \cup \quad (R - \pi_{A,B}(R \bowtie S)) \times \{(C\colon null)\}$$

(wobei $\bowtie$ noch durch $\times$, σ, π ersetzt werden kann).

D.h. der äußere Verbund fügt dem normalen Verbund die Tupel aus R zu, die keinen Partner haben (aufgefüllt mit $C\colon null$, um dasselbe Schema zu erhalten, da man sonst die Vereinigung nicht anwenden kann).

Äußerer Verbund (7)

- Der SQL-86-Standard hat keine expliziten Verbunde. Da man einen Verbund durch andere Konstrukte darstellen kann, ist dies kein echtes Problem.
- Aber einige Anfragen können durch den äußeren Verbund wesentlich kürzer dargestellt werden.
- Deshalb wurde der äußere Verbund im SQL-92-Standard hinzugefügt:

```
SELECT R.A, R.B, S.C
FROM   R LEFT OUTER JOIN S ON R.B = S.B
```

- Damit wurde SQL aber eine sehr komplexe Mischung aus relationaler Algebra und Tupelkalkül.

Inhalt

Definitionen: Einleitung (1)

- In diesem Abschnitt soll zunächst (als Anhang oder Zusammenfassung) gezeigt werden, daß man die relationale Algebra auch ohne Rückgriff auf die Logik auf wenigen Folien präzise definieren kann.

  Die dann folgenden Bemerkungen zur Ausdruckskraft sind aber nicht ein Anhang, sondern prüfungsrelevant.

- Zunächst müssen Datentypen gegeben sein, d.h.
  - ◇ Eine Menge $\mathcal{S}_{\mathcal{D}}$ von Datentypnamen, und für jedes $D \in \mathcal{S}_{\mathcal{D}}$ eine Menge $\mathcal{I}_{\mathcal{D}}[D]$ von Werten.

    Zur Vereinfachung wird in diesem kurzen Abschnitt nicht zwischen Konstanten und den durch sie benannten Werten unterschieden. Es werden auch keine Datentyp-Operationen betrachtet.

Definitionen: Einleitung (2)

- Sei weiter eine Menge $\mathcal{A}$ möglicher Attributnamen (Identifier) gegeben.
- Ein Relationsschema ρ (Schema einer einzigen Relation) ist eine endliche Folge von Paaren bestehend jeweils aus einem Attributnamen und einem Datentyp (d.h. aus $\mathcal{A} \times \mathcal{S}_{\mathcal{D}}$), geschrieben als

  $$\rho = (A_1 \colon D_1, \ \ldots, \ A_n \colon D_n).$$

  Es muß dabei $A_i \neq A_j$ für $i \neq j$ gelten.

  Keine Relation kann zwei Spalten mit gleichem Namen haben.

Definitionen: Einleitung (3)

- Ein relationales DB-Schema $\mathcal{S}$ besteht aus
  - ◇ einer endlichen Menge Relationsnamen $\mathcal{R}$, und
  - ◇ für jedes $R \in \mathcal{R}$, ein Relationsschema $sch(R)$.

  Zu einem Datenbankschema gehören normalerweise auch Integritätsbedingungen (Schlüssel, Fremdschlüssel, etc.), die sind für die Definition der relationalen Algebra aber nicht wichtig.

- Ein DB-Zustand $\mathcal{I}$ definiert eine endliche Relation $\mathcal{I}[R]$ für jeden Relationsnamen R des Schemas: Ist $sch(R) = (A_1 \colon D_1, \ldots, A_n \colon D_n)$, dann

  $$\mathcal{I}(R) \subseteq \mathcal{I}_{\mathcal{D}}[D_1] \times \cdots \times \mathcal{I}_{\mathcal{D}}[D_n].$$

Definitionen: Syntax (1)

