Teil 6: 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 Basisoperatioen kennen.

- Anfragen der relationalen Algebra des Typs "join-select-project" schreiben.

  Zusätzlich einige Anfragen schreiben, die Mengendifferenz und Vereinigung enthalten.

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

Inhalt

1. Einführung, Selektion, Projektion
2. Kartesisches Produkt, Verbund
3. Mengenoperationen
4. äußerer Verbund
5. Formale Definitionen, etwas Theorie

Beispieldatenbank (1)

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Beispieldatenbank (2)

- STUDENTS: 1 Zeile für jeden Studenten der Vorlesung
  - ◇ SID: "Studenten-ID" (Primärschlüssel).
  - ◇ FIRST, LAST: Vor- und Nachname.
  - ◇ EMAIL: E-Mail-Adresse (kann null sein).
- EXERCISES: eine Zeile für jede Übung.
  - ◇ CAT: Übungskategorie (Schlüssel mit ENO)
    Z.B. 'H': homework, 'M': midterm exam, 'F': final exam.
  - ◇ ENO: Übungsnummer (innerhalb der Kategorie).
  - ◇ TOPIC: Thema der Übung.
  - ◇ MAXPT: Maximalpunktzahl (Wieviele Punkte ist es wert?).

Beispieldatenbank (3)

- RESULTS: eine Zeile für jede eingereichte Lösung zu einer Übung.
  - ◇ SID: Student, der die Lösung geschrieben hat.
    Dies ist ein Fremdschlüssel, der STUDENTS refenziert.
  - ◇ CAT, ENO: Identifikation der Übung.
    Dies ist ein Fremdschlüssel, der EXERCISES referenziert.
    Zusammen mit SID bildet er den Primärschlüssel der Tabelle.
  - ◇ POINTS: Anzahl der Punkte, die der Student für die Lösung bekam.
  - ◇ Eine fehlende Zeile in dieser Tabelle bedeutet, dass der Student für diese Übung noch keine Lösung eingereicht hat.

Relationale Algebra (1)

- Die relationale Algebra (RA) ist eine theoretische Anfragesprache für das relationale Modell.
- Die relationale Algebra wird in keinem kommerziellen System auf Nutzer-Interface-Level verwendet.
- Varianten werden genutzt, um interne Anfragen (für Anfrageoptimierung und -ausführung) darzustellen.
- Kenntnis der relationalen Algebra wird helfen, SQL und relationale Datenbanksysteme 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.
- Eine Operation der relationalen Algebra ist $\cup$ (Union/Vereinigung). Dies ist klar, da Relationen auch Mengen sind.

Relationale Algebra (3)

- Eine andere Operation der relationalen Algebra ist die Selektion.

  Im Gegensatz zu Operationen wie + für Integers, 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:

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

RESULTS

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

=

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12

Relationale Algebra (4)

- Da die Ausgabe einer Operation der relationalen Algebra wieder eine Relation ist, kann sie Eingabe für eine andere relationale Algebra Operation 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.
- Man kennt arithmetische Ausdrücke wie $(x+2)*y$.
- In der relationalen Algebra verbindet man Relationen:

$$\pi_{\texttt{FIRST, LAST}}(\texttt{STUDENTS} \bowtie \sigma_{\texttt{CAT='M'}}(\texttt{RESULTS})).$$

Relationale Algebra (5)

Geringe 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 Verwendung einer drei-Werte-Logik wie in SQL würde die Definitionen stark verkomplizieren.

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

  In SQL sind Relationen Multimengen und können Nullwerte enthalten. Falls erforderlich, muss man die Duplikatelimination explizit verlangen (mit "DISTINCT"). In der relationalen Algabra muss man über Nullwerte 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 einer Anfragesprache 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 SQL-Anfrage (ohne Nullwerte, Aggregationen und Duplikate) auch in relationaler Algebra formuliert werden. Vgl. auch Folie 6-115.

Selektion (1)

- Die Operation σ_φ selektiert eine Teilmenge der Tupel einer Relation, nämlich diese, 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. `SELECT[condition](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 hat die folgende Form:
  ⟨Term⟩ ⟨Vergleichsoperator⟩ ⟨Term⟩
- Die Bedingung φ gibt einen Wahrheitswert (wahr oder falsch) für ein gegebenes Eingabetupel zurück.
- Einen ⟨Term⟩ ("Ausdruck") kann man für ein gegebenes Tupel zu einem Datentypelement auswerten:
  - ◇ zu einem Attributnamen,
  - ◇ zu einer Datentypkonstanten, oder
  - ◇ zu einem durch Datentypoperationen wie $+$, $-$, $*$, / zusammengesetzten Ausdruck aus Attributen and Konstanten.

Selektion (3)

- ⟨Vergleichsoperator⟩ ist
  - ◇ $=$ (gleich), $\neq$ (nicht gleich),
    Man kann auch "<>" anstatt $\neq$ schreiben.
  - ◇ $<$ (kleiner als), $>$ (größer als), $\leq$, $\geq$,
    Man kann auch <= anstatt $\leq$ und >= anstatt $\geq$ schreiben.
  - ◇ oder ein anderes Datentypprädikat (z.B. `LIKE`).
- Beispiele für Bedingungen:
  - ◇ `LAST = 'Smith'`
  - ◇ `POINTS >= 8`
  - ◇ `POINTS = MAXPT` (die Eingaberelation muss beide Attribute enthalten).

Selektion (4)

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

```
(1)  Create new temporary relation T;
(2)  foreach tuple t from input relation R do
(3)      Evaluate condition φ for tuple t;
(4)      if true then
(5)          insert t into T;
(6)      fi
(7)  od;
(8)  return T;
```

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

Selektion (5)

- Natürlich müssen die Attribute aus der Selektionsbedingung auch in der Eingabetabelle vorkommen:

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

- Folgendes ist zulässig, aber die Selektionsbedingung ist überflüssig, da die Bedingung stets wahr ist:

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

Selektion (6)

- Es ist kein Fehler, wenn das Ergebnis eines Ausdrucks der relationalen Algebra in einem bestimmten DB-Zustand leer ist:

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

- Es ist legal, aber meist ein Fehler, eine Bedingung zu verwenden, die stets falsch ist (inkonsistent):

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

Selektion (7)

- $\sigma_\varphi(R)$ entspricht folgender SQL-Anfrage:

```
SELECT *
FROM   R
WHERE  φ
```

- D.h. die Selektion entspricht der `WHERE`-Klausel.
- Eine andere Operation der relationalen Algebra, genannt "Projektion", entspricht der `SELECT`-Klausel in SQL. Das kann etwas verwirrend sein.

