# Teil 7: 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 "Verbund-Selektion-Projektion" schreiben.

  Zusätzlich einige Anfragen schreiben, die 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.

# 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.
- Eine Operation der relationalen Algebra ist $\cup$ (Vereinigung). Klar, weil Relationen Mengen sind.

# Relationale Algebra (3)

- Eine weitere Operation der RA ist die Selektion.

  Im Gegensatz zu Operationen wie + für Integers, ist die Selektion $\sigma$ 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:

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

# 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 7-115.

# Selektion (1)

- Die Operation $\sigma_\varphi$ selektiert eine Teilmenge der Tupel einer Relation, nämlich die, die die Bedingung $\varphi$ erfüllen. Die Selektion wirkt wie ein Filter auf die Eingabemenge.

  $\sigma$ ist der griechische Buchstabe sigma, $\varphi$ der griech. Buchstabe phi. Alle Lehrbücher verwenden $\sigma$ für Selektion, aber $\varphi$ 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 $\varphi$ hat die folgende Form:

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

  Die Bedingung liefert einen Wahrheitswert (wahr oder falsch) für ein gegebenes Eingabetupel. Vergleichsoperatoren: $=$, $\neq$, $<$, $\leq$, $>$, $\geq$ (siehe nächste Folie). Allgemein sind beliebige atomare Formeln zulässig.

- Ein Term ("Wertausdruck") ist:
  - ein Attributname,
  - eine Datentypkonstante, oder
  - ein durch Datentypoperationen wie $+$, $-$, $*$, $/$ aus Teiltermen zusammengesetzten Ausdruck.

    Er kann für ein gegebenes Tupel zu einem Datentypelement ausgewertet werden.

# 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:
  - `NACHNAME = 'Smith'`
  - `PUNKTE >= 8`
  - `PUNKTE = 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 Datenstrukturen (B-Baum Index) kann es möglich sein, $\sigma_\varphi(R)$ zu berechnen, ohne jedes Tupel aus der Eingaberelation durchzugehen.

# 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 legal, aber unnötig kompliziert ($\sigma$ mit immer wahrer Bedingung ist überflüssig):

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

# Selektion (6)

- Es ist kein Fehler, wenn das Ergebnis eines RA-Ausdrucks in einem gegebenen 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 dagegen ein (semantischer) Fehler, eine Bedingung zu verwenden, die immer falsch ist:

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

# 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 grundlegenden Selektions-Operation sind nur einfache Vergleiche ("atomare Formeln") als Bedingungen möglich.
- Man kann aber auch die Kombination einfacher Vergleiche durch die logischen Junktoren $\wedge$ (und), $\vee$ (oder), und $\neg$ (nicht) zulassen:

| $\varphi 1$ | $\varphi 2$ | $\varphi 1 \wedge \varphi 2$ | $\varphi 1 \vee \varphi 2$ | $\neg \varphi 1$ |
|---|---|---|---|---|
| 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 $\varphi_1$ und $\varphi_2$"
- $\varphi_1 \vee \varphi_2$ nennt man "Disjunktion von $\varphi_1$ und $\varphi_2$"
- $\neg\varphi_1$ nennt man "Negation von $\varphi_1$".
- Man kann auch "`and`", "`or`" und "`not`" 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 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 Elimination 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 7-4:

- STUDENTEN(SID, VORNAME, NACHNAME, EMAIL$^O$)
- AUFGABEN(ATYP, ANR, THEMA, MAXPT)
- BEWERTUNGEN(SID$\longrightarrow$STUDENTEN, (ATYP, ANR)$\longrightarrow$AUFGABEN, PUNKTE)

# Projektion (1)

- Die Projektion $\pi$ eliminiert Attribute (Spalten) aus der Eingaberelation.

  $\pi$ ist der griechische Buchstabe "pi".
  In Datenbanken bedeutet dies immer "Projektion", und nicht 3.14....
  Einige Autoren verwenden ein großes $\Pi$ zur Unterscheidung.
  In ASCII kann man z.B. schreiben: `PRO[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 erstellt die Projektion $\pi_{A_{i_1},\ldots,A_{i_k}}(R)$ für jedes Eingabetupel $(A_1 : d_1, \ldots, A_n : d_n)$ ein Ausgabetupel $(A_{i_1} : d_{i_1}, \ldots, A_{i_k} : d_{i_k})$.