- Man definiert rekursiv die Menge relationaler Algebra (RA) -Ausdrücke zusammen mit dem relationalen Schema jedes RA-Ausdrucks. Basisfälle:
  - ◇ $\boxed{R}$: Für jedes $R \in \mathcal{R}$, ist der Relationsname R ein RA-Ausdruck mit Schema $sch(R)$.
  - ◇ $\boxed{\{(A_1 \colon d_1, \ldots, A_n \colon d_n)\}}$ ("Relationskonstante") ist ein RA-Ausdruck, wenn $A_1, \ldots, A_n \in \mathcal{A}$, und $d_i \in \mathcal{I}_{\mathcal{D}}[D_i]$ für $1 \leq i \leq n$ mit $D_1, \ldots, D_n \in \mathcal{S}_{\mathcal{D}}$. Das Schema dieses RA Ausdrucks ist gegeben durch $(A_1 \colon D_1, \ldots, A_n \colon D_n)$.

Definitionen: Syntax (2)

- Rekursive Fälle, Teil 1: Sei Q ein RA-Ausdruck mit Schema $\rho = (A_1\colon D_1, \ldots, A_n\colon D_n)$. Dann sind auch die folgenden RA-Ausdrücke:
  - ◇ $\boxed{\sigma_{A_i=A_j}(Q)}$ für $i, j \in \{1, \ldots, n\}$. Mit Schema: ρ.
  - ◇ $\boxed{\sigma_{A_i=d}(Q)}$ für $i \in \{1, \ldots, n\}$ und $d \in \mathcal{I_D}[D_i]$. Mit Schema: ρ.
  - ◇ $\boxed{\pi_{B_1 \leftarrow A_{i_1}, \ldots, B_m \leftarrow A_{i_m}}(Q)}$ für $i_1, \ldots, i_k \in \{1, \ldots, n\}$ und $B_1, \ldots, B_m \in \mathcal{A}$, so daß $B_j \neq B_k$ für $j \neq k$. Mit Schema $(B_1\colon D_{i_1}, \ldots, B_m\colon D_{i_m})$.

Definitionen: Syntax (3)

- Rekursive Fälle, fortgesetzt: Seien Q_1 und Q_2 RA-Ausdrücke mit gleichem Schema ρ. Dann sind auch die folgenden RA-Ausdrücke mit dem Schema ρ:
  - ◇ $\boxed{(Q_1) \cup (Q_2)}$ und ◇ $\boxed{(Q_1) - (Q_2)}$
- Seien Q_1 und Q_2 RA-Ausdrücke mit den Schemata $(A_1\colon D_1, \ldots, A_n\colon D_n)$ bzw. $(B_1\colon E_1, \ldots, B_n\colon E_m)$. Ist $\{A_1, \ldots, A_n\} \cap \{B_1, \ldots, B_m\} = \emptyset$, dann ist auch folgendes ein RA-Ausdruck:
  - ◇ $\boxed{(Q_1) \times (Q_2)}$ Schema: $(A_1{:}D_1, \ldots, A_n{:}D_n, B_1{:}E_1, \ldots, B_m{:}E_m)$.

Definitionen: Syntax (4)

- Nichts anderes ist ein RA-Ausdruck.

  Dies ist formal notwendig, um die Definition zu vervollständigen. Die Definition besteht sonst nur aus Bedingungen der Form "Ist R ein RA-Ausdruck, so ist auch S ein RA-Ausdruck." Dies würde einschließen, daß alles ein RA-Ausdruck ist (die Folgerungen der Regeln sind dann immer wahr, somit sind die Regeln erfüllt). Dies ist natürlich bei dieser Definition nicht gemeint. Somit ist es notwendig herauszustellen, daß etwas nur dann ein RA-Ausdruck ist, wenn es durch begrenzt häufige Anwendung obiger Regeln konstruiert werden kann, da nichts anderes ein RA-Ausdruck ist.

- Übung: Definieren Sie eine kontextfreie Grammatik für RA-Ausdrücke.

  Ignorieren Sie dabei die Schema-Beschränkungen und die Forderung, daß die verwendeten Attributnamen deklariert sein müssen.

Abkürzungen

- Klammern kann man bei klarer Struktur (oder bei Äquivalenz der möglichen Strukturen) weglassen.