Erweiterte Selektion (1)

- In der Basis Selektionsoperation sind nur einfache Bedingungen, die einen einzelnen Vergleich enthalten ("atomare Formel"), möglich.
- Man kann die möglichen Bedingungen erweitern, indem man die Kombination einzelner Vergleiche durch die logischen Operatoren $\wedge$ (und), $\vee$ (oder), und $\neg$ (nicht) zulässt:

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)

- $\varphi_1 \wedge \varphi_2$ nennt man "Konjunktion von φ_1 und φ_2"
- $\varphi_1 \vee \varphi_2$ nennt man "Disjunktion von φ_1 und φ_2"
- $\neg\varphi_1$ nennt man "Negation von φ_1".
- Man kann auch "`und`", "`oder`" und "`nicht`" statt der Symbole "$\wedge$", "$\vee$", "$\neg$" verwenden.

  "$\wedge$" ist ähnlich wie das Mengenschnitt-Symbol "$\cap$", und tatsächlich sind die Tupel, die die Bedingung "$\wedge$" erfüllen, der Schnitt der Tupel, die die beiden Teilbedingungen erfüllen. In der gleichen Weise ist "$\vee$" vergleichbar mit "$\cup$" (Mengenvereinigung).

Erweiterte Selektion (3)

- Die Selektionsbedingung muss die Auswertung für jedes Eingabetupel getrennt zulassen.

  Somit sind, "existiert" ($\exists$) und "für alle" ($\forall$) und verschachtelte relationale Algebra Anfragen in Selektionsbedingungen nicht erlaubt.
- Diese 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 Übungen sind über "SQL"?

  Geben Sie die ganze Zeile der Tabelle aus. Die Projektion (zur Elimination von Spalten) wird unten behandelt.
- Geben Sie alle Einträge für HA 1 (`CAT='H'`) der Tabelle `RESULTS` aus, die weniger als 10 Punkte haben.

Dies bezieht sich auf das Schema auf Folie 6-4:

- `STUDENTS(`<u>`SID`</u>`, FIRST, LAST, EMAIL`^{O}`)`
- `EXERCISES(`<u>`CAT`</u>`, `<u>`ENO`</u>`, TOPIC, MAXPT)`
- `RESULTS(`<u>`SID`</u>$\rightarrow$`STUDENTS, (`<u>`CAT`</u>`, `<u>`ENO`</u>`)`$\rightarrow$`EXERCISES, POINTS)`

Projektion (1)

- Die Projektion π eliminiert Attribute (Spalten) aus der Eingaberelation.

  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 schreibt man `PROJECT[Columns](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 erstellt die Projektion $\pi_{A_{i_1},\ldots,A_{i_k}}(R)$ für jedes Eingabetupel $(A_1\colon d_1,\ldots,A_n\colon d_n)$ ein Ausgabetupel $(A_{i_1}\colon d_{i_1},\ldots,A_{i_k}\colon d_{i_k})$.

  Während σ gewisse Zeilen der Eingaberelation auswählt, und die anderen wegfallen läßt, wählt π gewisse Spalten aus, und vernachlässigt die anderen.

- D.h. die Attributwerte werden nicht verändert, aber nur die explizit genannten Attribute werden beibehalten. Alle anderen werden "wegprojiziert".

  Beachte: "eine Spalte wegzuprojizieren" ist DB-Slang. Normalerweise werden Dinge in oder auf etwas anderes projiziert.

Projektion (3)

- 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 (4)

- $\pi_{A_{i_1},\ldots,A_{i_k}}(R)$ kann wie folgt implementiert werden:

```
(1) Create new temporary relation T;
(2) foreach t = (A_1: d_1, ..., A_n: d_n) in R do
(3)     Compute u = (A_{i_1}: d_{i_1}, ..., A_{i_k}: d_{i_k});
(4)     insert u into T;
(5) od;
(6) return T;
```

- Dieses Programmfragment setzt voraus, dass "**insert**" Duplikate eliminiert.

Projektion (5)

- Die Projektion kann noch allgemeiner sein:
  - ◇ Attribute können umbenannt werden: $\pi_{B_1\leftarrow A_{i_1},\ldots,B_k\leftarrow A_{i_k}}(R)$ transformiert das Eingabetupel $(A_1\colon d_1,\ldots,A_n\colon d_n)$ in das Ausgabetupel $(B_1\colon d_{i_1},\ldots,B_k\colon d_{i_k})$.
  - ◇ Rückgabewerte können durch Datentypoperationen, wie + oder || (Stringkonkatenation), berechnet werden:

    $\pi_{\texttt{SID},\ \texttt{NAME}\leftarrow \texttt{FIRST}\ \texttt{||}\ \texttt{' '}\ \texttt{||}\ \texttt{LAST}}(\texttt{STUDENTS})$.
  - ◇ Spalten können mit Konstanten erstellt werden:

    $\pi_{\texttt{SID, FIRST, LAST, GRADE}\leftarrow \texttt{'A'}}(\texttt{STUDENTS})$.

Projektion (6)

- Die Projektion ist eine Abbildung, die auf jedes Eingabetupel angewandt wird.
- Jedes Eingabetupel wird lokal auf ein Ausgabetupel abgebildet. Hierbei sind nur Funktionen erlaubt, die auf einem einzigen Engabetupel definiert werden.

  Werte verschiedener Eingabetupel können nicht zu einem Ausgabetupel zusammengefasst werden (aber man betrachte kartesisches Produkt, siehe unten). Ansonsten sind sehr allgemeine Tupel-zu-Tupel Funktionen erlaubt.

Projektion (7)

- $\pi_{A_1,\ldots,A_n}(R)$ entspricht folgender SQL-Anfrage:

  ```
  SELECT DISTINCT A1, ..., An
  FROM   R
  ```

- Das Schlüsselwort `DISTINCT` ist nicht immer nötig.

  Die Anfrage ist ohne es schneller. `DISTINCT` ist unnötig, wenn $A_1, \ldots, A_n$ einen Schlüssel enthält. Manchmal sind Duplikate auch erwünscht.