  Während $\sigma$ gewisse Zeilen der Eingaberelation auswählt, und die anderen wegfallen läßt, wählt $\pi$ gewisse Spalten aus, und ignoriert die anderen. Die Eingabetabelle selbst wird natürlich nicht geändert.

- 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;
```

- Diese Programmskizze 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 $(A_1 : d_1, \ldots, A_n : d_n)$ in das Ausgabetupel $(B_1 : d_{i_1}, \ldots, B_k : d_{i_k})$.
  - ◇ Rückgabewerte können durch Datentypoperationen, wie $+$ oder || (Stringkonkatenation), berechnet werden:
    $$\pi_{\texttt{SID},\ \texttt{NAME} \leftarrow \texttt{VORNAME} \,||\, \texttt{' '} \,||\, \texttt{NACHNAME}}(\texttt{STUDENTEN}).$$
  - ◇ Spalten können mit Konstanten erstellt werden:
    $$\pi_{\texttt{SID, VORNAME, NACHNAME, NOTE} \leftarrow \texttt{'1'}}(\texttt{STUDENTEN}).$$

# Projektion (6)

- 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 (7)

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

```
SELECT DISTINCT A_1, ..., A_n
FROM   R
```

- Das Schlüsselwort `DISTINCT` ist nicht immer nötig.

  Die Duplikateliminierung kostet Laufzeit, daher sollte man `DISTINCT` nur angeben, wenn es nötig ist. `DISTINCT` ist unnötig, wenn $A_1, \ldots, A_n$ einen Schlüssel enthält. Manchmal sind Duplikate auch erwünscht.

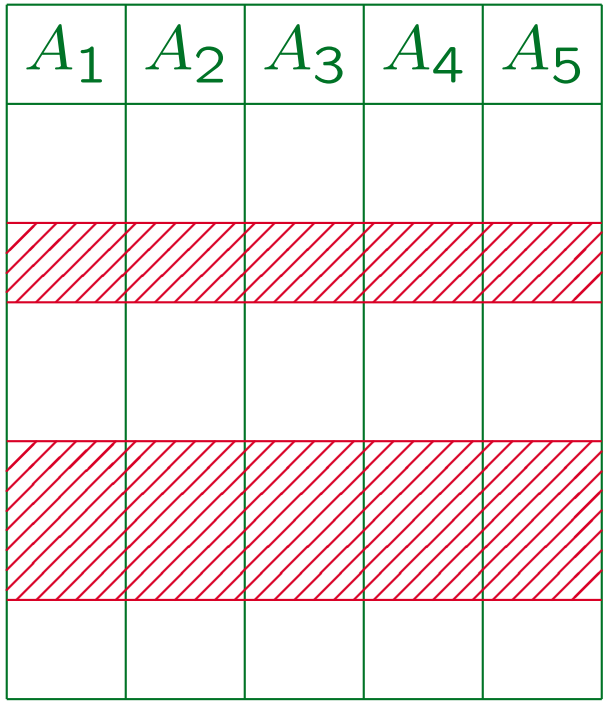
- $\pi_{B_1 \leftarrow A_1,\ldots,B_n \leftarrow A_n}(R)$ schreibt man in SQL wie folgt:

```
SELECT DISTINCT A_1 AS B_1, ..., A_n AS B_n
FROM   R
```

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

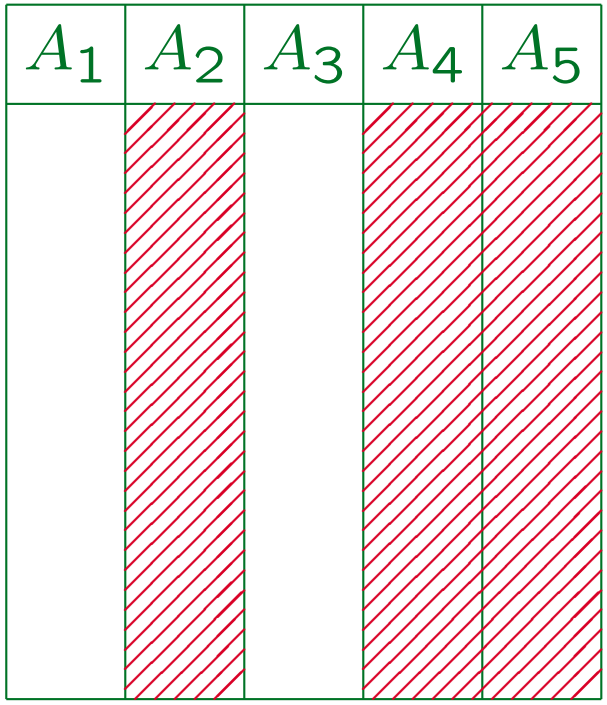
# Zusammenfassung

Selektion $\sigma$




(Filtert Eingabezeilen)

Projektion $\pi$




(wendet Funktion auf jede Zeile an)

# 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:

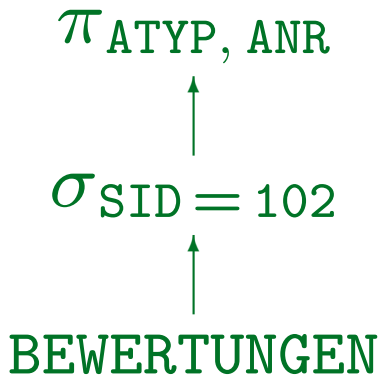
$$\pi_{\texttt{ATYP, ANR}}(\sigma_{\texttt{SID}=102}(\texttt{BEWERTUNGEN}))$$

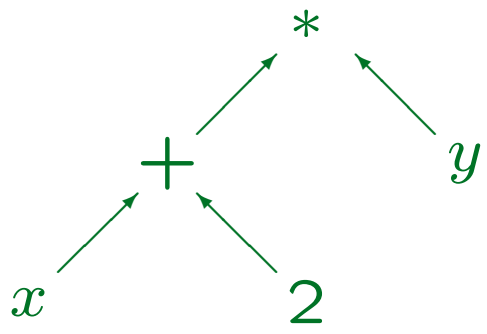
- Man kann ein Zwischenergebnis in einer temporären Relation speichern (andere Sicht: Makro-Definition).

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

# 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.

# Operationen schachteln (3)

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

```
SELECT ATYP, ANR
FROM   (SELECT *
        FROM   BEWERTUNGEN
        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 schachteln (4)

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

```
SELECT ATYP, ANR
FROM   BEWERTUNGEN
WHERE  SID = 102
```

- Komplexe Anfragen kann man Schritt für Schritt konstruieren:

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

- Dann kann man S102 wie eine Tabelle verwenden.

# Basisoperatoren

- 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.
- Dies entspricht der SQL-Anfrage:

```
SELECT *
FROM   R
```

# 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 7-4:

- STUDENTEN(SID, VORNAME, NACHNAME, EMAIL$^O$)
- AUFGABEN(ATYP, ANR, THEMA, MAXPT)
- BEWERTUNGEN(SID$\rightarrow$STUDENTEN, (ATYP, ANR)$\rightarrow$AUFGABEN, PUNKTE)

# Übungen (2)

- Welche der folgenden Ausdrücke der relationalen Algebra sind syntaktisch korrekt?
  Was bedeuten sie?

  - ☐ $\texttt{STUDENTEN}$.
  - ☐ $\sigma_{\texttt{MAXPT} \neq 10}(\texttt{AUFGABEN})$.
  - ☐ $\pi_{\texttt{VORNAME}}(\pi_{\texttt{NACHNAME}}(\texttt{STUDENTEN}))$.
  - ☐ $\sigma_{\texttt{PUNKTE} \leq 5}(\sigma_{\texttt{PUNKTE} \geq 1}(\texttt{BEWERTUNGEN}))$.
  - ☐ $\sigma_{\texttt{PUNKTE}}(\pi_{\texttt{PUNKTE} = 10}(\texttt{BEWERTUNGEN}))$.

# Inhalt

# Kartesisches Produkt (1)

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

  Für $\sigma$ und $\pi$ gilt, daß jedes Ausgabetupel von einem einzigen Eingabetupel abgeleitet ist. Da $\pi$ 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 `(...)` 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 ($\pi$).

# 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 $(R.A, R.B, S.B, S.C)$.
  - ◇ 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.

