Informationssyteme[1]

Kapitel 2: Die Relationale Algebra

Wintersemester 2020/21

Prof. Dr. Peer Kröger
Institut für Informatik
Arbeitsgruppe Informationssysteme und Data Mining

[1] Die folgenden Folien basieren in großen Teilen auf Material zur Vorlesung "Datenbanksysteme", die ich in meiner Zeit an der LMU München mehrmals gehalten habe. Das Material ist von den damaligen Kollegen maßgeblich (mit-)gestaltet worden, insbes. von Prof. Dr. C. Böhm.

Übersicht

1. Einleitung
2. Grundoperationen
3. Abgeleitete Operationen
4. Implementierung in SQL
5. Änderungsoperationen in SQL

Agenda

1. Einleitung
2. Grundoperationen
3. Abgeleitete Operationen
4. Implementierung in SQL
5. Änderungsoperationen in SQL

Mit Relationen arbeiten

- Es gibt viele formale Konzepte, um mit Relationen zu "arbeiten", d.h. insbesondere: Anfragen zu formulieren
- Die wichtigsten Beispiele sind:
  - Relationale Algebra: verfolgt eine prozeduralen Ansatz (wie soll das Ergebnis einer Anfrage berechnet werden)
  - Relationen-Kalkül: verfolgt einen deskriptiven Ansatz (wie soll das Ergebnis einer Anfrage "aussehen", welche Eigenschaften hat das Ergebnis)
- Sie dienen als theoretisches Fundament für konkrete Anfrage-Sprachen wie
  - SQL: basiert i.w. auf der relationalen Algebra
  - QBE (= Query By Example) und Quel: basieren auf dem Relationen-Kalkül

Algebra: Begriff

- Das Konzept der Algebra kennen wir aus der Mathematik
- Eine Algebra besteht i.w. aus zwei "Zutaten":
  - eine Menge M von Elementen (die in diesem Zusammenhang auch Operanden genannt werden)
  - eine Menge O von Operationen (Funktionen) über M
- Wichtigste Eigenschaft einer Algebra für uns ist die Abgeschlossenheit der Operationen in O:
  Wird ein Operator aus O auf Elemente aus M angewandt, ist das Ergebnis wieder ein Element der Menge M
  Oder anders ausgedrückt: in den Signaturen der Operationen in O kommen als Eingabe- und Ausgabe-Typen nur die Menge M vor
- Warum ist das wichtig: weil es die Verschachtlung von mehreren Operationen erlaubt

Algebra: Begriff

Beispiele: Algebren

- Natürliche Zahlen mit Addition, Multiplikation
- Zeichenketten mit Konkatenation
- Boolsche Algebra: Wahrheitswerte mit $\wedge$, $\vee$, $\neg$
- Mengen-Algebra: Menge der Mengen mit $\cup$, $\cap$, $\setminus$

Relationale Algebra

- Rechnen mit Relationen
- Operanden: Relationen (Tabellen)
- Beispiele für Operationen:
  - Selektion von Tupeln nach bestimmten Kriterien (z.B. Gehalt > 1000)
  - Kombination mehrerer Tabellen
- Abgeschlossenheit: Ergebnis einer Operation ist immer eine (neue) Relation (oft ohne eigenen Namen)
- Damit können einfache Terme der relationalen Algebra zu komplexeren zusammengesetzt werden: Anfragen bestehen sehr häufig aus mehreren, ineinander verschachtelten Operationen

Agenda

1. Einleitung
2. Grundoperationen
3. Abgeleitete Operationen
4. Implementierung in SQL
5. Änderungsoperationen in SQL

Übersicht

Grundoperationen der Relationalen Algebra

- Vereinigung zweier Relationen S, T: $R = S \cup T$
- Differenz zweier Relationen S, T: $R = S - T$
- Kart. Produkt zweier Relationen S, T: $R = S \times T$
- Selektion von Tupeln aus einer Relation S: $R = \sigma_F(S)$
- Projektion auf Attributmenge A einer Relation S: $R = \pi_A(S)$