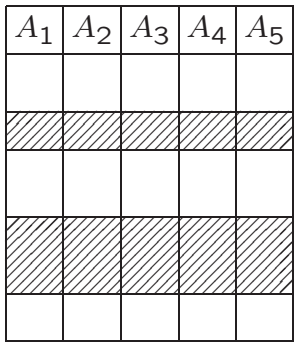
- $\pi_{B_1 \leftarrow A_1,\ldots,B_n \leftarrow A_n}(R)$ schreibt man in SQL wie folgt:

  ```
  SELECT DISTINCT A1 AS B1, ..., An AS Bn
  FROM   R
  ```

- Das Schlüsselwort `AS` kann weggelassen werden ("syntaktischer Zucker").

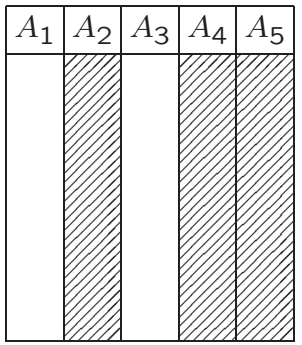
Zusammenfassung

Selektion σ



(Filtert gewisse Reihen)

Projektion π



(bildet jede Reihe ab)

Operationen verbinden (1)

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

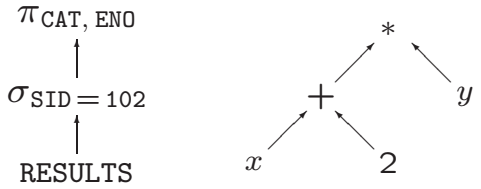
  $$\pi_{\texttt{CAT},\,\texttt{ENO}}(\sigma_{\texttt{SID}=102}(\texttt{RESULTS}))$$

- Ein Zwischenergebnis kann in einer temporären Relation gespeichert werden (kann man als Makro-Definition ansehen):

  $$\texttt{S102} := \sigma_{\texttt{SID}=102}(\texttt{RESULTS});$$
  $$\pi_{\texttt{CAT},\,\texttt{ENO}}(\texttt{S102})$$

Operationen verbinden (2)

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



- Zum Vergleich, ist der Operatorbaum des arithmetischen Ausdrucks $(x + 2) * y$ auf der rechten Seite dargestellt.

Zwischenergebnisse fließen entlang der Linien von unten nach oben.

Operationen verbinden (3)

- Auch in SQL-92 kann man das Ergebnis einer SQL-Anfrage als Eingabe für eine andere SQL-Anfrage nutzen:

```
SELECT CAT, ENO
FROM   (SELECT *
        FROM   RESULTS
        WHERE  SID = 102) AS S102
```

- Dies ist aber untypisch in SQL, und war im ersten SQL-Standard (SQL-86) nicht enthalten.
- Es ist kein guter Programmierstil die relationale Algebra in SQL 1:1 zu simulieren.

Operationen verbinden (4)

- In SQL kann man σ und π (und $\times$, siehe unten) in einer einzelnen SELECT-Anweisung zusammenfassen:

```
SELECT CAT, ENO
FROM   RESULTS
WHERE  SID = 102
```

- Komplexe Anfragen kann man Schritt für Schritt konstruieren:

```
CREATE VIEW S102
AS SELECT *
   FROM   RESULTS
   WHERE  SID = 102
```

- Dann kann man S102 wie eine gespeicherte Tabelle verwenden.

Basisoperatoren

- Die Blätter eines Operatorbaumes sind
  - ◇ die Namen von Datenbankrelationen
  - ◇ konstante Relationen (explizit genannte Tupel).
- Ein Relationsname R ist ein legaler Ausdruck der relationalen Algebra. Ihr Wert ist die gesamte, unter diesem Namen gespeicherte, Relation. Dies entspricht der SQL-Anfrage:

```
SELECT *
FROM   R
```

- Es ist nicht notwendig, eine Projektion auf alle Attribute zu schreiben.

Übungen (1)

Schreiben Sie folgende Anfrage in relationaler Algebra:

- Geben Sie die E-Mail-Adresse von Ann Smith aus.

  Schreiben Sie die Anfrage als Baum und verschachtelt mit Klammern.

Dies bezieht sich auf das Schema auf Folie 6-4:

- STUDENTS(SID, FIRST, LAST, EMAIL^O)
- EXERCISES(CAT, ENO, TOPIC, MAXPT)
- RESULTS(SID→STUDENTS, (CAT, ENO)→EXERCISES, POINTS)

Übungen (2)

- Welche der folgenden Ausdrücke der relationalen Algebra sind syntaktisch korrekt?
  Was bedeuten sie?
  - ☐ STUDENTS.
  - ☐ $\sigma_{\texttt{MAXPT} \neq 10}(\texttt{EXERCISES})$.
  - ☐ $\pi_{\texttt{FIRST}}(\pi_{\texttt{LAST}}(\texttt{STUDENTS}))$.
  - ☐ $\sigma_{\texttt{POINTS} \leq 5}(\sigma_{\texttt{POINTS} \geq 1}(\texttt{RESULTS}))$.
  - ☐ $\sigma_{\texttt{POINTS}}(\pi_{\texttt{POINTS} = 10}(\texttt{RESULTS}))$.

Inhalt

1. Einführung, Selektion, Projektion
2. Kartesisches Produkt, Verbund
3. Mengenoperationen
4. äußerer Verbund
5. Formale Definitionen, etwas Theorie

Kartesisches Produkt (1)

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

  Für σ und π gilt, dass jedes Ausgabetupel von einem einzigen Eingabetupel abgeleitet wird. Da π Duplikate eliminiert, ist es möglich, dass das gleiche Ausgabetupel von zwei verschiedenen Eingabetupeln abgeleitet wird, aber dann ist eins der beiden schon überflüssig.

- Dies wird durch das "kartesische Produkt" $\times$ getan.

  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 benötigt, setzt man (...) 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 große Restriktion. Man kann die Attribute erst umbenennen (π), und dann $\times$ anwenden.

Kartesisches Produkt (3)

- 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 $(R.A, R.B, S.B, S.C)$.
  - ◇ Man kann auch A und C verwenden (wie in SQL), da diese die Attribute eindeutig identifizieren.
- **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.

