Christian-Albrechts-Universität zu Kiel

Skript zur Vorlesung

Informationssysteme

Wintersemester 2018/19

Kapitel 3: Die Relationale Algebra

Arbeiten mit Relationen

- Es gibt viele *formale* Modelle, um...
  - mit Relationen zu arbeiten
  - Anfragen zu formulieren
- Wichtigste Beispiele:
  - Relationale Algebra
  - Relationen-Kalkül
- Sie dienen als theoretisches Fundament für konkrete Anfragesprachen wie
  - SQL: Basiert i.w. auf der relationalen Algebra
  - QBE (= Query By Example) und Quel: Basieren auf dem Relationen-Kalkül

Begriff Relationale Algebra

- Mathematik:
  - Algebra ist eine Operanden-Menge mit Operationen
  - Abgeschlossenheit: Werden Elemente der Menge mittels eines Operators verknüpft, ist das Ergebnis wieder ein Element der Menge
  - Beispiele
    - Natürliche Zahlen mit Addition, Multiplikation
    - Zeichenketten mit Konkatenation
    - Boolsche Algebra: Wahrheitswerte mit $\wedge$, $\vee$, $\neg$
    - Mengen-Algebra:
      - Wertebereich: die Menge (*Klasse*) der Mengen
      - Operationen z.B. $\cup$, $\cap$, – (Differenzmenge)

Begriff Relationale Algebra

- Relationale Algebra:
  - „Rechnen mit Relationen“
  - Was sind hier die Operanden? Relationen (Tabellen)
  - Beispiele für Operationen?
    - Selektion von Tupeln nach Kriterien (z.B. *Gehalt* > 1000)
    - Kombination mehrerer Tabellen
  - Abgeschlossenheit:
    Ergebnis einer Anfrage ist immer eine (neue) Relation (oft ohne eigenen Namen)
  - Damit können einfache Terme der relationalen Algebra zu komplexeren zusammengesetzt werden

Grundoperationen

- 5 Grundoperationen der Relationalen Algebra:
  - Vereinigung: $R = S \cup T$
  - Differenz: $R = S - T$
  - Kartesisches Produkt (Kreuzprodukt): $R = S \times T$
  - Selektion: $R = \sigma_F(S)$
  - Projektion: $R = \pi_{A,B,\ldots}(S)$
- Mit den Grundoperationen lassen sich weitere Operationen, (z.B. die Schnittmenge) nachbilden
- Manchmal wird die Umbenennung von Attributen als 6. Grundoperation bezeichnet

Vereinigung und Differenz

- Diese Operationen sind nur anwendbar, wenn die Schemata der beiden Relationen S und T übereinstimmen (Name *und* Domäne)
- Die Ergebnis-Relation R bekommt dieses Schema
- Vereinigung: $R = S \cup T = \{t \mid t \in S \vee t \in T\}$
- Differenz: $R' = S - T = \{t \mid t \in S \wedge t \notin T\}$
- Was wissen wir über die *Kardinalität* des Ergebnisses (Anzahl der Tupel von R)?

$$|R| = |S \cup T| \leq |S| + |T|$$

$$|R'| = |S - T| \geq |S| - |T|$$

- **Achtung**: Zwei Tupel gelten nur dann als *gleich*, wenn *alle* ihre Attributwerte gleich sind.

Beispiel

Mitarbeiter:

Name	Vorname
Huber	Egon
Maier	Wolfgang
Schmidt	Helmut

Studenten:

Name	Vorname
Müller	Heinz
Schmidt	Helmut

Alle Personen, die Mitarbeiter oder Studenten sind:

Mitarbeiter ∪ Studenten:

Name	Vorname
Huber	Egon
Maier	Wolfgang
Schmidt	Helmut
Müller	Heinz
~~Schmidt~~	~~Helmut~~

Duplikat-Elimination!

Alle Mitarbeiter ohne diejenigen, die auch Studenten sind:

Mitarbeiter – Studenten:

Name	Vorname
Huber	Egon
Maier	Wolfgang

Kartesisches Produkt

Wie in Kapitel 2 bezeichnet das Kreuzprodukt

$$R = S \times T$$

die Menge aller möglichen Kombinationen von Tupeln aus S und T

- Seien $a_1, a_2, ..., a_s$ die Attribute von S und $b_1, b_2, ..., b_t$ die Attribute von T
- Dann ist $R = S \times T$ die folgende Menge (Relation):
  $\{(a_1,...,a_s,b_1,...,b_t) \mid (a_1,...,a_s) \in S \wedge (b_1,...,b_t) \in T\}$
- Für die Anzahl der Tupel gilt:

$$|S \times T| = |S| \cdot |T|$$

Beispiel

Mitarbeiter

PNr	Name	Vorname	Abteilung
001	Huber	Erwin	01
002	Mayer	Hugo	01
003	Müller	Anton	02