Bemerkungen:

- Mit den Grundoperationen lassen sich weitere Operationen, (z.B. die Schnittmenge) nachbilden
- In manchen Büchern wird die Umbenennung von Attributen als 6. Grundoperation bezeichnet
- T, S heißen auch Basis-Relationen

Vereinigung

Vereinigung: $S \cup T$

Die Vereinigung von S und T beinhaltet alle Tupel aus S und alle Tupel aus T, formal:

$$R = S \cup T = \{t \mid t \in S \vee t \in T\}$$

- Achtung: Vereinigung von S und T nur anwendbar, wenn die Schemata von S und T übereinstimmen (Namen und Domänen aller Attribute): $schema(S) = schema(T)$
- Die Ergebnis-Relation R bekommt dieses Schema: $schema(R) = schema(S) = schema(T)$
- Kardinalität: $|R| = |S \cup T| \leq |S| + |T|$
- Duplikate in R? Was genau sind Duplikate und was passiert damit?

Vereinigung

Beispiel: Vereinigung

Basis-Relationen:

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

Differenz

Differenz: $S \setminus T$

Die Differenz von S und T beinhaltet alle Tupel aus S, die nicht in T sind, formal:

$$R = S \setminus T = \{t \mid t \in S \land t \notin T\}$$

- Achtung: Differenz von S und T nur anwendbar, wenn $schema(S) = schema(T)$
- Es gilt wiederum: $schema(R) = schema(S) = schema(T)$
- Kardinalität: $|R| = |S \setminus T| \geq |S| - |T|$
- Achtung 2: Zwei Tupel gelten nur dann als gleich, wenn alle Attribut-Werte gleich sind

Differenz

Beispiel: Differenz

Basis-Relationen:

Mitarbeiter:

Name	Vorname
Huber	Egon
Maier	Wolfgang
Schmidt	Helmut

Studenten:

Name	Vorname
Müller	Heinz
Schmidt	Helmut

Alle Mitarbeiter ohne diejenigen, die auch Studenten sind:

Mitarbeiter – Studenten:

Name	Vorname
Huber	Egon
Maier	Wolfgang

Kartesisches Produkt

Kartesisches Produkt: $S \times T$

Das kartesische Produkt von S und T beinhaltet alle Kombinationen von Tupeln aus S und T sind, formal:

Sei $schema(S) = (s_1, ..., s_n)$ und $schema(T) = t_1, ..., t_m)$.

$$R \;\; = S \times T = \;\; \{(v_{s_1}, ..., v_{s_n}, v_{t_1}, ..., v_{t_m}) \\ \mid ((v_{s_1}, ..., v_{s_n}) \in S \wedge (v_{t_1}, ..., v_{t_m}) \in T\}$$

- Anwendung nicht eingeschränkt bzgl. Schemas
- Es gilt: $schema(R) = schema(S) \circ schema(T)$ (d.h. Schemas werden konkateniert)
- Kardinalität: $|R| = |S \times T| \geq |S| \cdot |T|$

Kartesisches Produkt

Beispiel: Differenz

Verknüpfung von Mitarbeiter und Abteilungen:

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

Gibt uns das alle Mitarbeiter mit ihren jeweiligen Abteilungs-Infos?

Selektion

Selektion: $\sigma_F(S)$

Die Selektion von S mit Formel ("Selektions-Bedingung") F beinhaltet alle Tupel t aus S, die die Bedingung F erfüllen ($F(t) = \textbf{true}$)[2], formal:

$$R = \sigma_F(S) = \{t \mid t \in S \land F(t) = \textbf{true}\}$$

- Es gilt: $\mathit{schema}(R) = \mathit{schema}(S)$
- Die Formel F besteht aus
  - Konstanten (z.B. "Huber", 42, ...)
  - Attributen (als Name oder Nummer)
  - Vergleichsoperationen $(=, <, \leq, >, \geq, \neq)$
  - Boolsche Operatoren $(\land, \lor, \neg)$
- Kardinalität: $|R| = |\sigma_F(S)| \leq |S|$

[2] Eine formale Definition von "Auswertung einer Formel" sehen wir im Kapitel über Tupelkalkül.