Kartesisches Produkt (4)

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

```
(1)  Create new temporary relation T;
(2)  foreach tuple t in R do
(3)      foreach tuple u in S do
(4)          insert t ∘ u into T;
(5)      od;
(6)  od;
(7)  return T;
```


Kartesisches Produkt (5)

- 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 ist das Ziel, nur ausgewählte Paare von Tupeln zu verbinden.
- Somit ist das kartesische Produkt nur als Eingabe für eine folgende Selektion sinnvoll.

Kartesisches Produkt (6)

- $R \times S$ wird in SQL geschrieben als

```
SELECT *
FROM   R, S
```

- In SQL ist es kein Fehler wenn zwei Relationen gleiche Attributnamen haben, da man Attribute auch in der Form "$R.A$" oder "$S.A$" referenzieren kann.

  Wenn die Anfrage wie oben ausgeführt wird, und R und S beide ein Attribut "A" haben, dann wird die Ergebnisrelation zwei Spalten mit dem gleichen Namen "A" haben. Dies ist für gespeicherte Relationen verboten, aber es kann bei Anfrageergebnissen vorkommen (wie im Beispiel). Man kann verschachtelte Anfragen als Eingaberelation unter `FROM` verwenden, dann erhält man aber bei jedem Versuch das dopppelte Attribut A in der Anfrage zu verwenden, einen Fehler ("Spalte mehrdeutig definiert").

Umbenennung

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

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

- 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.

  Einige Autoren definieren es so, aber die formale Definition ist nicht einfach.

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 "`R JOIN[A=B] S`".
- Der Verbund ist eine der wichtigsten und sinnvollsten Operationen der relationalen Algebra.

  Gleich nach der Selektion.

Verbund/Join (2)

STUDENTS ⋈ RESULTS

SID	FIRST	LAST	EMAIL	CAT	ENO	POINTS
101	Ann	Smith	···	H	1	10
101	Ann	Smith	···	H	2	8
101	Ann	Smith	···	M	1	12
102	Michael	Jones	(null)	H	1	9
102	Michael	Jones	(null)	H	2	9
102	Michael	Jones	(null)	M	1	10
103	Richard	Turner	···	H	1	5
103	Richard	Turner	···	M	1	7

- Studentin Maria Brown taucht nicht auf, da sie keine Hausaufgabe abgegeben und nicht an einer Klausur teilgenommen hat.
- Das oben gezeigte ist der natürliche Verbund der beiden Tabellen. Im folgenden wird aber zunächst der Standard-Verbund erklärt.

Verbund/Join (3)

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

```
(1)  Create new temporary relation T;
(2)  foreach tuple t in R do
(3)      foreach tuple u in S do
(4)          if t.A = u.B then
(5)              insert t ∘ u into T;
(6)          fi;
(7)      od;
(8)  od;
(9)  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.

  Natürlich speichert ein DBMS keine Zwischenergebnisse ab, es sei denn, es ist notwendig. Jeder Algebraoperator berechnet die Tupel nur auf Befehl ("pipelined evaluation"). Dann ist der nested loop join das gleiche wie das kartesische Produkt gefolgt von einer Selektion.

- Viele verschiedene Algorithmen wurden zur Berechnung eines Verbunds entwickelt.

  Z.B. "merge join", "index join", "hash join". Der nested loop join ist nur dann effizient, wenn eine der beiden Relationen klein ist. So kann die zusammengefasste Operation oft effizienter ausgewertet werden als $\times$ gefolgt von σ.

Verbund/Join (5)

- Die Verbundbedingung muss nicht die Form $A = B$ haben (obwohl sie am meisten 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 für den Verbund ist, Tupel basierend auf Fremdschlüssel zu verbinden, z.B.

  $$\texttt{RESULTS} \underset{\texttt{SID}=\texttt{SID}'}{\bowtie} \pi_{\texttt{SID}' \leftarrow \texttt{SID},\texttt{FIRST},\texttt{LAST},\texttt{EMAIL}}(\texttt{STUDENTS})$$

  Die Umbenennung des Attributs "SID" ist notwendig, da das kartesische Produkt disjunkte Attributnamen verlangt. Aber man vgl. unten: natürlicher Verbund.

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 "Semiverbund" ($\ltimes$, $\rtimes$) agiert nur als Filter.

  Er macht zuerst den Verbund, und projiziert die Ergebnistupel auf die Attribute der linken (linker Semiverbund) oder rechten Relation (rechter Semiverbund).

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

  Während das kartesische Produkt und der allgemeine Verbund verlangen, dass die Attribute beider Relationen verschiedene Namen haben, verwendet der natürliche Verbund die gleichen Namen für die Verbund-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 Datenbankzustand $\mathcal{I}$ alle Tupel der Form

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

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

Natürlicher Verbund (3)

- Der natürliche Vebund entspricht nicht nur dem kartesischen Produkt gefolgt von einer Selektion, sondern
  - benennt automatisch eine Kopie jedes gleichen Attributs vor dem kartesischen Produkt um, und
  - verwendet eine Projektion, um diese doppelten Attribute am Schluß zu eliminieren.
- Z.B., wenn $R(A, B)$, und $S(B, C)$ gegeben ist, dann ist $R \bowtie S$ eine Abkürzung für

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

Natürlicher Verbund (4)

Bemerkung zu relationalem Datenbank-Design:

- 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.
- Wenn Domainnamen als Attributnamen verwendet werden, geschieht dies automatisch.
- Man sollte versuchen zu verhindern, dass Attribute, die wahrscheinlich nicht verbunden werden, den gleichen Namen bekommen.

Verbund in SQL (1)

- $R \underset{A=B}{\bowtie} S$ schreibt man in SQL wie $\sigma_{A=B}(R \times S)$:

```
SELECT *
FROM   R, S
WHERE  A = B
```

- Attribute können explizit mit dem Relationsnamen referenziert werden (das ist notwendig, wenn der Attributname in beiden Relationen vorkommt):

```
SELECT *
FROM   R, S
WHERE  R.A = S.B
```


Verbund in SQL (2)

- In SQL-92 kann man auch schreiben:

```
SELECT *
FROM   R JOIN S ON R.A = S.B
```

- Das zeigt den Einfluß der relationalen Algebra, ist aber nicht wirklich im Sinne von SQL.

  Die klassische Form des Verbunds in SQL (vgl. nächste Folie) wird seit Jahren verwendet, und von vielen Menschen als leichter zu lesen empfunden. Obwohl SQL die Operation `UNION` der relationalen Algebra schon immer hatte, ist es stärker auf dem logischen Formalismus des "relationalen Tupelkalküls" basiert. Die neue Sytax wurde wahrscheinlich nur eingeführt, da der "äußere Verbund" (siehe unten) auf klassische Weise schwieriger zu formulieren ist.

- Z.B. wird die neue alternative Syntax des Verbunds in Oracle 8i nicht unterstützt.

Algebraische Gesetze (1)

- 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 (von links nach rechts) ist unterschiedlich.
- Falls jedoch danach eine Projektion folgt, ist das egal (man kann π auch für diesen Zweck einführen):

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

Algebraische Gesetze (2)

- 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 jede richtige Lösung wird es volle Punktzahl geben, ausgenommen wenn Punkte für unnötige Komlikationen abgezogen werden (z.B. π auf alle Spalten).

Allgemeines Anfragemuster (1)

- Folgende Anfragestruktur ist sehr gebräuchlich:

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

  - ◇ Erst verbindet man alle Tabellen, die für die Anfrage benötigt werden.
  - ◇ Dann selektiert man die relevanten Tupel.

    Da man beim ersten Schritt alle Tabellen verbindet, kann sich die Selektionsbedingung auf Attribute aller Tabellen beziehen.
  - ◇ Als drittes projiziert man auf die Attribute, die ausgeben werden sollen.

Allgemeines Anfragemuster (2)

- Muster sind häufig nützliche conventions of thought.
- Aber die Operationen der relatonalen Algebra können auf jede Weise verbunden 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 komplizieren, indem man eine Projektion auf alle Attribute oder eine Selektion mit einer stets wahren Bedingung verwendet (die einfach die Identitätsabbildung ist).
- Im Gegensatz dazu werden in SQL die Schlüsselworte `SELECT` und `FROM` benötigt, und die Reihenfolge muss immer so sein:

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


Allgemeines Anfragemuster (3)

- $\pi_{A_1,\ldots,A_k}(\sigma_\varphi(R_1 \bowtie \cdots \bowtie R_n))$ schreibt man in SQL:

```
SELECT DISTINCT A1, ..., Ak
FROM   R1, ..., Rn
WHERE  φ AND ⟨Join-Bedingung⟩
```

- Es ist ein verbreiteter Fehler eine Verbundbedingung zu vergessen.

  Dann erhält man ein kartesisches Produkt, welches zu falschen Antworten und häufig sehr großen Anfrageergebnissen führt.
- Normalerweise werden je zwei Relationen durch Gleichungen verbunden, z.B. $R_1.B_1 = R_2.B_2$.
- "`DISTINCT`" wird nicht immer benötigt (siehe oben).

Allgemeines Anfragemuster (4)

- 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.

Allgemeines Anfragemuster (5)

- 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?
  - ◇ Stattdessen müßte der Verbund sein $R \bowtie S \bowtie T$.
  - ◇ Oft ist eine graphische Representation der Fremdschlüsselbeziehungen zwischen Tabellen hilfreich (die typischen Verbunden entspricht).

Übung

Schreiben Sie folgende Anfragen in relationaler Algebra:

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

Dies bezieht sich auf das Schema auf Folie 6-4:

- STUDENTS(SID, FIRST, LAST, EMAIL^o)
- EXERCISES(CAT, ENO, TOPIC, MAXPT)
- RESULTS(SID→STUDENTS, (CAT, ENO)→EXERCISES, POINTS)

Selbstverbund (1)

- Manchmal ist es notwendig, sich auf mehr als ein Tupel einer Relation gleichzeitig zu beziehen.
- Z.B. Wer hat auf irgendeine Übung mehr Punkte als Student 101?
- In diesem Fall benötigt man zwei Tupel der Relation RESULTS, um ein Ergebnistupel zu berechnen:
  - ◇ Ein Tupel für Student 101.
  - ◇ Ein Tupel der gleichen Übung, bei dem POINTS größer als in dem ersten Tupel ist.

Selbstverbund (2)

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

$$S := \rho_{\mathtt{X}}(\mathtt{RESULTS}) \underset{\substack{\mathtt{X.CAT}=\mathtt{Y.CAT} \\ \wedge\, \mathtt{X.ENO}=\mathtt{Y.ENO}}}{\bowtie} \rho_{\mathtt{Y}}(\mathtt{RESULTS});$$

$$\pi_{\mathtt{X.SID}}(\sigma_{\mathtt{X.POINTS}>\mathtt{Y.POINTS} \wedge \mathtt{Y.SID}=101}(S))$$

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

Inhalt

1. Einführung, Selektion, Projektion

2. Kartesisches Produkt, Verbund

3. Mengenoperationen

4. äußerer Verbund

5. Formale Definitionen, etwas Theorie

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/Union).

Mengenoperationen (2)

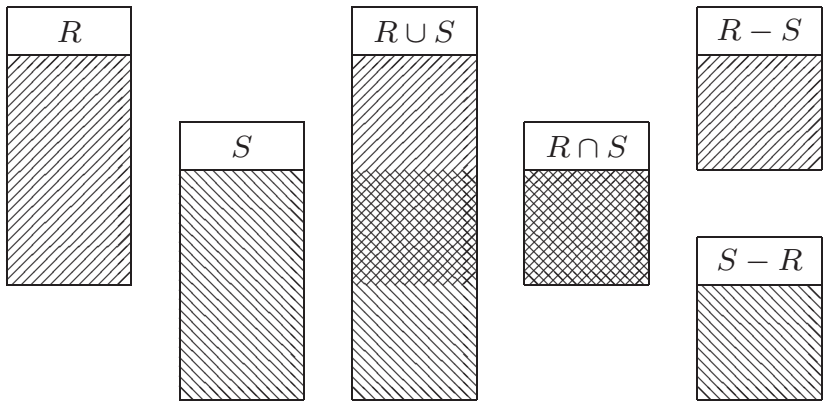
- $R - S$ enthält alle Tupel, die in R, aber nicht in S sind (Mengendifferenz/Set Difference).
- $R \cap S$ enthält Tupel, die in beiden, R und S, sind (Durchschnitt/Intersection).
- Der Durchschnitt ist (wie der Verbund) eine abgeleitete Operation: Er kann durch $-$ ausgedrückt werden:

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

- Übung: Beweisen Sie diese Gleichung.

  Z.B. Zeichnen Sie ein Venn-Diagram.

Mengenoperationen (3)



Mengenoperationen (4)

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

```
(1)  Create new temporary relation T;
(2)  foreach tuple t in R do
(3)      insert t into T;
(4)  od;
(5)  foreach tuple t in S do
(6)      insert t into T;
(7)  od;
(8)  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:

```
(1)  Create new temporary relation T;
(2)  foreach tuple t in R do
(3)      Remove := false;
(4)      foreach tuple u in S do
(5)          if u = t then
(6)              Remove := true;
(7)      od;
(8)      if not Remove then
(9)          insert t into T;
(10) od;
(11) return T;
```


Vereinigung/Union (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.