Abteilungen

ANr	Abteilungsname
01	Buchhaltung
02	Produktion

Mitarbeiter × Abteilungen

PNr	Name	Vorname	Abteilung	ANr	Abteilungsname
001	Huber	Erwin	01	01	Buchhaltung
001	Huber	Erwin	01	02	Produktion
002	Mayer	Hugo	01	01	Buchhaltung
002	Mayer	Hugo	01	02	Produktion
003	Müller	Anton	02	01	Buchhaltung
003	Müller	Anton	02	02	Produktion

Frage: Ist dies richtig/gewünscht/intuitiv?

Selektion

- Mit der Selektion $R = \sigma_F(S)$ werden diejenigen Tupel aus einer Relation S ausgewählt, die eine durch die logische Formel F vorgegebene Eigenschaft erfüllen
- R bekommt das gleiche Schema wie S
- Die Formel F besteht aus:
  - Konstanten („Meier“)
  - Attributen: Als Name (PNr) oder Nummer ($1)
  - Vergleichsoperatoren: $=$, $<$, $\leq$, $>$, $\geq$, $\neq$
  - Boole‘sche Operatoren: $\wedge$, $\vee$, $\neg$
- Formel F wird für jedes Tupel von S ausgewertet

Beispiel

Mitarbeiter

PNr	Name	Vorname	Abteilung
001	Huber	Erwin	01
002	Mayer	Hugo	01
003	Müller	Anton	02

Alle Mitarbeiter von Abteilung 01:

$\sigma_{\text{Abteilung}=01}(\text{Mitarbeiter})$

PNr	Name	Vorname	Abteilung
001	Huber	Erwin	01
002	Mayer	Hugo	01

Kann jetzt die Frage von S. 9 beantwortet werden?

Beispiel

Mitarbeiter × Abteilungen

PNr	Name	Vorname	Abteilung	ANr	Abteilungsname
001	Huber	Erwin	01	01	Buchhaltung
001	Huber	Erwin	01	02	Produktion
002	Mayer	Hugo	01	01	Buchhaltung
002	Mayer	Hugo	01	02	Produktion
003	Müller	Anton	02	01	Buchhaltung
003	Müller	Anton	02	02	Produktion

$\sigma_{\text{Abteilung=ANr}}$(Mitarbeiter × Abteilungen)

PNr	Name	Vorname	Abteilung	ANr	Abteilungsname
001	Huber	Erwin	01	01	Buchhaltung
002	Mayer	Hugo	01	01	Buchhaltung
003	Müller	Anton	02	02	Produktion

Die Kombination aus Selektion und Kreuzprodukt heißt Join

Projektion

- Die Projektion $R = \pi_{A,B,...}(S)$ erlaubt es,
  - Spalten einer Relation auszuwählen
  - bzw. nicht ausgewählte Spalten zu streichen
  - die Reihenfolge der Spalten zu verändern
- In den Indizes sind die selektierten Attribut-Namen oder -Nummern ($1) aufgeführt
- Für die Anzahl der Tupel des Ergebnisses gilt:

$$| \pi_{A,B,...}(S) | \leq | S |$$

Grund: Nach dem Streichen von Spalten können Duplikat-Tupel entstanden sein, die eliminiert werden, damit wieder eine Relation (Menge von Tupeln) entsteht.

Projektion: Beispiel

Mitarbeiter

PNr	Name	Vorname	Abteilung
001	Huber	Erwin	01
002	Mayer	Josef	01
003	Müller	Anton	02
004	Mayer	Maria	01

$\pi_{\text{Name, Abteilung}}(\text{Mitarbeiter}) = \ldots$

Zwischenergebnis (Multimenge):

Name	Abteilung
Huber	01
Mayer	01
Müller	02
Mayer	01

Duplikate

Elimination →

Endergebnis (Menge):

Name	Abteilung
Huber	01
Mayer	01
Müller	02

Duplikat-Elimination

- Erforderlich nach...
  - Projektion
  - Vereinigung

  } „billige“ Basisoperationen, aber...
- Wie funktioniert Duplikat-Elimination?

```
for (int i = 0 ; i < R.length ; i++)
    for (int j = 0 ; j < i ; j++)
        if (R[i] == R[j])
            // R[j] markieren um es später zu löschen
```

- Aufwand? n=R.length: O(n^2)
- Besserer Algorithmus mit Sortieren: O($n \log n$)

⇒ An sich billige Grund-Operationen werden nur durch die Duplikat-Elimination teuer

Beispiel-Anfragen

Gegeben sei folgendes Relationenschema:

Städte (SName: String, SEinw: Integer, Land: String)

Länder (LName: String, LEinw: Integer, Partei*: String)

* bei Koalitionsregierungen: jeweils eigenes Tupel pro Partei

---

- Bestimme alle Großstädte (≥500.000) und ihre Einwohner

  $\pi_{SName,SEinw}(\sigma_{SEinw \geq 500.000}(Städte))$

- In welchem Land liegt die Stadt Passau?