Selektion

Beispiel: Selektion

Mitarbeiter

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

Alle Mitarbeiter von Abteilung 01:

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

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

Bekommen wir jetzt auch alle Mitarbeiter mit den Abteilungs-Infos?

Selektion

Beispiel: Kreuzprodukt mit anschl. Selektion

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

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

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

Projektion: $\pi_A(S)$

Die Projektion von S auf eine Menge von Attributen $A \subseteq schema(S)$ löscht für alle Tupel aus S die Attribute $schema(S) \setminus A$, formal:

Sei $A = \{a_1, ..., a_n\} \subseteq schema(S)$.

$$R = \pi_A(S) = \{(v_{a_1}, ...v_{a_n} \mid (v_{a_1}, ...v_{a_n}, ...) \in S\}$$

- Die Projektion erlaubt es also, Spalten einer Relation auszuwählen, bzw. nicht ausgewählte Spalten zu streichen und (im geordenten RS) die Reihenfolge der Spalten zu verändern
- Es gilt: $schema(R) = A$
- Kardinalität: $|\pi_A(S)| \leq |S|$ (WARUM? und WANN?)

Projektion

Beispiel: Projektion

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

Duplikaten-Elimination

- Erforderlich nach vermeintlich “billigen” Basisoperationen:
  - Projektion
  - Vereinigung
- Wie funktioniert Duplikat-Elimination?

  for(int $i = 0$; $i < |R|$; $i++$)
      for(int $j = 0$; $j < i$; $j++$)
          if($R[i] == R[j]$) lösche $R[j]$;

- Aufwand? $n = |R|$: $O(n^2)$
- Besserer Algorithmus mit Sortieren: $O(n \log n)$
- Folge: eigentlich billige Grundoperationen werden durch Duplikat-Elimination teuer!

Beispiel-Anfragen mit Grundoperationen

Running Example

Gegeben sei folgendes Relationen-Schema:

Städte (*SName*: String, *SEinw*: Int, *Land*: String)

Länder (*LName*: String, *LEinw*: Int, *Partei*: String

(Bei Koalitionsregierungen: jeweils eigenes Tupel fro Regierungspartei in "Länder")

- Bestimme alle Großstädte mit mehr als 500.000 Einwohner und ihre Einwohnerzahl:

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

Beispiel-Anfragen mit Grundoperationen

Running Example

Gegeben sei folgendes Relationen-Schema:

Städte (*SName*: String, *SEinw*: Int, *Land*: String)

Länder (*LName*: String, *LEinw*: Int, *Partei*: String

(Bei Koalitionsregierungen: jeweils eigenes Tupel fro Regierungspartei in "Länder")

- In welchem Land liegt die Stadt "Buxtehude":

$$\pi_{Land}(\sigma_{Name=\text{"Buxtehude"}}(\text{Städte}))$$

Beispiel-Anfragen mit Grundoperationen

Running Example

Gegeben sei folgendes Relationen-Schema:

Städte (*SName*: String, *SEinw*: Int, *Land*: String)

Länder (*LName*: String, *LEinw*: Int, *Partei*: String

(Bei Koalitionsregierungen: jeweils eigenes Tupel fro Regierungspartei in "Länder")

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

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

Beispiel-Anfragen mit Grundoperationen

Running Example

Gegeben sei folgendes Relationen-Schema:

Städte (*SName*: String, *SEinw*: Int, *Land*: String)

Länder (*LName*: String, *LEinw*: Int, *Partei*: String

(Bei Koalitionsregierungen: jeweils eigenes Tupel fro Regierungspartei in "Länder")