  Die obige Definition verlangt viele Klammern, um mit einfachen Regeln sicherzustellen, daß die Struktur immer eindeutig festgelegt ist. Mit komplexeren Regeln ist es möglich, die Anzahl der Klammern zu verringern. Man kann dazu insbesondere auch Bindungsstärken festlegen (Operator-Prioritäten), z.B. bindet $\times$ stärker als $\cup$. Dies ist aber für theoretische Untersuchungen nicht wichtig.

- Wie oben erklärt, kann man zusätzliche Algebraoperationen (z.B. $\bowtie$) als Abkürzungen einführen.

  Dies ist wieder für die praktische Anwendung der Anfragesprache wichtig, aber nicht für die theoretischen Ergebnisse, da die Abkürzungen immer zu ihrer vollen Form expandiert werden können.

Definitionen: Semantik (1)

- Das Ergebnis einer Anfrage Q, d.h. eines RA-Ausdrucks, in einem DB-Zustand $\mathcal{I}$ ist eine Relation.
- Das Anfrageergebnis wird $\mathcal{I}[Q]$ geschrieben und rekursiv entsprechend der Struktur von Q definiert:
  - ◇ Ist Q ein Relationsname R, dann $\mathcal{I}[Q] := \mathcal{I}(R)$.

    D.h. es wird einfach der Wert der Relation R aus dem aktuellen Datenbank-Zustand genommen.
  - ◇ Ist Q die konstante Relation $\{(A_1\colon d_1, \ldots, A_n\colon d_n)\}$, dann $\mathcal{I}[Q] := \{(d_1, \ldots, d_n)\}$.

Definitionen: Semantik (2)

- Definition des Wertes $\mathcal{I}[Q]$ eines RA-Ausdrucks Q im Zustand $\mathcal{I}$, Fortsetzung:
  - ◇ Hat Q die Form $\sigma_{A_i=A_j}(Q_1)$, dann
    $$\mathcal{I}[Q] := \{(d_1, \ldots, d_n) \in \mathcal{I}[Q_1] \mid d_i = d_j\}.$$
  - ◇ Hat Q die Form $\sigma_{A_i=d}(Q_1)$, dann
    $$\mathcal{I}[Q] := \{(d_1, \ldots, d_n) \in \mathcal{I}[Q_1] \mid d_i = d\}.$$
  - ◇ Hat Q die Form $\pi_{B_1\leftarrow A_{i_1},\ldots,B_m\leftarrow A_{i_m}}(Q_1)$, dann
    $$\mathcal{I}[Q] := \{(d_{i_1}, \ldots, d_{i_m}) \mid (d_1, \ldots, d_n) \in \mathcal{I}[Q_1]\}.$$

Definitionen: Semantik (3)

- Definition von $\mathcal{I}[Q]$, fortgesetzt:
  - ◇ Hat Q die Form $(Q_1) \cup (Q_2)$ dann
    $$\mathcal{I}[Q] := \mathcal{I}[Q_1] \cup \mathcal{I}[Q_2].$$
  - ◇ Hat Q die Form $(Q_1) - (Q_2)$ dann
    $$\mathcal{I}[Q] := \mathcal{I}[Q_1] - \mathcal{I}[Q_2].$$
  - ◇ Hat Q die Form $(Q_1) \times (Q_2)$, dann
    $$\begin{aligned}\mathcal{I}[Q] := \{(d_1, \ldots, d_n, e_1, \ldots, e_m) \mid\ & (d_1, \ldots, d_n) \in \mathcal{I}[Q_1],\\ & (e_1, \ldots, e_m) \in \mathcal{I}[Q_2]\}.\end{aligned}$$

Grenzen der RA (1)

- R sei ein Relationsname mit Schema $(A\colon D, B\colon D)$ und $\mathcal{I}_{\mathcal{D}}[D]$ sei endlich.
- Der transitive Abschluß von $\mathcal{I}[R]$ ist die Menge aller $(d, e) \in \mathcal{I}_{\mathcal{D}}[D] \times \mathcal{I}_{\mathcal{D}}[D]$, so daß es ein $n \in \mathbb{N}$ $(n \geq 1)$ und $d_0, \ldots, d_n \in \mathcal{I}_{\mathcal{D}}[D]$ gibt mit $d = d_0$, $e = d_n$ und $(d_{i-1}, d_i) \in \mathcal{I}(R)$ für $i = 1, \ldots, n$.
- Z.B. könnte R die Relation "`ELTERN`" sein, dann besteht der transitive Abschluß aus allen Ahnenbeziehungen (Eltern, Großeltern, Urgroßeltern, ...).