  $\pi_{Land}(\sigma_{SName=Passau}(Städte))$

- Bestimme die Namen aller Städte, deren Einwohnerzahl die eines beliebigen Landes übersteigt:

  $\pi_{SName}(\sigma_{SEinw>LEinw}(Städte \times Länder))$

Beispiel-Anfragen

Gegeben sei folgendes Relationenschema:

Städte (SName: String, SEinw: Integer, Land: String)

Länder (LName: String, LEinw: Integer, Partei*: String)

* bei Koalitionsregierungen: jeweils eigenes Tupel pro Partei

---

- Finde alle Städtenamen in CDU-regierten Ländern

  $\pi_{SName}(\sigma_{Land=LName}(Städte \times \sigma_{Partei=CDU}(Länder)))$

  oder auch:

  $\pi_{SName}(\sigma_{Land=Lname \wedge Partei=CDU}(Städte \times Länder))$

- Welche Länder werden von der SPD *allein* regiert

  $\pi_{LName}(\sigma_{Partei=SPD}(Länder)) - \pi_{LName}(\sigma_{Partei\neq SPD}(Länder))$

Beispiel Bundesländer

Länder:

LName	LEinw	Partei
Baden-Württemberg	10.745.000	Grüne
Baden-Württemberg	10.745.000	SPD
Bayern	12.510.000	CSU
Bayern	12.510.000	FDP
Berlin	3.443.000	SPD
Berlin	3.443.000	Linke
Brandenburg	2.512.000	SPD
Brandenburg	2.512.000	Linke
Bremen	662.000	SPD
Bremen	662.000	Grüne
Hamburg	1.774.000	SPD
…	…	…

Beispiel Bundesländer

$S = \sigma_{Partei = SPD}(Länder)$:

LName	LEinw	Partei
Baden-W.	10.745.000	SPD
Berlin	3.443.000	SPD
Brandenburg	2.512.000	SPD
Bremen	662.000	SPD
Hamburg	1.774.000	SPD
…	…	…

$T = \sigma_{Partei \neq SPD}(Länder)$:

LName	LEinw	Partei
Baden-W.	10.745.000	Grüne
Bayern	12.510.000	CSU
Bayern	12.510.000	FDP
Berlin	3.443.000	Linke
Brandenburg	2.512.000	Linke
Bremen	662.000	Grüne
…	…	…

- S – T würde nicht das gewünschte Ergebnis liefern, da bei der Mengendifferenz nur exakt gleiche Tupel berücksichtigt werden
- Attribut *Partei* stört; Ergebnis wäre daher wieder Zwischenergebnis S
- $\pi_{LName}(S) - \pi_{LName}(T)$: (gewünschtes Ergebnis)

LName
Hamburg

Abgeleitete Operationen

- Eine Reihe nützlicher Operationen lassen sich mit Hilfe der 5 Grundoperationen ausdrücken:
  - Durchschnitt $R = S \cap T$
  - Quotient $R = S \div T$
  - Join $R = S \bowtie T$

Durchschnitt

- Idee: Finde gemeinsame Elemente in zwei Relationen (Schemata müssen übereinstimmen):

$$R' = S \cap T = \{t \mid t \in S \wedge t \in T\}$$

- Beispiel:
  Welche Personen sind gleichzeitig Mitarbeiter und Student?

Mitarbeiter:	**Name**	**Vorname**
	Huber	Egon
	Maier	Wolfgang
	Schmidt	Helmut

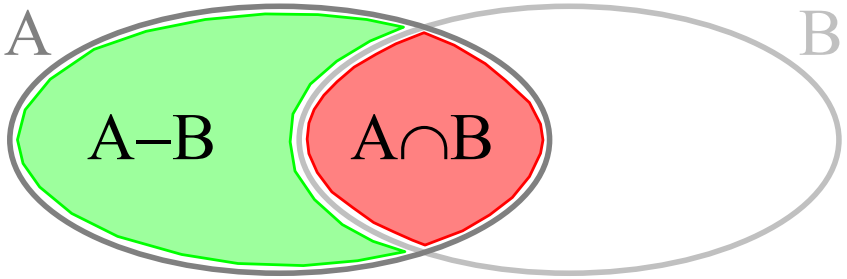
Studenten:	**Name**	**Vorname**
	Müller	Heinz
	Schmidt	Helmut