- Finde alle Städtenamen in Ländern mit der Partei "XYZ" in der Regierung:

$$\pi_{SName}(\sigma_{Land=LName}(\text{Städte} \times \sigma_{Partei=\text{``XYZ''}}(\text{Länder})))$$

alternativ:

$$\pi_{SName}(\sigma_{Land=LName \wedge Partei=\text{``XYZ''}}(\text{Städte} \times \text{Länder}))$$

Beispiel-Anfragen mit Grundoperationen

Running Example

Gegeben sei folgendes Relationen-Schema:

Städte (*SName*: String, *SEinw*: Int, *Land*: String)

Länder (*LName*: String, *LEinw*: Int, *Partei*: String

(Bei Koalitionsregierungen: jeweils eigenes Tupel fro Regierungspartei in "Länder")

- Welche Ländern werden von der Partei "XYZ" alleine regiert:

$$\pi_{SName}(\sigma_{Partei=\text{``XYZ''}}(\text{Länder})) \setminus \pi_{SName}(\sigma_{Partei\neq\text{``XYZ''}}(\text{Länder}))$$

Agenda

1. Einleitung
2. Grundoperationen
3. Abgeleitete Operationen
4. Implementierung in SQL
5. Änderungsoperationen in SQL

Übersicht

Eine Reihe nützlicher Operationen lassen sich mit Hilfe der fünf Grundoperationen ausdrücken, z.B.:

- Durchschnitt $R = S \cap T$
- Quotient: $R = S \div T$
- Join: $R = S \bowtie T$

Weitere abgeleitete Operationen sind natürlich möglich (haben aber i.d.R. keinen eigenen Namen).

Durchschnitt

Durchschnitt

Der Durchschnitt zweier Relationen S und T bestimmt die gemeinsame Elemente in S und T, formal:

$$R = S \cap T = \{t \mid t \in S \wedge t \in T\}$$

- Bedingung: $schema(S) = schema(T)$
- Es gilt: $schema(R) = schema(S) = schema(T)$

Durchschnitt

Beispiel: Differenz

Welche Personen sind gleichzeitig Mitarbeiter und Student?

Mitarbeiter:

Name	Vorname
Huber	Egon
Maier	Wolfgang
Schmidt	Helmut

Studenten:

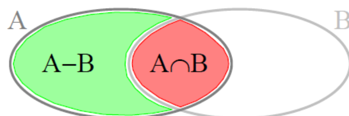
Name	Vorname
Müller	Heinz
Schmidt	Helmut

Mitarbeiter $\cap$ Studenten:

Name	Vorname
Schmidt	Helmut

Durchschnitt

- Implementierung mit Hilfe der Basisoperationen:



$A \cap B = A \setminus (A \setminus B)$

- Bemerkung: Manche Lehrbücher definieren den Durchschnitt als Grundoperation und die Differenz als abgeleitete Operation (nicht verwirren lassen, das ist natürlich genauso möglich!)

Quotient

- Der Quotient dient zur Simulation eines Allquantors ("Alle Tupel in S mit der Eigenschaft T")
- Umkehrung des kartesischen Produktes (daher "Quotient")

Beispiel: Quotient

S

Programmierer	**Sprache**
Müller	Java
Müller	Python
Müller	C#
Huber	Python
Huber	Java

T

Sprache
Python
C#
Java

Welche Programmierer können alle Sprachen?