Grenzen der RA (2)

- Theorem: Es gibt keinen RA-Ausdruck Q, so daß $\mathcal{I}[Q]$ die transitive Hülle von $\mathcal{I}[R]$ ist (für alle DB-Zustände $\mathcal{I}$).
- Zur Berechnung der Vorfahren braucht man einen zusätzlichen Verbund für jede weitere Generation.
- Daher kann man keine Anfrage schreiben, die bei beliebigen DB-Zuständen funktioniert.

  Natürlich kann man eine Anfrage schreiben, die z.B. bis zu den Ururgroßeltern funktioniert. Aber dann funktioniert sie nicht korrekt, falls die Datenbank (Ur)^{3}großeltern enthält.

Grenzen der RA (3)

- Das impliziert natürlich, daß die relationale Algebra nicht berechnungsuniversell ist:
  - ◇ Nicht jede Funktion von DB-Zuständen auf Relationen (Antwortmengen), die man mit einem C-Programm berechnen könnte, kann auch in relationaler Algebra formuliert werden.

    Man kann dies auch nicht fordern, da man garantieren möchte, daß die Anfrageauswertung terminiert. Für die Relationenalgebra gilt, daß Anfragen immer in endlicher Zeit vollständig ausgewertet werden können. Für allgemeine Programme ist dagegen unentscheidbar, ob sie terminieren. Jede berechnungsuniverselle Sprache muß notwendigerweise zulassen, daß Anfragen/Programme formuliert werden können, deren Ausführung nicht endet.

Grenzen der RA (4)

- Alle RA-Ausdrücke können in einer Zeit berechnet werden, die polynomial in der Größe der DB ist.
- Somit können auch sehr komplexe Funktionen nicht in relationaler Algebra formuliert werden.

  Wenn Sie z.B. einen Weg finden sollten, das Travelling Salesman Problem in relationaler Algebra zu formulieren, haben Sie das berühmte P=NP Problem gelöst. Da dies sehr unwahrscheinlich ist, sollten Sie es gar nicht versuchen, sondern ein C-Programm schreiben.
- Aber man kann nicht alle Probleme von polynomialer Komplexität in RA formulieren.

  Vgl. transitive Hülle. Mit einem Fixpunktoperator und einer linearen Ordnung auf den Domains ist dies möglich ($\rightarrow$ Deduktive DB).

Ausdruckskraft (1)

- Eine Anfragesprache $\mathcal{L}$ für das relationale Modell nennt man streng relational vollständig gdw. es für jedes DB-Schema $\mathcal{S}$ und für jeden RA-Ausdruck Q_1 bezüglich $\mathcal{S}$ eine Anfrage $Q_2 \in \mathcal{L}_{\mathcal{S}}$ gibt, so daß für alle DB-Zustände $\mathcal{I}$ für $\mathcal{S}$ die beiden Anfragen das gleiche Ergebnis liefern: $\mathcal{I}[Q_1] = \mathcal{I}[Q_2]$.
- D.h. $\mathcal{L}$ ist streng relational vollständig gdw. jede Anfrage Q_1 der Relationenalgebra in eine äquivalente Anfrage Q_2 in $\mathcal{L}$ übersetzt werden kann.

Ausdruckskraft (2)

- Z.B. ist SQL streng relational vollständig.
- Wenn die Übersetzung der Anfragen in beide Richtungen möglich ist, haben beide Anfragesprachen die gleiche Ausdruckskraft.