- Z.B. soll es neben registrierten Studenten, die Hausaufgaben abgeben und Prüfungen machen, auch Gäste geben, die an der Vorlesung teilnehmen:

  $$\texttt{GUESTS}(\underline{\texttt{FIRST}}, \underline{\texttt{LAST}}, \texttt{EMAIL}^{o}).$$

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

Vereinigung/Union (2)

- Mit $\cup$ ist dies einfach:

  $$\pi_{\texttt{EMAIL}}(\texttt{STUDENTS}) \cup \pi_{\texttt{EMAIL}}(\texttt{GUESTS}).$$

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

$$\texttt{MPOINTS} := \pi_{\texttt{SID,POINTS}}(\sigma_{\texttt{CAT='M'} \wedge \texttt{ENO=1}}(\texttt{RESULTS}));$$

$$\begin{aligned}
&\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'A'}}(\sigma_{\texttt{POINTS} \geq 12}(\texttt{MPOINTS})) \\
\cup\ &\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'B'}}(\sigma_{\texttt{POINTS} \geq 10 \wedge \texttt{POINTS} < 12}(\texttt{MPOINTS})) \\
\cup\ &\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'C'}}(\sigma_{\texttt{POINTS} \geq 7 \ \wedge \texttt{POINTS} < 10}(\texttt{MPOINTS})) \\
\cup\ &\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'F'}}(\sigma_{\texttt{POINTS} < 7}\ (\texttt{MPOINTS}))
\end{aligned}$$

Vereinigung/Union (3)

- In SQL kann man UNION zwischen zwei SELECT-Anweisungen schreiben:

```
SELECT SID, 'A' AS GRADE
FROM   RESULTS
WHERE  CAT = 'M' AND ENO = 1 AND POINTS >= 12
UNION
SELECT SID, 'B' AS GRADE
FROM   RESULTS
WHERE  CAT = 'M' AND ENO = 1
AND    POINTS >= 10 AND POINTS < 12
UNION
...
```


Vereinigung/Union (4)

- UNION war schon im ersten SQL-Standard (SQL-86) enthalten und wird in allen DBMS unterstützt.
- Es gibt keine andere Möglichkeit eine Vereinigung zu formulieren.

  Aber die SQL-92 Verbundoperatoren sind wiederum nicht notwendig.
- UNION ist in SQL als Algebraoperator etwas seltsam.

  In dem theoretischen "relationalen Tupelkalkül", auf dem SQL basiert, ist es möglich, Tupelvariablen zu erklären, die nicht an eine spezifische Relation gebunden sind. Dann kann man z.B. eine Diskunktion verwenden, um über Tupel zu sprechen, die in einer von zwei oder mehreren Relationen enthalten sind. Dies läßt aber auch "unsichere" Anfragen zu, die schwierig auszuschließen sind. Deshalb wurde diese Möglichkeit in SQL entfernt. Der Preis dafür war, dass man den "fremden" UNION-Operator zufügen musste.

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)

- Muss sich die Anfrage nichtmonoton verhalten, ist klar, dass obige Operationen nicht ausreichen und man die Mengendifferenz "$-$" verwenden muss. Z.B.
  - ◇ 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 (3)

STUDENTS			
SID	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

RESULTS			
SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

EXERCISES			
CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

Mengendifferenz (4)

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

  $\texttt{NO_SOL} := \pi_{\texttt{SID}}(\texttt{STUDENTS}) - \pi_{\texttt{SID}}(\texttt{RESULTS})$;

  $\pi_{\texttt{FIRST},\texttt{LAST}}(\texttt{STUDENTS} \bowtie \texttt{NO_SOL})$

- Übung: Wo ist der Fehler in dieser Anfrage?

  $$\pi_{\texttt{SID},\texttt{FIRST},\texttt{LAST}}(\texttt{STUDENTS}) - \pi_{\texttt{SID}}(\texttt{RESULTS})$$

- Ist dies die richtige Lösung?

  $$\pi_{\texttt{FIRST},\texttt{LAST}}(\texttt{STUDENTS} \underset{\texttt{SID}\neq\texttt{SID2}}{\bowtie} \pi_{\texttt{SID2}\leftarrow\texttt{SID}}(\texttt{RESULTS}))$$

Mengendifferenz (5)

- Wenn man $-$ verwendet, ist der Anti-Verbund 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-Verbund verbreitet, man könnte $R \,\overline{\ltimes}\, S$ verwenden.

Mengendifferenz (6)

- 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:

  $$\pi_{\texttt{SID}}(\sigma_{\texttt{CAT}=\texttt{'H'} \wedge \texttt{ENO}=2}(\texttt{RESULTS})) - \pi_{\texttt{SID}}(\sigma_{\texttt{CAT}=\texttt{'H'} \wedge \texttt{ENO}=1}(\texttt{RESULTS}))$$

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

Mengendifferenz (7)

- Seien R und S in SQL wie folgt dargestellt:
  - ◇ `SELECT` A_1, ..., A_n `FROM` R_1, ..., R_m `WHERE` φ_1
  - ◇ `SELECT` B_1, ..., B_n `FROM` S_1, ..., S_k `WHERE` φ_2
- Dann kann man $R - S$ darstellen als

```
SELECT A1, ..., An
FROM   R1, ..., Rm
WHERE  φ1 AND NOT EXISTS
       (SELECT * FROM S1, ..., Sk
        WHERE  φ2
        AND    B1 = A1 AND ... AND Bn = An)
```


Mengendifferenz (8)

- Die `NOT EXISTS`-Bedingung ist wahr, wenn die Unteranfrage 0 Antworten zurückliefert.

  Unteranfragen werden detaillierter in Kapitel 8 (SQL II) erklärt. Die Unteranfrage wird einmal für jedes Tupel $A_1, \ldots, A_n$ der Hauptanfrage ausgewertet. Die Unteranfrage gibt nur dann ein nichtleeres Ergebnis, wenn die zweite Anfrage das geiche Tupel berechnen kann.

