

Data Warehousing und Data Mining

Materialisierte Sichten I
Optimierung mit MV



Ulf Leser

Wissensmanagement in der
Bioinformatik



GROUP BY

```
SELECT    region, product, SUM(amount)
FROM      sales, ...
...
GROUP BY  region, product
```

- Gruppierung
 - **Partitioniere** die Tupel der Relation nach den Attributen der GROUP BY Klausel
 - Gruppierungsattribute: G
 - Berechne die Aggregatfunktion für jede Teilmenge
 - für jede Gruppe entsteht eine Zeile in der Ergebnisrelation
- Problem: Wie macht man das bei **sehr vieler Tupeln**?
 - Tupelstrom kann nicht im Hauptspeicher gehalten werden



Hashbasierte Implementierung

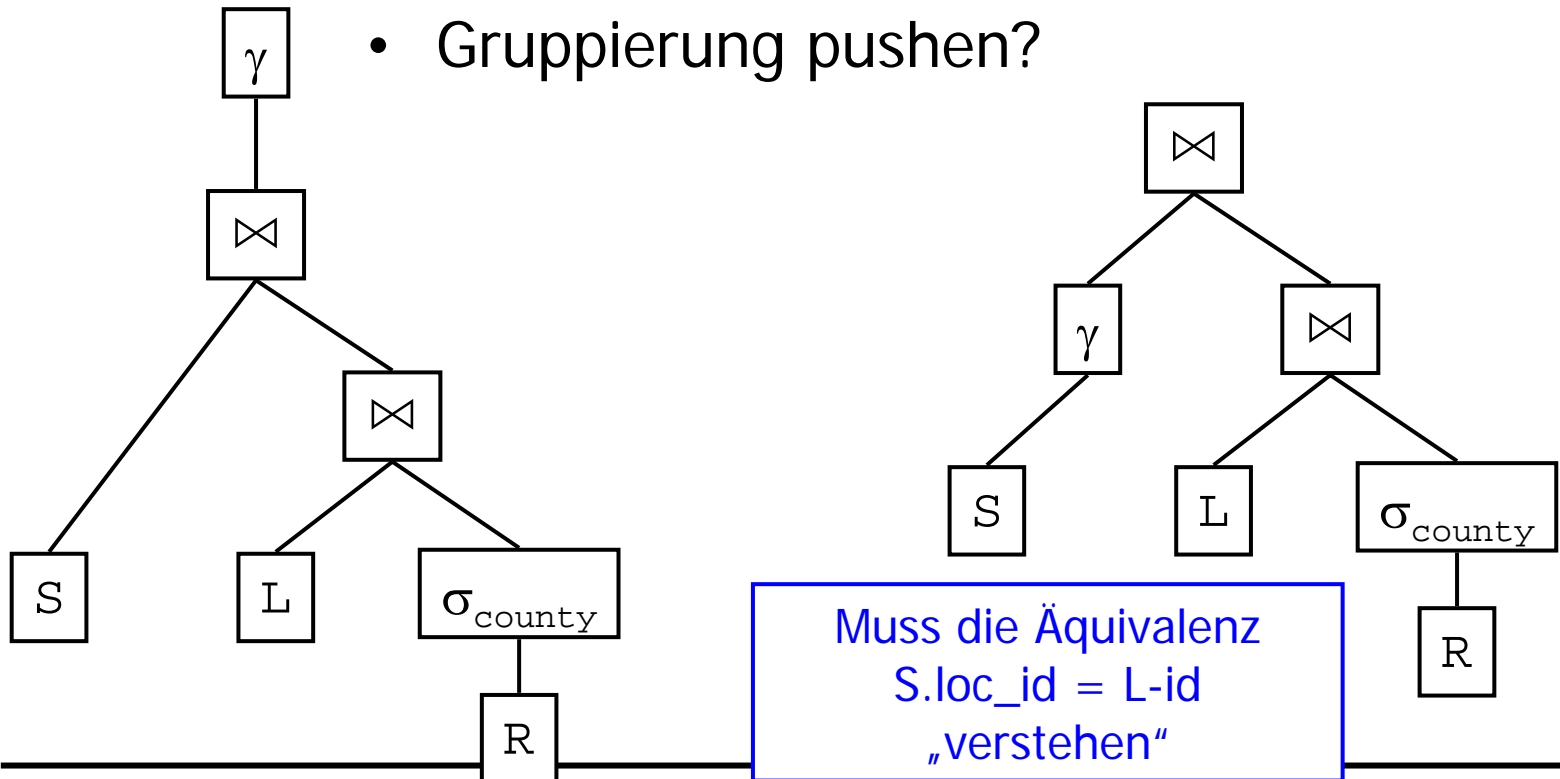
- Implementierung über Hashing
 - Lese Tupelstrom und verwende Wert von **G als Index in Hashtabelle**
 - Benötigt eine geeignete Hashfunktion – am Anfang ist unklar, wie viele verschiedene Werte von G es gibt
 - Erzeugt einen Bucket pro Wert von G
 - Iteriere über alle Buckets und wende Aggregatfunktion an
- Problem: Evt. müssen **alle Buckets im Hauptspeicher** gehalten werden
 - Bei distributiven (und algebraischen) Funktionen muss nur ein (wenige) Wert pro Bucket gehalten werden
 - Bei holistischen Funktionen müssen wir **effektiv alle Tupel im Speicher** halten
 - Denn man weiß nicht, wann das letzte Tupel einer Partition kommt