# 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 nützlich:

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

- 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 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 $\bowtie$ 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 $\sigma_{A=B}(R \times S)$:

```
(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.

  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 $\sigma$.

- 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.

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

  Die Umbenennung des Attributs "SID" ist notwendig, da das kartesische Produkt disjunkte Attributnamen verlangt. Aber aber 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 |

$\overset{\bowtie}{}_{B=C}$

| $C$ | $D$ |
|---|---|
| 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 Vebund 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.
- Z.B., wenn $R(A, B)$, und $S(B, C)$ gegeben ist, dann ist $R \bowtie S$ eine Abkürzung für

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

# 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.
- 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 schon immer die Operation `UNION` der relationalen Algebra hatte, basiert es sonst stärker auf dem logischen Formalismus des Tupelkalküls. Die neue Sytax wurde wahrscheinlich nur eingeführt, da der "äußere Verbund" (siehe unten) auf klassische Weise schwieriger zu formulieren ist.

- Z.B. wurde die neue alternative Syntax des Verbunds bis Oracle 8i nicht unterstützt (aber in 9i).

# 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 $\pi$ auch für diesen Zweck einführen):

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

# Algebraische Gesetze (2)

- Vom Anfrageoptimierer eines relationalen DBMS werden weitere algebraische Gesetze verwendet.
- Z.B. wenn sich $\varphi$ 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. $\pi$ auf alle Spalten), wird es die volle Punktzahl geben.

# 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 komplizieren, indem man eine Projektion auf alle Attribute oder eine Selektion mit einer stets wahren Bedingung verwendet (die einfach die Identitätsabbildung ist).

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

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

# Häufiges Anfragemuster (3)

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

  SELECT DISTINCT $A_1, \ldots, A_k$
  FROM $R_1, \ldots, R_n$
  WHERE $\varphi$ AND ⟨Verbund-Bedingungen⟩

- 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.

- Normalerweise werden Relationen durch Gleichungen verknüpft, z.B. $R_1.B_1 = R_2.B_2$.

- "DISTINCT" wird nicht immer benötigt (siehe oben).

# Häufiges 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.

# Häufiges 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 $R \bowtie S \bowtie T$.
  - ◇ 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 7-4:

- STUDENTEN(SID, VORNAME, NACHNAME, EMAIL$^O$)
- AUFGABEN(ATYP, ANR, THEMA, MAXPT)
- BEWERTUNGEN(SID$\rightarrow$STUDENTEN, (ATYP, ANR)$\rightarrow$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).

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

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

- 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)

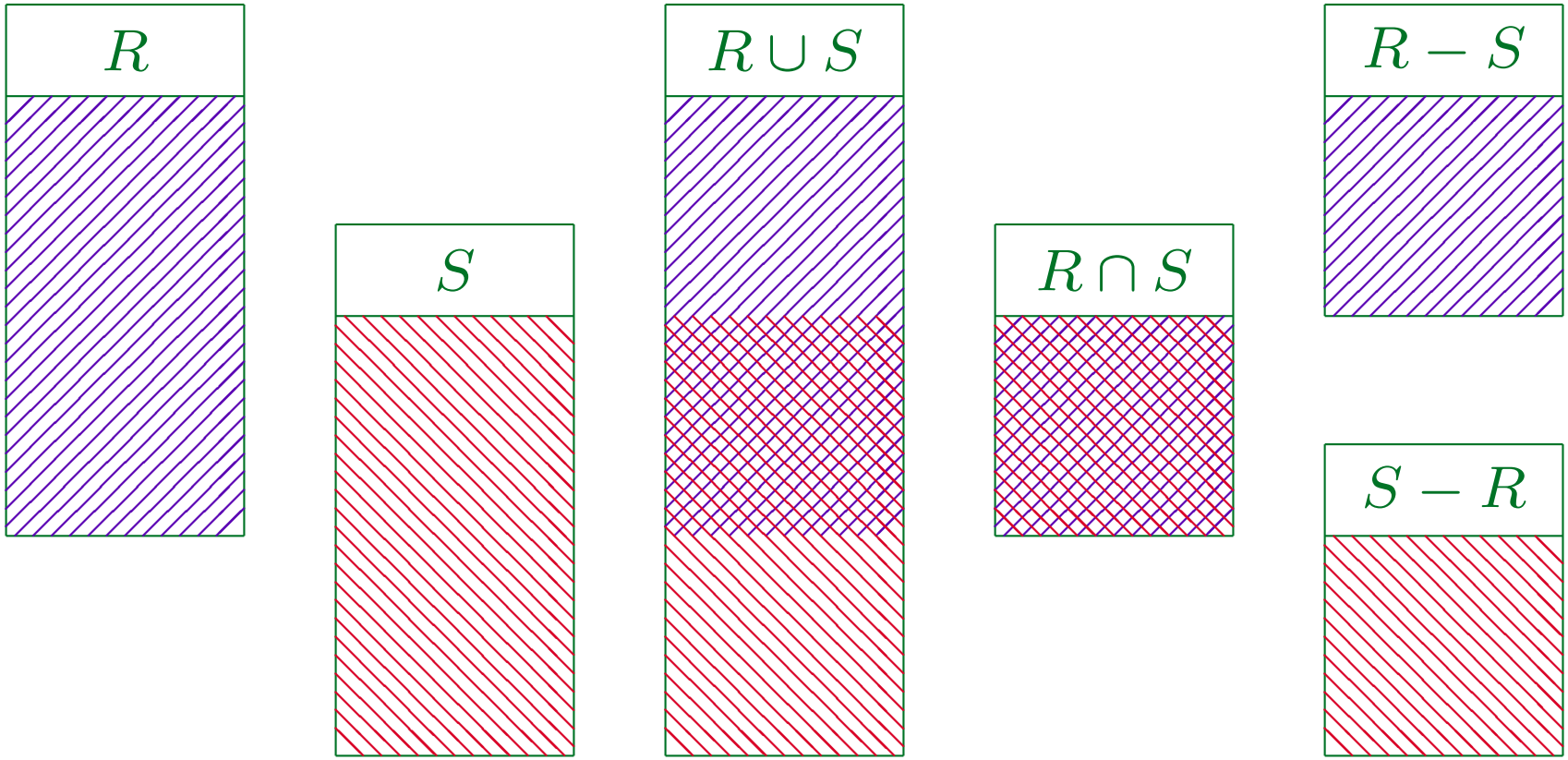
- $R - S$ enthält alle Tupel, die in $R$, aber nicht in $S$ sind (Mengendifferenz/Set Difference).
- $R \cap S$ enthält Tupel, die in beiden, $R$ und $S$, sind (Durchschnitt/Intersection).
- Der Durchschnitt ist (wie der Verbund) eine abgeleitete Operation: Er kann durch $-$ ausgedrückt werden:

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

- Übung: Beweisen Sie diese Gleichung.

  Z.B. Zeichnen Sie ein Venn Diagramm.

# Mengenoperationen (3)



# Mengenoperationen (4)

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