Mitarbeiter ∩ Studenten:

Name	**Vorname**
Schmidt	Helmut

Durchschnitt

- Implementierung der Operation „Durchschnitt“ mit Hilfe der Grundoperation „Differenz“:



- $A \cap B = A - (A - B)$
- Achtung! Manche Lehrbücher definieren:
  - Durchschnitt ist Grundoperation
  - Differenz ist abgeleitete Operation

  (Definition gleichwertig, also genauso möglich)

Quotient

- Dient zur Simulation eines Allquantors
- Beispiel:

R_1

Programmierer	Sprache
Müller	Java
Müller	Basic
Müller	C++
Huber	C++
Huber	Java

R_2

Sprache
Basic
C++
Java

- Welche Programmierer programmieren in **allen** Sprachen?

$R_1 \div R_2$

Programmierer
Müller

- Umkehrung des kartesischen Produktes (daher: *Quotient*)

Join

- Wie vorher erwähnt:
  Selektion über Kreuzprodukt zweier Relationen
  - Theta-Join (Θ): $R \bowtie_{A \Theta B} S$

    Allgemeiner Vergleich:

    A ist ein Attribut von R und B ein Attribut von S
    Θ ist einer der Operatoren:
    $=, <, \leq, >, \geq, \neq$
  - Equi-Join: $R \bowtie_{A = B} S$
  - Natural Join: $R \bowtie S$:
    - Ein Equi-Join bezüglich aller gleichbenannten Attribute in R und S.
    - Gleiche Spalten werden gestrichen (Projektion)

Join

- Implementierung mit Hilfe der Grundoperationen

$$R \bowtie_{A \Theta B} S = \sigma_{A \Theta B}(R \times S)$$

Gegeben sei folgendes Relationenschema:

Städte (SName: String, SEinw: Integer, Land: String)

Länder (LName: String, LEinw: Integer, Partei*: String)

* bei Koalitionsregierungen: jeweils eigenes Tupel pro Partei

---

- Finde alle Städtenamen in CDU-regierten Ländern

$$\pi_{SName}(\text{Städte} \bowtie_{Land=LName} \sigma_{Partei=CDU}(\text{Länder}))$$

- Bestimme die Namen aller Städte, deren Einwohnerzahl die eines beliebigen Landes übersteigt:

$$\pi_{SName}(\text{Städte} \bowtie_{SEinw>LEinw} \text{Länder}))$$

SQL

- Die wichtigste Datenbank-Anfragesprache SQL beruht wesentlich auf der relationalen Algebra
- Grundform einer Anfrage*:

Projektion → SELECT ⟨Liste von Attributsnamen bzw. *⟩
Kreuzprodukt → FROM ⟨ein oder mehrere Relationennamen⟩
Selektion → [WHERE ⟨Bedingung⟩]

- Mengenoperationen:

  SELECT ... FROM ... WHERE
  UNION
  SELECT ... FROM ... WHERE

---

* SQL ist case-insensitive: SELECT = select = SeLeCt

SQL

- Hauptunterschied zwischen SQL und rel. Algebra:
  - Operatoren bei SQL nicht beliebig schachtelbar
  - Jeder Operator hat seinen festen Platz
- Trotzdem:
  - Man kann zeigen, daß jeder Ausdruck der relationalen Algebra gleichwertig in SQL formuliert werden kann
  - Die feste Anordnung der Operatoren ist also keine wirkliche Einschränkung (Übersichtlichkeit)
  - Man sagt, SQL ist *relational vollständig*
- Weitere Unterschiede:
  - Nicht immer werden Duplikate eliminiert (Projektion)
  - zus. Auswertungsmöglichkeiten (Aggregate, Sortieren)

SELECT

- Entspricht Projektion in der relationalen Algebra
- Aber: Duplikatelimination nur, wenn durch das Schlüsselwort DISTINCT explizit verlangt
- Syntax:

```
SELECT * FROM ...                          -- Keine Projektion
SELECT A1, A2, ... FROM ...                -- Projektion ohne
                                           -- Duplikatelimination
SELECT DISTINCT A1, A2, ...                -- Projektion mit
                                           -- Duplikatelimination
```

- Bei der zweiten Form kann die Ergebnis„*relation*“ also u.U. Duplikate enthalten
- Grund: Performanz

SELECT

- Bei den Attributen A_1, A_2, ... lässt sich angeben...
  - Ein Attributname einer beliebigen Relation, die in der FROM-Klausel angegeben ist
  - Ein skalarer Ausdruck, der Attribute und Konstanten mittels arithmetischer Operationen verknüpft
  - Im Extremfall: Nur eine Konstante
  - Aggregationsfunktionen (siehe später)
  - Ein Ausdruck der Form A_1 AS A_2:
    A_2 wird der neue Attributname (Spaltenüberschrift)
- Beispiel:

```
select   pname,
         preis*13.7603 as oespr,
         preis*kurs as usdpr,
         'US$' as currency
from     produkt, waehrungen....
```

pname	oespr	usdpr	currency
nagel	6.88	0.45	US$
dübel	1.37	0.09	US$
...			