Sortierung

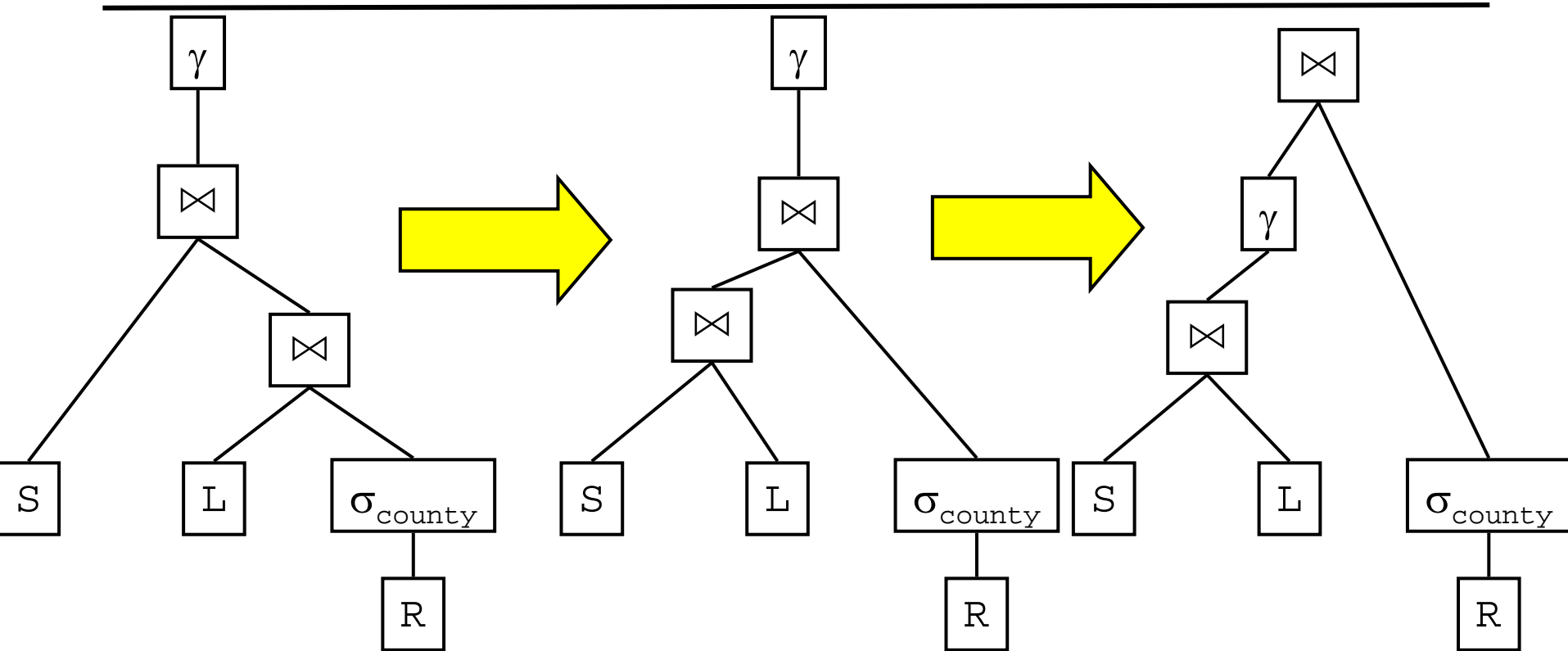
- Implementierung durch Sortierung
 - **Sortiere den Tupelstrom** nach den Werten von G
 - Im Hauptspeicher oder mit externer Sortierung
 - Lies den sortieren Tupelstrom
 - Tupeln puffern oder sofort aggregieren
 - Wenn sich Werte von G ändern beginnt eine neue Partition
 - Ggf: Berechne Aggregatfunktion
 - **Pipelining**: Wert kann sofort weitergereicht werden
- Vorteil: Benötigt im schlimmsten nur so viel Platz wie die größte Partition
 - Bei distributiven Funktionen sogar **nur einen Wert**
- Nachteil: Erfordert Sortierung