$S \div T$

Programmierer
Müller

Join

Join

Der Join von S und T bzgl. einer Join-Bedingung F ist eine Selektion über dem Kreuzprodukt von S und ST, formal

$$R = S \bowtie_F T = \{t \mid t \in S \times T \land F(t) = \textbf{true}\}$$

- Als Grundoperationen: $S \bowtie_F T = \sigma_F(S \times T)$

Join-Arten

Je nachdem, welche Gestalt F hat, unterscheiden wir:

- Theta-Join (Θ): $S \bowtie_{A \Theta B} T$:
  - Allgemeiner Vergleich
  - A ist ein Attribut von S und B ist ein Attribut von T
  - Θ ist einer der Operatoren $=, <, \leq, >, \geq, \neq$
- Equi-Join: $S \bowtie_{A=B} T$:
  - Vergleich auf Gleichheit
  - wie oben aber $\Theta =$ "$=$"
- Natürlicher (engl. "natural") Join
  - Equi-Join bzgl. aller gleichbenannten Attribute in S und T

    (Achtung, die Attribute müssen wirklich denselben Namen haben!!!)
  - Gleiche Spalten werden zu einer gestrichen (Projektion)

Join

Beispiele: Join

Gegeben wieder folgendes Relationen-Schema (wie oben):

Städte (*SName*: String, *SEinw*: Int, *Land*: String)

Länder (*LName*: String, *LEinw*: Int, *Partei*: String

- Finde alle Städtenamen der Länder, die von Partei "XYZ" regiert werden:

$$\pi_{Sname}(\text{Städte} \bowtie_{Land=LName} \sigma_{Partei=\text{"XYZ"}}(\text{Länder}))$$

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

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

Agenda

1. Einleitung
2. Grundoperationen
3. Abgeleitete Operationen
4. Implementierung in SQL
5. Änderungsoperationen in SQL

Grundform von SQl-Anfragen

- Wie bereits erwähnt, beruht die DML von SQL wesentlich auf der relationalen Algebra
- SQL-Anfragen haben eine feste Grundform:

`SELECT`	⟨Liste von Attributsnamen oder $\star$	$\longleftarrow$	Projektion
`FROM`	⟨ein oder mehrere Relationennamen⟩	$\longleftarrow$	Kreuzprodukt
`WHERE`	⟨Bedingung⟩	$\longleftarrow$	Selektion

- Mengenoperationen können nur "außerhalb" dieser Grundform verwendet werden:

```
SELECT ... FROM ... WHERE
UNION                        (oder: UNION ALL, INTERSECT, MINUS/EXCEPT)
SELECT ... FROM ... WHERE
```

- Und auch nochmal: SQL ist Case-insensitiv!!!

SQL versus Relationale Algebra

- Hauptunterschied zwischen SQL und relationaler 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 (sondern dient hauptsächlich der Übersichtlichkeit)
  - Man sagt, SQL ist relational vollständig
- Weitere Unterschiede:
  - Nicht immer werden Duplikate eliminiert (Projektion — werden wir gleich sehen ...)
  - zus. Auswertungsmöglichkeiten (Aggregation, Gruppierung, Sortieren — werden wir später sehen)

SELECT Klausel

- Entspricht der Projektion in der relationalen Algebra
- Achtung: Duplikat-Elimination nur, wenn durch das zusätzliche Schlüsselwort DISTINCT explizit verlangt
- Alternativen:

`SELECT * FROM ...`	⟵	Keine Projektion
`SELECT A, B FROM ...`	⟵	Projektion ohne Duplikat-Elimination
`SELECT DISTINCT A, B FROM ...`	⟵	Projektion mit Duplikat-Elimination

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

SELECT Klausel

- Bei der Attribut-Liste lässt sich angeben:
  - Ein Attribut-Name einer beliebigen Relation, die in der `FROM`-Klausel angegeben ist
  - Ein skalarer Ausdruck, der Attribute und Konstanten mittels arithmetischer Operationen verknüpft
  - Im Extremfall sogar nur eine Konstante
  - Aggregationsfunktionen (siehe später)
  - Ein Ausdruck der Form `A AS B`: *A* wird der neue Attribut-Name (Spaltenüberschrift) von *B*