  Z.B. enthält SQL Aggregationen, die in relationaler Algebra nicht simuliert werden können. Daher ist SQL mächtiger als die relationale Algebra. Aber mann kann die relationale Algebra natürlich mit Aggregationsoperatoren erweitern.

- "Relational vollständig" (ohne "streng") gestattet Folgen von Anfragen zu verwenden und Zwischenergebnisse in temporären Relationen zu speichern.

Ausdruckskraft (3)

- Folgende Sprachen haben gleiche Ausdruckskraft:
  - ◇ Relationale Algebra
  - ◇ SQL ohne Aggregationen und Nullwerte sowie mit verbindlicher Duplikateliminierung
  - ◇ Tupelkalkül (bereichsunabhängige Formeln)

    Mit der Möglichkeit, auch Variablen für Tupel einzuführen, die nicht direkt an Relationen gebunden sind.
  - ◇ Bereichskalkül (bereichsunabhängige Formeln)
  - ◇ Datalog (eine Prolog-Variante) ohne Rekursion
- Deshalb ist die Menge der Funktionen, die in RA ausgedrückt werden können, nicht willkürlich.

Von RA zum BK (1)

- Es soll jetzt gezeigt werden, dass es für jede Anfrage der relationalen Algebra eine äquivalente Anfrage des Bereichskalküls gibt.

  Äquivalente Anfragen liefern für jeden Zustand die gleiche Antwort.

- Dazu konstruieren wir für jeden Ausdruck Q der relationalen Algebra mit Schema $(A_1{:}\,D_1, \ldots, A_n{:}\,D_n)$ eine Formel F des Bereichskalküls mit freien Variablen $\{A_1, \ldots, A_n\}$ der entsprechenden Datentypen, so dass $(d_1, \ldots, d_n) \in \mathcal{I}[Q]$ gdw. $\langle \mathcal{I}, \mathcal{A} \rangle \models F$ für die Variablenbelegung $\mathcal{A} = \{A_1/d_1, \ldots, A_n/d_n\}$.

Von RA zum BK (2)

- Der Beweis ist mit struktureller Induktion.

  Man kann also annehmen, dass man für die kleineren Teilausdrücke bereits entsprechende Formeln konstruiert hat, und kann diese Formeln bei der Konstruktion der Formel für den Gesamtausdruck verwenden.

- Ist Q ein Relationsname R, und $(A_1{:}\,D_1, \ldots, A_n{:}\,D_n)$ das Schema dieser Relation. Dann ist die entsprechende Formel:

$$R(A_1, \ldots, A_n).$$

- Ist Q eine konstante Relation $(A_1{:}\,c_1, \ldots, A_n{:}\,c_n)$. Dann ist die entsprechende Formel:

$$A_1 = c_1 \wedge \cdots \wedge A_n = c_n.$$

Von RA zum BK (3)

- Ist Q eine Selektion $\sigma_\varphi(Q_1)$, und ist F_1 die Q_1 entsprechende Formel, so konstruiert man folgende Formel:
$$F_1 \wedge \varphi.$$
- Ist Q eine einfache Projektion $\pi_{A_1,\ldots,A_n}(Q_1)$, und sind $\{B_1,\ldots,B_k\}$ die Attribute aus dem Schema von Q_1, die nicht in $\{A_1,\ldots,A_n\}$ enthalten sind (die wegprojizierten Attribute), so konstruiert man die Formel:
$$\exists\, B_1 \ldots \exists\, B_k\colon F_1,$$
wobei F_1 die Q_1 entsprechende Formel ist.