- Wenn man "`NOT EXISTS`" in SQL verwendet, erhält man automatisch den Anti-Verbund: Es ist nicht notwendig Attribute wegzuprojiziren, und sie hinterher durch einen Verbund wiederherzustellen.
- SQL-92 hat auch "`EXCEPT`", das man wie `UNION` verwenden kann. Z.B. nicht in Oracle 8 (dort "`MINUS`").

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 6-4:

- STUDENTS(<u>SID</u>, FIRST, LAST, EMAIL^o)
- EXERCISES(<u>CAT</u>, <u>ENO</u>, TOPIC, MAXPT)
- RESULTS(<u>SID</u>→STUDENTS, (<u>CAT</u>, <u>ENO</u>)→EXERCISES, POINTS)

Vereinigung vs. Verbund

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

Results_1

STUDENT	H	M	F
Jim Ford	95	60	75
Ann Lloyd	80	90	95

Results_2

STUDENT	CAT	PCT
Jim Ford	H	95
Jim Ford	M	60
Jim Ford	F	75
Ann Lloyd	H	80
Ann Lloyd	M	90
Ann Lloyd	F	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/Union
- $-$: Mengendifferenz

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

Inhalt

1. Einführung, Selektion, Projektion

2. Kartesisches Produkt, Verbund

3. Mengenoperationen

4. äußerer Verbund

5. Formale Definitionen, etwas Theorie

Äußerer Verbund (1)

- Der gewöhnliche 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 bewahrt 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)

$R \overset{\rtimes}{}_{A=B} S$:

```
(1)   Create new temporary relation T;
(2)   foreach tuple t in R do
(3)       HasJoinPartner := false;
(4)       foreach tuple u in S do
(5)           if t.A = u.B then
(6)               insert t ∘ u into T;
(7)               HasJoinPartner := true;
(8)           fi;
(9)       od;
(10)      if not HasJoinPartner then
(11)          insert t ∘ (null, ..., null) into T;
(12)  od;
(13)  return T;
```


Äußerer Verbund (4)

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

$$\texttt{STUDENTS} \rtimes \pi_{\texttt{SID,ENO,POINTS}}(\sigma_{\texttt{CAT='H'}}(\texttt{RESULTS}))$$

SID	FIRST	LAST	EMAIL	ENO	POINTS
101	Ann	Smith	···	1	10
101	Ann	Smith	···	2	8
102	Michael	Jones	(null)	1	9
102	Michael	Jones	(null)	2	9
103	Richard	Turner	···	1	5
104	Maria	Brown	···	(null)	(null)

Äußerer Verbund (5)

- Übung: Gibt es irgendeinen Unterschied zwischen
  - ◇ STUDENTS ⋈ RESULTS und
  - ◇ STUDENTS ⟕ RESULTS?
- 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.

  Notwendig in MySQL (hat keine Unteranfragen).

Äußerer Verbund (6)

- Der äußere Verbund ist eine abgeleitete Operation (wie ⋈, ∩), 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 $R \rtimes S$ eine Abkürzung für

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

(wobei ⋈ noch durch ×, 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

1. Einführung, Selektion, Projektion

2. Kartesisches Produkt, Verbund

3. Mengenoperationen

4. äußerer Verbund

5. Formale Definitionen, etwas Theorie

Definitionen: Syntax (1)

Sei folgendes gegeben:

- Eine Menge $\mathcal{S}_\mathcal{D}$ von Datentypnamen, und für jedes $D \in \mathcal{S}_\mathcal{D}$ eine Menge $val(D)$ von Werten.

  Wie zuvor schon bemerkt, unterscheiden wir zur Einfachheit nicht zwischen Konstanten und ihren Werten.

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

  (Integritätsbedingungen sind hier nicht wichtig.)

Definitionen: Syntax (2)

- 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 $d_i \in val(D_i)$ 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 (3)

- Rekursive Fälle, Teil 1: Sei Q ein RA-Ausdruck mit Schema $\rho = (A_1{:}\, D_1, \ldots, A_n{:}\, D_n)$. 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 $d \in val(D_i)$. 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 $B_1, \ldots, B_m \in \mathcal{A}$, sodass $B_j \neq B_k$ für $j \neq k$. Mit Schema $(B_1{:}\, D_{i_1}, \ldots, B_m{:}\, D_{i_m})$.

Definitionen: Syntax (4)

- 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 $(A_1{:}\, D_1, \ldots, A_n{:}\, D_n)$ bzw. $(B_1{:}\, E_1, \ldots, B_n{:}\, E_m)$. 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 (5)

- 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 Rgelen 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 (ohne Restriktionen über erklärte Attributnamen).

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 komlexeren Regeln ist es möglich, die Anzahl der Klammern zu verringern. Man kann auch Bindungsstärken definieren (Operatorvorränge), z.B. bindet $\times$ stärker als $\cup$. Dies ist aber für theoretische Untersuchungen nicht wichtig.

- Wie oben erklärt kann man zusätzliche Algebraoperationen (z.B. Verbund) 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 erweitert werden können.

Definitionen: Semantik (1)

- Ein DB-Zustand $\mathcal{I}$ definiert eine endliche Relation $\mathcal{I}(R)$ für jeden Relationsnamen R im DB-Schema.
  Ist $sch(R) = (A_1\colon D_1, \ldots, A_n\colon D_n)$, dann $\mathcal{I}(R) \subseteq val(D_1) \times \cdots \times val(D_n)$.
- 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 $\mathcal{I}[Q] := \mathcal{I}(R)$.
  - ◇ Ist Q eine 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 Ergebnisses $\mathcal{I}[Q]$ eines RA-Ausdrucks Q im Zustand $\mathcal{I}$, fortgesetzt:
  - ◇ 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
    $$\mathcal{I}[Q] := \{(d_1, \ldots, d_n, e_1, \ldots, e_m) \mid (d_1, \ldots, d_n) \in \mathcal{I}[Q_1], (e_1, \ldots, e_m) \in \mathcal{I}[Q_2]\}.$$

Monotonie

- Definition: Ein DB-Zustand $\mathcal{I}_1$ ist kleiner als (oder gleich groß wie) ein DB-Zustand $\mathcal{I}_2$, geschrieben $\mathcal{I}_1 \subseteq \mathcal{I}_2$, genau dann wenn $\mathcal{I}_1(R) \subseteq \mathcal{I}_2(R)$ für alle Relationsnamen R im Schema.
- Theorem: Enthält ein RA-Ausdruck Q nicht den Mengendifferenz-Operator $-$, dann gilt für alle DB-Zustände $\mathcal{I}_1, \mathcal{I}_2$:
  $$\mathcal{I}_1 \subseteq \mathcal{I}_2 \implies \mathcal{I}_1[Q] \subseteq \mathcal{I}_2[Q].$$
- Übung: Beweisen Sie das Theorem durch Induktion über die Struktur von Q ("strukturelle Induktion").