FROM

- Enthält mindestens einen Eintrag der Form R_1
- Enthält die FROM-Klausel mehrere Einträge
  - FROM R_1, R_2, ...

  so wird das kartesische Produkt gebildet:
  - $R_1 \times R_2 \times ...$
- Enthalten zwei verschiedene Relationen R_1, R_2 ein Attribut mit gleichem Namen, dann ist dies in der SELECT- und WHERE-Klausel mehrdeutig
- Eindeutigkeit durch vorangestellten Relationennamen:

```
SELECT   Mitarbeiter.Name, Abteilung.Name, ...
FROM     Mitarbeiter, Abteilung
WHERE    ...
```

FROM

- Man kann Schreibarbeit sparen, indem man den Relationen lokal (innerhalb der Anfrage) kurze Namen zuweist (Alias-Namen):

```
SELECT    m.Name, a.Name, ...
FROM      Mitarbeiter m, Abteilung a
WHERE     ...
```

- Dies lässt sich in der SELECT-Klausel auch mit der Sternchen-Notation kombinieren:

```
SELECT    m.*, a.Name AS Abteilungsname, ...
FROM      Mitarbeiter m, Abteilung a
WHERE     ...
```

- Manchmal Self-Join einer Relation mit sich selbst:

```
SELECT    m1.Name, m2.Name, ...
FROM      Mitarbeiter m1, Mitarbeiter m2
WHERE     ...
```

WHERE

- Entspricht der Selektion der relationalen Algebra
- Enthält genau ein logisches Prädikat Φ (Funktion die einen booleschen Wert (wahr/falsch) zurück gibt).
- Das logische Prädikat besteht aus
  - Vergleichen zwischen Attributwerten und Konstanten
  - Vergleichen zwischen verschiedenen Attributen
  - Vergleichsoperatoren*: = , < , <= , > , >= , <>
  - Test auf *Wert undefiniert*: A_1 IS NULL/IS NOT NULL
  - Inexakter Stringvergleich: A_1 LIKE ‘Datenbank%‘
  - A_1 IN (2, 3, 5, 7, 11, 13)

*Der Gleichheitsoperator wird **nicht** etwa wie in Java verdoppelt

WHERE

- Innerhalb eines Prädikates: Skalare Ausdrücke:
  - Numerische Werte/Attribute mit +, –, *, / verknüpfbar
  - Strings: char_length, Konkatenation || und substring
  - Spezielle Operatoren für Datum und Zeit
  - Übliche Klammernsetzung.
- Einzelne Prädikate können mit AND, OR, NOT zu komplexeren zusammengefasst werden
- Idee der Anfrageauswertung:
  Alle Tupel des kartesischen Produktes aus der FROM-Klausel werden getestet, ob sie Φ erfüllen
- Effizientere Ausführung ist meist möglich
- Das DBMS optimiert die Anfragen automatisch sehr gut

WHERE

- Inexakte Stringsuche: A_1 LIKE ‘Datenbank%‘
  - bedeutet: Alle Datensätze, bei denen Attribut A_1 mit dem Präfix *Datenbank* beginnt.
  - Entsprechend: A_1 LIKE ‘%Daten%‘
  - In dem Spezialstring hinter LIKE ...
    - % steht für einen beliebig belegbaren Teilstring
    - _ steht für genau ein einzelnes frei belegbares Zeichen
- Beispiel:

  Alle Mitarbeiter, deren Nachname auf ‘er‘ endet:

```
select * from mitarbeiter
where name like ‘%er ‘
```

Mitarbeiter

PNr	Name	Vorname	ANr
001	Huber	Erwin	01
002	Mayer	Josef	01
003	Müller	Anton	02
004	Schmidt	Helmut	01

Join

- Normalerweise wird der Join wie bei der relationalen Algebra als Selektionsbedingung über dem kartesischen Produkt formuliert.
- Beispiel: Join zwischen Mitarbeiter und Abteilung
  select * from Mitarbeiter m, Abteilungen a **where** m.ANr = a.ANr
- In neueren SQL-Dialekten auch möglich:
  - **select * from** Mitarbeiter m **join** Abteilungen a **on** a.ANr=m.ANr
  - **select * from** Mitarbeiter **join** Abteilungen **using** (ANr)
  - **select * from** Mitarbeiter **natural join** Abteilungen

  Nach diesem Konstrukt können mit einer WHERE-Klausel weitere Bedingungen an das Ergebnis gestellt werden.