Erster Versuch

```
SELECT S.product_id, L.id, SUM(amount)
FROM sales S, location L, region R
WHERE S.loc_id = L.id AND
      L.region_id = R.id AND
      R.country = 'BRD'
GROUP BY S.product_id, L.id
```

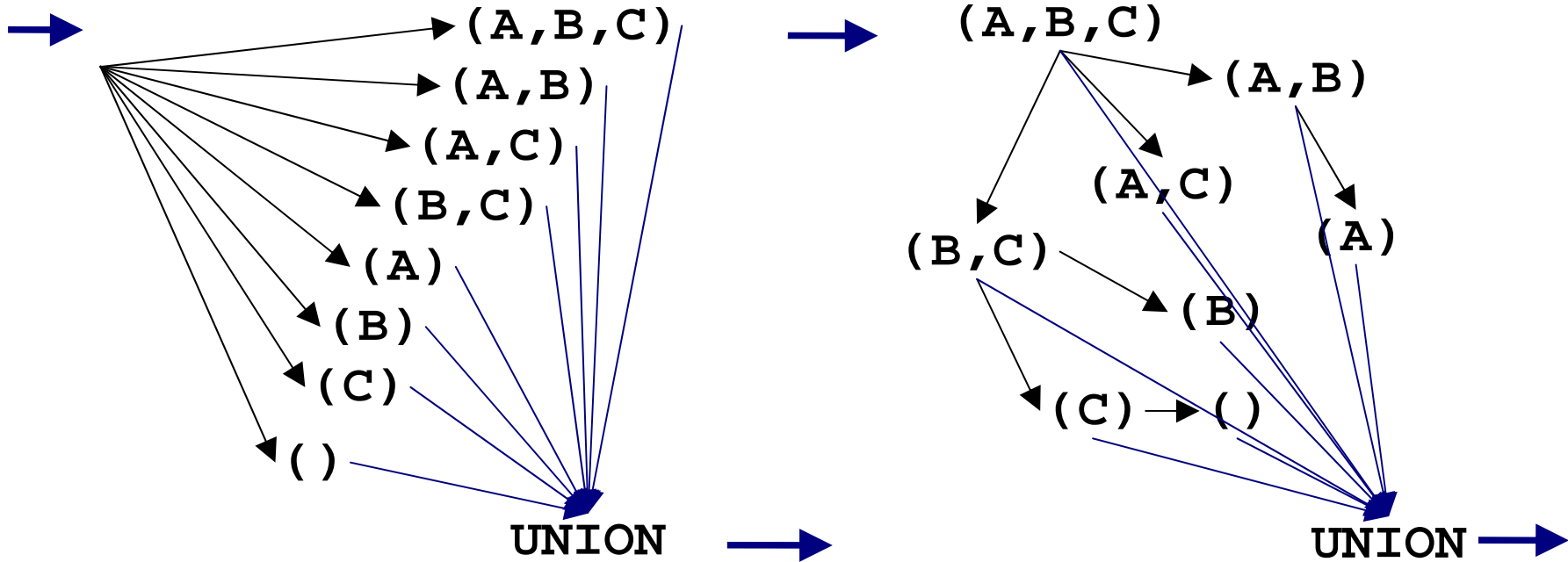


Alternative



- Bedingung: **L.region_id** muss im Ergebnis der Gruppierung sein
 - Z.B Umschreiben in `SELECT s.product_id, L.id, max(l.region_id)`
- Pushen oder nicht? **Kostenbasierte Entscheidung**
 - Der Join $S \bowtie L$ ist größer im dritten Plan, da der Filter auf ‚BRD‘ erst später erfolgt
 - Dafür gibt es weniger Tupel im ersten Zwischenergebnis

Realisierung CUBE



Naiver Ansatz

Ausnutzen von
(direkter) Ableitbarkeit

Mögliche Tricks

1. Smallest-parent

- Berechne Gruppierung aus dem **Elternteil mit den wenigsten Tupeln**
- Beispiel: (A) kann aus (AB), (AC) oder (ABC) berechnet werden
 - (ABC) hat sicher am meisten Tupel, aber $|AB| \geq |AC|$
- Gewinn: Weniger Rechenaufwand, eventuell können abgeschlossene Gruppierungen früher verdrängt werden

2. Cache-results

- Berechne Ketten von ableitbaren Gruppierungen
- Halte dazu **Ergebnisse im Hauptspeicher**, um IO zu sparen