Äquivalenz (1)

- Definition: Zwei RA-Ausdrücke Q_1 und Q_2 sind äquivalent, genau dann wenn sie das gleiche Schema haben, und für alle DB-Zustände $\mathcal{I}$ gilt:

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

- Es gibt zwei Begriffsinterpretationen für die 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. Im folgenden ist es nicht wichtig, welche Alternative gewählt wird.

Äquivalenz (2)

- Beispiele für Äquivalenzen:
  - $\sigma_{\varphi_1}(\sigma_{\varphi_2}(Q))$ ist äquivalent zu $\sigma_{\varphi_2}(\sigma_{\varphi_1}(Q))$.
  - $(Q_1 \times Q_2) \times Q_3$ ist äquivalent zu $Q_1 \times (Q_2 \times Q_3)$.
  - Ist A ein Attribut im Schema von Q_1, dann ist $\sigma_{A=d}(Q_1 \times Q_2)$ äquivalent zu $(\sigma_{A=d}(Q_1)) \times Q_2$
- Theorem: Die Äquivalenz von RA-Ausdrücken ist unentscheidbar.

  D.h. man kann kein Programm schreiben, dass zwei gewöhnliche RA-Ausdrücke Q_1 und Q_2 liest, und "ja" oder "nein" ausgibt, abhängig davon ob Q_1 und Q_2 äquivalent sind, und das garantiert nach endlicher Rechenzeit anhält.

Einschränkungen der RA (1)

- R sei ein ein Relationsname mit Schema (A: D, B: D) und $val(D)$ sei endlich.
- Der transitive Abschluß von $\mathcal{I}(R)$ ist die Menge aller $(d, e) \in val(D) \times val(D)$, sodass es ein $n \in \mathbb{N}$ ($n \geq 1$) und $d_0, \ldots, d_n \in val(D)$ 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 "`PARENT`" sein, dann besteht der transitive Abschluß aus allen Ahnenbeziehungen (Eltern, Großeltern, Groß-Großeltern, ...).

Einschränkungen der RA (2)

- Theorem: Es gibt keinen RA-Ausdruck Q, sodass $\mathcal{I}[Q]$ der transitive Abschluß von $\mathcal{I}(R)$ ist für alle DB-Zustände $\mathcal{I}$.
- Z.B. braucht man in dem Ahnenbeispiel einen zusätzlichen Verbund für jede weitere Generation.
- Deshalb kann man keine Anfrage schreiben, die bei allen DB-Zuständen funktioniert, wenn man nicht weiß, wieviele Generationen die Datenbank umfasst.

  Natürlich kann man eine Anfrage schreiben, die z.B. bis zu den Groß-Großeltern funktioniert. Aber dann funktioniert es nicht korrekt, falls die Datenbank Groß-Groß-Großeltern enthält.

Einschränkungen der RA (3)

- Das impliziert natürlich, dass die relationale Algebra nicht computationally vollständig ist:
  - ◇ Nicht jede Funktion von Datenbankzuständen zu Relationen, für die ein C-Programm existiert, kann in relationaler Algebra formuliert werden.
  - ◇ Dies kann auch nicht erwartet werden, da man sichern möchte, dass die Anfrageauswertung immer beendet wird. Dies ist für die RA gesichert.

    Es ist unentscheidbar, ob allgemeine Programme enden ("Halteproblem"). Jede computationally vollständige Sprache muss notwendigerweise zulassen, dass Anfragen/Programme formuliert werden, die eventuell nicht enden. Dies kann vorher nicht überprüft werden.

Einschränkungen 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.

  Z.B. Wenn Sie einen Weg finden, dass Travelling Salesman Problem in relationaler Algebra zu formulieren, haben Sie das berühmte P=NP Problem gelöst (mit einer Lösung, die niemand erwartet). Da dies sehr unwahrscheinlich ist, sollten Sie es nicht probieren, sondern ein C-Programm schreiben.
- Aber man kann nicht alle Probleme von polynomialer Komplexität in RA formulieren.

  Vgl. transitive Komplexität. Mit einem Fixpunktoperator und einer linearen Ordnung auf dem Bereich ist dies möglich (→ Deduktive DB).

Ausdruckskraft (1)

- Eine Anfragesprache $\mathcal{L}$ für das relationale Modell nennt man stark relational vollständig, wenn für jedes DB-Schema $\mathcal{S}$ und für jeden RA-Ausdruck Q_1, der $\mathcal{S}$ betrifft, es eine Anfrage gibt $Q_2 \in \mathcal{L}_{\mathcal{S}}$, sodass für alle DB-Zustände $\mathcal{I}$, die $\mathcal{S}$ betreffen, die zwei Anfragen das gleiche Ergebnis liefern: $\mathcal{I}[Q_1] = \mathcal{I}[Q_2]$.
- D.h. die Forderung ist, dass jeder RA-Ausdruck in eine äquivalente Anfrage in dieser Sprache übersetzt werden kann.

Ausdruckskraft (2)

- Z.B. SQL ist stark 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 "stark") gestattet Folgen von Anfragen zu verwenden und Zwischenergebnisse in zeitweiligen Relationen zu speichern.

Ausdruckskraft (3)

- Die folgenden Sprachen haben die gleiche Ausdruckskraft:
  - ◇ Relationale Algebra
  - ◇ SQL ohne Aggregationen und mit verbindlicher Duplikateliminierung.
  - ◇ relationales Tupelkalkül (First-Order-Logik mit Variablen für Tupel, siehe unten), Domain RK
  - ◇ Datalog (eine Prolog-Variante) ohne Rekursion
- Deshalb ist die Menge der Funktionen, die in RA ausgedrückt werden können, nicht willkürlich.