Beispiel (Wdh. S. 12)

select * **from** Mitarbeiter m, Abteilungen a...

PNr	Name	Vorname	m.ANr	a.ANr	Abteilungsname
001	Huber	Erwin	01	01	Buchhaltung
001	Huber	Erwin	01	02	Produktion
002	Mayer	Hugo	01	01	Buchhaltung
002	Mayer	Hugo	01	02	Produktion
003	Müller	Anton	02	01	Buchhaltung
003	Müller	Anton	02	02	Produktion

...**where** m.ANr = a.ANr

PNr	Name	Vorname	m.ANr	a.ANr	Abteilungsname
001	Huber	Erwin	01	01	Buchhaltung
002	Mayer	Hugo	01	01	Buchhaltung
003	Müller	Anton	02	02	Produktion

Beispiele:

- Gegeben sei folgendes Datenbankschema:
  - Kunde (KName, KAdr, Kto)
  - Auftrag (KName, Ware, Menge)
  - Lieferant (LName, LAdr, Ware, Preis)
- Welche Lieferanten liefern Mehl oder Milch?

```
select distinct LName
from Lieferant
where Ware = 'Mehl' or Ware = 'Milch'
```

- Welche Lieferanten liefern irgendetwas, das der Kunde Huber bestellt hat?

```
select distinct LName
from Lieferant l, Auftrag a
where l.Ware = a.Ware and KName = 'Huber'
```

Beispiele (Self-Join):

Kunde (KName, KAdr, Kto)
Auftrag (KName, Ware, Menge)
Lieferant (LName, LAdr, Ware, Preis)

- Name und Adressen aller Kunden, deren Kontostand kleiner als der von Huber ist

```
select k1.KName, k1.KAdr
from Kunde k1, Kunde k2
where k1.Kto < k2.Kto and k2.KName = 'Huber'
```

- Finde alle Paare von Lieferanten, die eine gleiche Ware liefern

```
select distinct L1.Lname, L2.LName
from Lieferant L1, Lieferant L2
where L1.Ware=L2.Ware and L1.LName<L2.LName
```

?

Beispiele (Self-Join)

Lieferant*

Müller	Mehl
Müller	Haferfl
Bäcker	Mehl

select * from Lieferant L1, Lieferant L2 **where** L1.Ware=L2.Ware:

Müller	Mehl	Müller	Mehl
Müller	Mehl	Bäcker	Mehl
Müller	Haferfl	Müller	Haferfl
Bäcker	Mehl	Müller	Mehl
Bäcker	Mehl	Bäcker	Mehl

Nach Projektion:

L1.LName > L2.LName

~~Müller~~	~~Müller~~
~~Müller~~	~~Bäcker~~
Bäcker	Müller
~~Bäcker~~	~~Bäcker~~

L1.LName = L2.LName

Outer Join

- Problem:
  Beim gewöhnlichen („inner“) Join gehen diejenigen Tupel verloren, die keine Join-Partner in der jeweiligen anderen Relation haben.
- Beispiel:
  Auflistung aller Kunden mit ihren aktuellen Bestellungen:
  select * from kunde k, auftrag a **where** k.kname = a.kname
  Kunden ohne aktuellen Auftrag erscheinen nicht.

Kunde:

KName	KAdr	Kto
Maier	M	10
Huber	M	25
Geizhals	RO	0

Auftrag:

KName	Ware	...
Maier	Brot	
Maier	Milch	
Huber	Mehl	

Kunde ⋈ Auftrag:

KName	KAdr	Kto	Ware	...
Maier	M	10	Brot	
Maier	M	10	Milch	
Huber	M	25	Mehl	

Geizhals erscheint nicht mehr in der erweiterten Liste.