Beispiel: SELECT

Gegeben folgendes Relationen-Schema mit Produkten (Preise in US-Dollar): USProducts (*PName*: String, *Preis*: Float)

```
SELECT PName AS Product,
   Preis * 0,85 AS Price,
   'EUR' AS Currency
FROM USProducts
```

Product	Price	Currency
Nagel	0.88	EUR
Duebel	0.45	EUR
Schraube	1.02	EUR

`FROM` Klausel

- Enthält mindestens einen Relationen-Namen
- Enthält die `FROM`-Klausel mehrere Einträge `FROM S,T,...`, so wird das kartesische Produkt $S \times T \times \ldots$ gebildet
- Enthalten zwei verschiedene Relationen S und T ein Attribut mit gleichem Namen, dann ist dies in den `SELECT`- und `WHERE`-Klauseln mehrdeutig
- Eindeutigkeit dieser Attribute lässt sich auf verschiedene Wege (jeweils mit Punkt-Notation) herstellen:
  - Unter Verwendung von Relationen-Namen:
    ```
    SELECT Mitarbeiter.Name, Abteilung.Name
    FROM Mitarbeiter, Abteilung
    WHERE ...
    ```
  - Unter Verwendung von Alias-Namen:
    ```
    SELECT m.Name, a.Name
    FROM Mitarbeiter m, Abteilung a
    WHERE ...
    ```

FROM Klausel

- Diese Notationen lassen sich auch mit der Sternchen-Notation in der SELECT-Klausel kombinieren, z.B. mit Alias-Namen:

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

- Ebenfalls wichtig ist diese Eindeutigkeit ggfls. bei einem Selbst-Join, also einem Join einer Relation mit sich selbst, z.B.:

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

WHERE Klausel

- Entspricht der Selektion
- Enthält ein logisches Prädikat Φ (= Funktion, die einen boolschen Wert zurück gibt)
- Das Prädikat Φ besteht aus:
  - Vergleichen zwischen Attributwerten und Konstanten
  - Vergleichen zwischen verschiedenen Attributen
  - Als Vergleichsoperatoren gibt es:
    - Arithmetik:=, <, <=, >, >=, <>
    - Test auf undefinierten Wert: IS NULL
    - Test auf definierten Wert: IS NOT NULL
    - Stringvergleich: LIKE
    - Test auf "enthalten in": IN (<Werte-Liste>)
  - Für Strings (VARCHAR, ect.) gilt das Zeichen % als Wildcard

`WHERE` Klausel

- Die Operanden der Vergleiche können wiederum zusammen gesetzt sein (es gilt die übliche Klammersetzung):
  - Numerische Werte/Attribute sind mit mit `+`, `-`, `*`, `/` verknüpfbar
  - Für Strings gibt es Operatoren wie `char_length`, `||` (Konkatenation) und `substring`
  - Spezielle Operatoren für Datum und Zeit
- Einzelne Prädikate können mit `AND`, `OR`, `NOT` zu komplexeren zusammengefasst werden
- Alle Tupel des kartesischen Produktes aus der FROM-Klausel werden getestet, ob sie Φ erfüllen (Das DBMS optimiert diese Auswertung meist automatisch sehr gut)

String-Vergleich

- Inexakte Stringsuche: `A LIKE 'Daten%'` bedeutet: alle Datensätze, bei denen Attribut *A* mit dem Präfix "Bund" beginnt
- Entsprechend: `A LIKE '%t%'`
- Neben der Wildcard % (ein beliebiger Teilstring) kann man auch _ (genau ein beliebiges Zeichen) verwenden

Beispiel: String-Matching

Alle Mitarbeiter, deren Nachname auf "er" endet:

```
SELECT * FROM Mitarbeiter
WHERE Name LIKE '%er'
```

Mitarbeiter

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

Joins in SQL

- 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
```
- Weitere Bedingungen (die nichts mit der Join-Bedingung zu tun haben, das Ergebnis aber nochmal einschränken, können natürlich mit `AND` in die `WHERE`-Klausel geschrieben werden

Joins in SQL

- In neueren SQL-Dialekten ist in manchen Fällen auch eine Trennung der Join-Bedingung aus der WHERE-Klausel möglich:

  ```
  SELECT ...
  ```

  - ```
    FROM Mitarbeiter m JOIN Abteilungen a
             ON m.ANr = a.ANr
    WHERE ...
    ```
  - ```
    FROM Mitarbeiter JOIN Abteilungen
             USING ANr
    WHERE ...
    ```
  - ```
    FROM Mitarbeiter  NATURAL JOIN Abteilungen
    WHERE ...
    ```

- Alle Varianten sind äquivalent zueinander und natürlich auch zur entsprechenden Variante mit Join-Bedingung in WHERE-Klausel)
    ```

Beispiele

- Gegeben folgende Tabelle:

Customer

CName	**CAdr**	**Account**
Muller	D.C.	0
Musg	SFO	200 Mio
Trumpf	D.C.	-400 Mio

  - Gesucht: alle Kunden mit positivem Kontostand

    ```
    SELECT * FROM Customer WHERE Account ≥ 0
    ```

  - Gesucht: Die Namen aller Kunden mit positivem Kontostand aus D.C.:

    ```
    SELECT CName FROM Customer
       WHERE Account ≥ 0 AND CAdr = 'D.C.'
    ```

Beispiele

- Gegeben folgendes Relationen-Schema:
  Kunde(**KName**, KAdr, Kontostand)
  Auftrag(**KName**, **Produkt**, Menge)
  Lieferant(**LName**, LAdr, **Produkt**, Preis)
- Gesucht: welche Lieferanten liefern “Milch” or “Bier”
  ```
  SELECT DISTINCT LName FROM Lieferant
     WHERE Produkt = ‘Milch’ OR Produkt = ‘Bier’
  ```
- Gesucht: welche Lieferanten liefern irgendwas, das “Huber” bestellt hat
  ```
  SELECT DISTINCT LName FROM Lieferant l, Auftrag a
     WHERE l.Produkt = a.Produkt AND KName = ‘Huber’
  ```

Beispiele

- Gegeben folgendes Relationen-Schema:
  Kunde(**KName**, KAdr, Kontostand)
  Auftrag(**KName**, **Produkt**, Menge)
  Lieferant(**LName**, LAdr, **Produkt**, Preis)

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

```
SELECT DISTINCT k1.KName, k1.KAdr
   FROM Kunde k1, Kunde k2
   WHERE k1.Kontostand < k2.Kontostand
         AND
         k2.KName = 'Huber'
```

Outer Joins

- Beim normalen (auch "inneren", engl. "inner") Join gehen die Tupel verloren, die keine Join-Partner bzgl. der Join-Bedingung 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 im Resultat:

Kunde:

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

Auftrag:

KName	Ware	...
Maier	Brot	
Maier	Milch	
Huber	Mehl	

Kunde $\bowtie$ Auftrag:

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

Outer Joins

- Ein äußerer (engl. "outer") Join ergänzt das Join-Ergebnis um die Tupel, die keinen Join-Partner in der anderen Relation haben
- Das Ergebnis wird mit NULL-Werten aufgefüllt:

```
SELECT * FROM Kunde
   NATURAL OUTER JOIN Auftrag
```

Nun erscheinen auch Kunden ohne aktuellen Auftrag im Resultat:

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 Joins in SQL

- Alle Alternativen:
  - `LEFT JOIN` or `LEFT OUTER JOIN`
    die linke Relation ist verlustfrei
  - `RIGHT JOIN` or `RIGHT OUTER JOIN`
    die rechte Relation ist verlustfrei
  - `FULL JOIN` or `FULL OUTER JOIN`
    beide Relationen verlustfrei
- Outer Joins können kombiniert werden mit `NATURAL`, `ON`, `USING`
- Übersicht:

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 'H%'
UNION
SELECT * FROM Studenten WHERE Name LIKE 'H%'
```

- Bei einigen Anbietern ist auch möglich:

```
SELECT * FROM Mitarbeiter UNION Studenten WHERE ...
```

- Varianten:
  - UNION: Vereinigung mit Duplikat-Elimination
  - UNION ALL: Vereinigung ohne Duplikat-Elimination
  - INTERSECT: Schnittmenge
  - EXCEPT, MINUS: Differenz
- Achtung: Zwei Tupel sind nur dann gleich, wenn alle ihre Attribut-Werte gleich sind

UNION, INTERSECT, EXCEPT

Bemerkungen:

- 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

Bemerkungen:

- 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

Agenda

1. Einleitung
2. Grundoperationen
3. Abgeleitete Operationen
4. Implementierung in SQL
5. Änderungsoperationen in SQL

Einführung

- Bisher: Nur Anfragenan 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 x%

UPDATE

- Syntax:

```
UPDATE   <Relation>
SET      <Attribut 1> = <Ausdruck 1>,
         <Attribut 2> = <Ausdruck 2>,
         ...
WHERE    <Bedingung>
```

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

UPDATE

- UPDATE ist i.a. eine Mehrtupel-Operation

Beispiel: Update

Alle Angestellte gleich behandeln:

```
UPDATE Angestellte
SET Gehalt = 6000
```

Oder besser nicht absolut sondern relativ zu bisherigen Wert:

```
UPDATE Angestellte
SET Gehalt = Gehalt * 1.02
```

Lieber nur einen Angestellten erhöhen?

```
UPDATE Angestellte
SET Gehalt = 6000
WHERE PNr = 7
```

UPDATE

- Achtung: UPDATE-Operationen können zur Verletzung von Integritätsbedingungen führen
- In diesem Fall erfolgt ein Abbruch der Operation mit Fehlermeldung (siehe auch später Transaktionskonzept)

DELETE

- 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: Delete

Löschen aller Angestellten mit Gehalt 0

```
DELETE FROM Angestellte WHERE Gehalt = 0
```

INSERT

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

```
INSERT INTO <Relation> [(<Attr1>, <Attr2>, ...)]
VALUES (<Konstante1>, <Konstante2>, ...)
```

- Wirkung:
  - Ist die optionale Attributliste hinter dem Relationennamen angegeben, dann...
    - ... können unvollständige Tupel eingefügt werden: Nicht aufgeführte Attribute werden mit `NULL`-Werten belegt
    - ... werden die Werte durch die Reihenfolge in der Attributsliste zugeordnet
  - Wirkung (Forts.): Ist die optionale Attributliste hinter dem Relationennamen nicht angegeben, dann...
    - ... können unvollständige Tupel nur durch explizite Angabe von `NULL`-Werten eingegeben werden
    - ... werden die Werte durch die Reihenfolge in der DDL-Definition der Relation zugeordnet (mangelnde Datenunabhängigkeit!)

INSERT

Beispiel: Insert (Konstante)

Relationen-Schema: Angestellte (PNr, Name, Vorname, AbtNr)

- Mit Attributliste:

```
INSERT INTO Angestellte (Vorname, Name, PNr)
VALUES (‘Donald’, ‘Duck’, 123)
```

- Ohne Attributliste:

```
INSERT INTO Angestellte
VALUES (123, ‘Duck’, ‘Donald’, NULL)
```

- Ergebnis:

PNr	Name	Vorname	AbtNr
123	Duck	Donald	NULL

INSERT

- Syntax zum Einfügen berechneter Tupel:

```
INSERT INTO <Relation> [(<Attr1>, <Attr2>, ...)]
(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)

INSERT

Beispiel: Insert (berechnet)

Füge alle Lieferanten in die Kunden-Relation mit Kontostand 0 ein

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

```
INSERT INTO Kunde
(SELECT DISTINCT LName, LAdr, 0 FROM Lieferant)
```