Von RA zum BK (4)

- Für eine allgemeine Projektion $\pi_{B_1\leftarrow t_1,\ldots,B_k\leftarrow t_k}(Q_1)$ konstruiert man folgende Formel:
$$\exists X_1\ \ldots\ \exists X_k\colon B_1 = X_1 \wedge \cdots \wedge B_k = X_k \wedge$$
$$(\exists A_1\ \ldots\ \exists A_n\colon F_1 \wedge X_1 = t_1 \wedge \cdots \wedge X_k = t_k),$$
dabei sind
  - ◇ F_1 die Q_1 entsprechende Formel,
  - ◇ $X_1,\ldots,X_k$ neue Variablen,

    D.h. nicht frei in F_1 und verschieden von den B_i. Diese "Zwischenvariablen" werden eingeführt weil die B_i in den t_i vorkommen können, z.B. $B \leftarrow B{+}1$. Hier würde $B = B{+}1$ nicht funktionieren.
  - ◇ $A_1,\ldots,A_n$ die Attribute von Q_1.

    D.h. die freien Variablen von F_1.

Von RA zum BK (5)

- Für ein kartesisches Produkt $Q_1 \times Q_2$ konstruiert man die Formel
$$F_1 \wedge F_2,$$
dabei sind F_1 und F_2 die Formeln, die rekursiv für Q_1 und Q_2 konstruiert wurden.

  Die einfache "und"-Verknüpfung würde auch für den natürlichen Verbund funktionieren.
- Für die Vereinigung $Q_1 \cup Q_2$ konstruiert man die Formel
$$F_1 \vee F_2,$$
dabei entsprechen F_1 und F_2 wieder Q_1 und Q_2.

Von RA zum BK (6)

- Für die Mengendifferenz $Q_1 - Q_2$ konstruiert man die Formel
$$F_1 \wedge \neg F_2,$$
wobei F_1 und F_2 die Formeln für Q_1 und Q_2 sind.
- Alle konstruierten Formeln sind bereichsbeschränkt und damit bereichsunabhängig.

  Im Bereichskalkül sind nur bereichsunabhängige Formeln zugelassen, um die Auswertbarkeit in endlicher Zeit zu garantieren (man muß für die Variablen nur die in der DB vorkommenden Werte einsetzen). Die Bereichsbeschränktheit ist noch stärker und läßt sich syntaktisch prüfen. Obwohl $Q_1 \times Q_2$ und $\sigma_\varphi(Q_1)$ beide eine Konjunktion "$F_1 \wedge F_2$" bzw. "$F_1 \wedge \varphi$" liefern, ist der Unterschied, dass bei F_1 und F_2 die Variablen an einen endlichen Bereich gebunden sind, aber bei φ nicht.

Vom BK zur RA (1)

- Es soll jetzt auch umgekehrt gezeigt werden, dass es zu jeder bereichsunabhängigen Formel des Bereichskalküls einen äquivalenten Ausdruck der Relationenalgebra gibt.
- Damit sind RA und BK gleich mächtig:
  - ◇ Es gibt also keine abstrakte Anfrage (Funktion von Datenbank-Zuständen auf Antworten), die nur in dem einen Formalismus beschrieben werden kann, aber nicht im anderen.

    Man kann Anfragen in beiden Richtungen übersetzen.

Vom BK zur RA (2)

- Während die für einen RA-Ausdruck Q erzeugte Formel F auch praktische Bedeutung hat, zeigt der in umgekehrter Richtung erstellte RA-Ausdruck nur die Existenz.

  Man würde niemals einen so komplizierten/ineffizienten RA-Ausdruck manuell schreiben. Ein Problem ist, dass man die Bereichsunabhängigkeit nicht testen kann. Die Aussage ist also nur: Falls die Formel bereichsunabhängig war, ist der RA-Ausdruck äquivalent. Mit bereichsbeschränkten Formeln hätte man eine Chance, einen praktisch relevanten RA-Ausdruck zu erstellen, aber das ist mindestens nicht einfach (man muß u.a. Konjunktionen umsortieren und ggf. verdoppeln). Für die theoretische Aussage reicht es, überhaupt einen äquivalenten RA-Ausdruck zu konstruieren, und das ist recht einfach möglich, wie im folgenden gezeigt werden soll.

Vom BK zur RA (3)

- Zunächst erstellt man RA-Ausdrücke, die den aktiven Bereich einer Sorte s berechnen.

  Also alle Werte dieser Sorte, die aktuell in der Datenbank vorkommen, oder in der gegebenen Formel.