Outer Join

- Ein Outer Join ergänzt das Join-Ergebnis um die Tupel, die keinen Joinpartner in der anderen Relation haben.
- Das Ergebnis wird mit NULL-Werten aufgefüllt:
  select * from kunde natural outer join auftrag

Kunde:

KName	KAdr	Kto
Maier	M	10
Huber	M	25
Geizhals	RO	0

Auftrag:

KName	Ware	...
Maier	Brot	
Maier	Milch	
Huber	Mehl	

Kunde nat. outer join Auftrag:

KName	KAdr	Kto	Ware	...
Maier	M	10	Brot	
Maier	M	10	Milch	
Huber	M	25	Mehl	
Geizhals	RO	0	NULL	

Outer Join

- Aufstellung aller Möglichkeiten:
  - **[inner] join**: keine verlustfreie Relation
  - **left [outer] join**: die linke Relation ist verlustfrei
  - **right [outer] join**: die rechte Relation ist verlustfrei
  - **full [outer] join**: beide Relationen verlustfrei
- Kombinierbar mit Schlüsselworten **natural**, **on**, **using:**
  select * from L **[left/right/full [outer]] natural join** R

L

A	B
1	2
2	3

R

B	C
3	4
4	5

inner

A	B	C
2	3	4

left

A	B	C
1	2	⊥
2	3	4

right

A	B	C
2	3	4
⊥	4	5

full

A	B	C
1	2	⊥
2	3	4
⊥	4	5

UNION, INTERSECT, EXCEPT

- Üblicherweise werden mit diesen Operationen die Ergebnisse zweier **select-from-where**-Blöcke verknüpft:

  select * **from** Mitarbeiter **where** name **like** ‘A%‘
  union
  select * **from** Studenten **where** name **like** ‘A%‘

- Bei neueren Informationssystemen ist auch möglich:

  select * **from** Mitarbeiter **union** Studenten **where** ...

- Es gibt folgende Mengen-Operationen:
  - **union**: Vereinigung **mit** Duplikatelimination
  - **union all**: Vereinigung **ohne** Duplikatelimination
  - **intersect**: Schnittmenge
  - **except, minus**: Mengen-Differenz
- Achtung: Zwei Tupel sind nur dann gleich, wenn alle ihre Attributwerte gleich sind

UNION, INTERSECT, EXCEPT

- Die relationale Algebra verlangt, dass die beiden Relationen, die verknüpft werden, das gleiche Schema besitzen (Namen und Wertebereiche)
- SQL verlangt nur kompatible Wertebereiche, d.h.:
  - beide Wertebereiche sind **character** (Länge usw. egal)
  - beide Wertebereiche sind Zahlen (Genauigkeit egal)
  - oder beide Wertebereiche sind gleich
- Die Namen der Attribute müssen nicht gleich sein
- Befinden sich Attribute gleichen Namens auf unterschiedlichen Positionen, sind trotzdem nur die Positionen maßgeblich (das ist oft irritierend)

UNION, INTERSECT, EXCEPT

- Mit dem Schlüsselwort **corresponding** beschränken sich die Operationen automatisch auf die gleich benannten Attribute (Projektion)
- Beispiel:

R:

A	B	C
1	2	3
2	3	4

S:

A	C	D
2	2	3
5	3	2

select * from *R*
union
select * from *S*:

A	B	C
1	2	3
2	3	4
2	2	3
5	3	2

select * from *R*
union corresponding
select * from *S*:

A	C
1	3
2	4
2	2
5	3

UNION, INTERSECT, EXCEPT

- Bei **corresponding** wird *vor* der Vereinigung automatisch eine Projektion bezüglich aller gleich benannten Attribute durchgeführt
- Dies kann man besser durch explizites Aufzählen der Attribute in den **select**-Klauseln erreichen.
- So ist auch möglich:

select A,B,C from *R*
union
select A,D,C from *S*:

A	B	C
1	2	3
2	3	4
2	3	2
5	2	3

Oder auch:

select A,B,C,**null as** D from *R*
union
select A,**null**,C,D from *S*:

A	B	C	D
1	2	3	**null**
2	3	4	**null**
2	**null**	2	3
5	**null**	3	2

UNION, INTERSECT, EXCEPT

- Bei **corresponding** ist ggf. die Reihenfolge der Attribute der erstgenannten Teilanfrage maßgeblich:

T:

X	Y
1	2

U:

Y	X
3	4

select * from *T*
union
select * from *U*:

X	Y
1	2
3	4

select * from *T*
union corresponding
select * from *U*:

X	Y
1	2
4	3

- Die Attribute der zweiten Teil-Anfrage ändern evtl…
  - ohne **corresponding**: sie ändern ihren Namen
  - mit **corresponding**: sie ändern ihre Reihenfolge

Änderungs-Operationen

- Bisher: Nur *Anfragen* an das Datenbanksystem
- Änderungsoperationen modifizieren den Inhalt eines oder mehrerer Tupel einer Relation
- Grundsätzlich unterscheiden wir:
  - INSERT: Einfügen von Tupeln in eine Relation
  - DELETE: Löschen von Tupeln aus einer Relation
  - UPDATE: Ändern von Tupeln einer Relation
- Diese Operationen sind verfügbar als...
  - Ein-Tupel-Operationen
    z.B. die Erfassung eines neuen Mitarbeiters
  - Mehr-Tupel-Operationen
    z.B. die Erhöhung aller Gehälter um 2.1%

Die UPDATE-Anweisung

- Syntax:

  update *relation*
  set $attribut_1$ = $ausdruck_1$
  [, … ,
  $attribut_n$ = $ausdruck_n$]
  [**where** *bedingung*]

- Wirkung:
  In allen Tupeln der Relation, die die Bedingung erfüllen (falls angegeben, sonst in allen Tupeln), werden die Attributwerte wie angegeben gesetzt.