```
(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 (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(VORNAME, NACHNAME, 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:

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

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

# Vereinigung (3)

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

```
SELECT SID, '1' AS NOTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'Z' AND ANR = 1 AND PUNKTE >= 12
UNION
SELECT SID, '2' AS NOTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'Z' AND ANR = 1
AND    PUNKTE >= 10 AND PUNKTE < 12
UNION
...
```

# Vereinigung (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 deklarieren, die nicht an eine spezifische Relation gebunden sind. Dann kann man z.B. eine Disjunktion verwenden, um über Tupel zu sprechen, die in einer von zwei 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 zu SQL hinzufügen mußte.

# Mengendifferenz (1)

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

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

- 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)

- 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 (3)

**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 |

# Mengendifferenz (4)

- 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 (5)

- 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 (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:

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

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

# Mengendifferenz (7)

- Seien $R$ und $S$ in SQL wie folgt dargestellt:
  - SELECT $A_1, \ldots, A_n$ FROM $R_1, \ldots, R_m$ WHERE $\varphi_1$
  - SELECT $B_1, \ldots, B_n$ FROM $S_1, \ldots, S_k$ WHERE $\varphi_2$
- Dann kann man $R - S$ darstellen als

```
SELECT A_1, ..., A_n
FROM   R_1, ..., R_m
WHERE  φ_1 AND NOT EXISTS
       (SELECT * FROM S_1, ..., S_k
        WHERE  φ_2
        AND    B_1 = A_1 AND ... AND B_n = A_n)
```

# Mengendifferenz (8)

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

  Unteranfragen werden detaillierter in Kapitel 9 (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-Join: Es ist nicht notwendig Attribute wegzuprojizieren, und sie hinterher durch einen Verbund wiederherzustellen.
- SQL-92 hat auch "`EXCEPT`", das man wie `UNION` verwenden kann. Aber 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 7-4:

- STUDENTEN(SID, VORNAME, NACHNAME, EMAIL$^O$)
- AUFGABEN(ATYP, ANR, THEMA, MAXPT)
- BEWERTUNGEN(SID$\rightarrow$STUDENTEN, (ATYP, ANR)$\rightarrow$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:

- $\sigma_\varphi$: 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 $\rho$, 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:

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |

$\overset{\text{left}}{\bowtie}$

| $B$ | $C$ |
|---|---|
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

$=$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | |
| $a_2$ | $b_2$ | $c_2$ |

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

# Äußerer Verbund (2)

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

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |

$\overset{\subset}{\bowtie}$

| $B$ | $C$ |
|---|---|
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

$=$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_2$ | $b_2$ | $c_2$ |
| | $b_3$ | $c_3$ |

- Der volle äußere Verbund eliminiert gar kein Tupel:

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |

$\overset{\subset}{\bowtie}$

| $B$ | $C$ |
|---|---|
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

$=$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | |
| $a_2$ | $b_2$ | $c_2$ |
| | $b_3$ | $c_3$ |

# Äußerer Verbund (3)

$R \underset{A=B}{\bowtie} S$:

```
(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:

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

| 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 $\bowtie$ BEWERTUNGEN und
  - ◇ STUDENTEN $\ltimes$ 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.

  Notwendig in MySQL (hat keine Unteranfragen).

# 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 $R \ltimes S$ eine Abkürzung für

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

(wobei $\bowtie$ noch durch $\times$, $\sigma$, $\pi$ 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: 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:
  - $\sigma_{A_i=A_j}(Q)$ für $i, j \in \{1, \ldots, n\}$. Mit Schema: $\rho$.
  - $\sigma_{A_i=d}(Q)$ für $i \in \{1, \ldots, n\}$ und $d \in val(D_i)$. Mit Schema: $\rho$.
  - $\pi_{B_1 \leftarrow A_{i_1}, \ldots, B_m \leftarrow A_{i_m}}(Q)$ 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 $(B_1 : D_{i_1}, \ldots, B_m : D_{i_m})$.

# Definitionen: Syntax (4)

- Rekursive Fälle, fortgesetzt: Seien $Q_1$ und $Q_2$ RA-Ausdrücke mit gleichem Schema $\rho$. Dann sind auch die folgenden RA-Ausdrücke mit dem Schema $\rho$:
  - $\boxed{(Q_1) \cup (Q_2)}$ 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 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)

- Ein DB-Zustand $\mathcal{I}$ definiert eine endliche Relation $\mathcal{I}[R]$ für jeden Relationsnamen $R$ des Schemas.

  Ist $sch(R) = (A_1 : D_1, \ldots, A_n : D_n)$, 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 $Q$ die konstante Relation $\{(A_1 : d_1, \ldots, A_n : d_n)\}$, 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
    $$\mathcal{I}[Q] := \{(d_1, \ldots, d_n, e_1, \ldots, e_m) \mid (d_1, \ldots, d_n) \in \mathcal{I}[Q_1], (e_1, \ldots, e_m) \in \mathcal{I}[Q_2]\}.$$

# Monotonie

- Definition: Ein DB-Zustand $\mathcal{I}_1$ ist kleiner als ein DB-Zustand $\mathcal{I}_2$ (oder gleich), geschrieben $\mathcal{I}_1 \subseteq \mathcal{I}_2$, gdw. $\mathcal{I}_1(R) \subseteq \mathcal{I}_2(R)$ 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 $Q_2$ heißen 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 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. 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 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.

# Grenzen der RA (1)

- $R$ sei ein Relationsname mit Schema $(A\colon D, B\colon D)$ und $val(D)$ sei endlich.
- Der transitive Abschluß von $\mathcal{I}[R]$ ist die Menge aller $(d, e) \in val(D) \times val(D)$, so daß 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 "`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 $(\text{Ur})^3$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 mit verbindlicher Duplikateliminierung.
  - ◇ Tupelkalkül (Logik 1. Stufe, Variablen für Tupel)
  - ◇ Bereichskalkül (Logik 1. Stufe, Variablen für Datenwerte)
  - ◇ Datalog (eine Prolog-Variante) ohne Rekursion
- Deshalb ist die Menge der Funktionen, die in RA ausgedrückt werden können, nicht willkürlich.