  - ◇ Seien $R_1.A_1, \ldots, R_n.A_n$ alle Attribute der Sorte s.
  - ◇ Seien $c_1, \ldots, c_m$ alle Konstanten der Sorte s in der Anfrage.
  - ◇ Sei X ein Variablenname.
  - ◇ Dann sei $DOM_{X,s}$ der folgende RA-Ausdruck:

$$\begin{array}{lcl} \pi_{X \leftarrow A_1}(R_1) & \cup \cdots \cup & \pi_{X \leftarrow A_n}(R_n) \\ \cup \{(X\colon c_1)\} & \cup \cdots \cup & \{(X\colon c_m)\}. \end{array}$$

Vom BK zur RA (4)

- Ziel ist es nun, für jede BK-Formel F bezüglich der Variablendeklaration $\{X_1/s_1, \ldots, X_n/s_n\}$ einen RA-Ausdruck Q mit Schema $(X_1\colon s_1, \ldots, X_n\colon s_n)$ zu entwickeln, so dass für alle $d_i \in \mathcal{I}[s_i]$, $i = 1, \ldots, n$ gilt:

$$(d_1, \ldots, d_n) \in \mathcal{I}[Q]$$

  gdw.

  - ◇ $\langle \mathcal{I}, \{X_1/d_1, \ldots, X_n/d_n\} \rangle \models F$
  - ◇ d_i ist im aktiven Bereich der Sorte s_i enthalten $(i = 1, \ldots, n)$.

    Für bereichsunabhängige Formeln ist diese Bedingung überflüssig, aber die Induktion muß auch nicht bereichsunabhängige Teilformeln behandeln.

Vom BK zur RA (5)

- Für atomare Formeln $p(t_1, \ldots, t_k)$ mit Datenbank-Relation p mit Attributen $A_1, \ldots, A_k$ erzeuge man folgenden Ausdruck:

  $\pi_{X_1,\ldots,X_n}(\sigma_{Y_1=t_1 \wedge \cdots \wedge Y_k=t_k}($
  $DOM_{X_1,s_1} \times \cdots \times DOM_{X_n,s_n} \times \pi_{Y_1 \leftarrow A_1,\ldots,Y_k \leftarrow A_k}(p)))$

  Dabei sind $Y_1, \ldots, Y_k$ neue Variablen.

- Für atomare Formeln $p(t_1, \ldots, t_k)$ mit Datentyp-Prädikat p oder Gleichungen erzeuge man:

$$\sigma_{p(t_1,\ldots,t_k)}(DOM_{X_1,s_1} \times \cdots \times DOM_{X_n,s_n})$$

Vom BK zur RA (6)

- Für Formeln der Form $\neg F_1$ erzeuge man

$$(DOM_{X_1,s_1} \times \cdots \times DOM_{X_n,s_n}) - Q_1,$$

  dabei der Q_1 die rekursiv für F_1 konstruierte RA-Ausdruck.

- Für Formeln der Form $F_1 \vee F_2$ erzeuge man

$$Q_1 \cup Q_2$$

  wobei Q_1 und Q_2 die F_1 und F_2 entsprechenden Ausdrücke sind.

Vom BK zur RA (7)

- Sei nun die Formel $\exists Y\colon F_1$ betrachtet, und sei Q_1 der F_1 entsprechende Algebra-Ausdruck.
  - Falls $Y \notin \{X_1, \ldots, X_n\}$, so konstruiert man

$$\pi_{X_1,\ldots,X_n}(Q_1)$$

  - Ist dagegen $Y = X_i$ für ein $i \in \{1, \ldots, n\}$, so verwendet man den Ausdruck

$$DOM_{X_i,s_i} \times \pi_{X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_n}(Q_1).$$

- Die Junktoren $\wedge$, $\leftarrow$, $\rightarrow$, $\leftrightarrow$ und der Quantor $\forall$ brauchen nicht behandelt zu werden, weil man sie durch $\neg$, $\vee$, $\exists$ ausdrücken/ersetzen kann.