Die UPDATE-Anweisung

- UPDATE ist i.a. eine Mehrtupel-Operation
- Beispiel:
  update Angestellte
  set Gehalt = 6000

- Wie kann man sich auf ein einzelnes Tupel beschränken?
  Spezifikation des Schlüssels in WHERE-Bedg.

- Beispiel:
  update Angestellte
  set Gehalt = 6000
  where PNr = 7

Die UPDATE-Anweisung

- Der alte Attribut-Wert kann bei der Berechnung des neuen Attributwertes herangezogen werden
- Beispiel:
  Erhöhe das Gehalt aller Angestellten, die weniger als 3000,-- € verdienen, um 2%

```
update  Angestellte
set     Gehalt = Gehalt * 1.02
where   Gehalt < 3000
```

- UPDATE-Operationen können zur Verletzung von Integritätsbedingungen führen:
  Abbruch der Operation mit Fehlermeldung.

Die DELETE-Anweisung

- Syntax:
  delete from *relation*
  [**where** *bedingung*]
- Wirkung:
  - Löscht alle Tupel, die die Bedingung erfüllen
  - Ist keine Bedingung angegeben, werden *alle* Tupel gelöscht
  - Abbruch der Operation, falls eine Integritätsbedingung verletzt würde (z.B. Fremdschlüssel ohne *cascade*)
- Beispiel: Löschen aller Angestellten mit Gehalt 0

  delete from Angestellte
  where Gehalt = 0

Die INSERT-Anweisung

- Zwei unterschiedliche Formen:
  - Einfügen konstanter Tupel (Ein-Tupel-Operation)
  - Einfügen berechneter Tupel (Mehr-Tupel-Operation)

- Syntax zum Einfügen konstanter Tupel:
  insert into *relation* (*attribut*$_1$, *attribut*$_2$,...)
  values (*konstante*$_1$, *konstante*$_2$, ...)

- oder:
  insert into *relation*
  values (*konstante*$_1$, *konstante*$_2$, ...)

Einfügen konstanter Tupel

- Wirkung:
  Ist die optionale Attributliste hinter dem Relationennamen angegeben, dann...
  - können unvollständige Tupel eingefügt werden:
    Nicht aufgeführte Attribute werden mit NULL belegt
  - werden die Werte durch die Reihenfolge in der Attributsliste zugeordnet
- Beispiel:
  insert into Angestellte (Vorame, Name, PNr)
  values ('Donald', 'Duck', 678)

PNr	Name	Vorname	ANr
678	Duck	Donald	NULL

Einfügen konstanter Tupel

- Wirkung:
  Ist die Attributliste *nicht* angegeben, dann...
  - können unvollständige Tupel nur durch explizite Angabe von NULL eingegeben werden
  - werden die Werte durch die Reihenfolge in der DDL-Definition der Relation zugeordnet (mangelnde Datenunabhängigkeit!)
- Beispiel:
  insert into Angestellte
  values (678, 'Duck', 'Donald', NULL)

PNr	Name	Vorname	ANr
678	Duck	Donald	NULL

Einfügen berechneter Tupel

- Syntax zum Einfügen berechneter Tupel:
  insert into *relation* [**(***attribut*$_1$ **, ...)**]
  (select ... **from** ... **where** ... **)**
- Wirkung:
  - Alle Tupel des Ergebnisses der SELECT-Anweisung werden in die Relation eingefügt
  - Die optionale Attributliste hat dieselbe Bedeutung wie bei der entsprechenden Ein-Tupel-Operation
  - Bei Verletzung von Integritätsbedingungen (z.B. Fremdschlüssel nicht vorhanden) wird die Operation nicht ausgeführt (Fehlermeldung)
- Dies ist eigentlich eine Form einer Unteranfrage (Subquery: siehe auch Kapitel 4)

Einfügen berechneter Tupel

- Beispiel:
  Füge alle Lieferanten in die Kunden-Relation ein
  (mit Kontostand 0)

- Datenbankschema:
  - Kunde (KName, KAdr, Kto)
  - Lieferant (LName, LAdr, Ware, Preis)

insert into Kunde
(select distinct LName, LAdr, 0 **from** Lieferant**)**