- Beispiel: (ABC) \rightarrow (AB) \rightarrow (B) \rightarrow ()
 - Dazu muss (ABC) und (AB) in den Hauptspeicher passen (oder sortieren)
- Gewinn: Weniger IO

3. Amortize-scans

- Wenn schon eine Gruppierung geladen werden muss, berechne **möglichst viele der Kinder** gleichzeitig
- Beispiel: Aus (ABC) berechne mit einem Scan (AB), (BC), (AC)
- Gewinn: Weniger Scans, weniger IO

Mögliche Tricks 2

4. Share-sorts

- Wenn eine Gruppierung G sortiert gelesen werden kann, berechne alle **Gruppierungen mit derselben Attributreihenfolge**
 - Alle Gruppierungen, deren Attribute ein Präfix von G sind
- Beispiel: $(ABC) \rightarrow (AB) \rightarrow (A) \rightarrow ()$, aber nicht $(ABC) \rightarrow (BC)$
- Gewinn: Keine zusätzlichen Sortierungen (und Gruppierung mit Sortierung war speicherplatzeffizienter als Hashen)

5. Share-partitions

- Wenn eine hashbasierte Gruppierung aus Speichermangel in Partitionen zerlegt erfolgt, dann **benutze dieselben Partitionen** für alle ableitbaren Gruppierungen
- Berechnet mehrere GROUP BYs auf einer Menge von Partitionen
- Beispiel: (ABC) für Werte $A < 10$, dann dito $\rightarrow (AB) \rightarrow (A) \rightarrow ()$
- Gewinn: Weniger IO, weniger Hashen

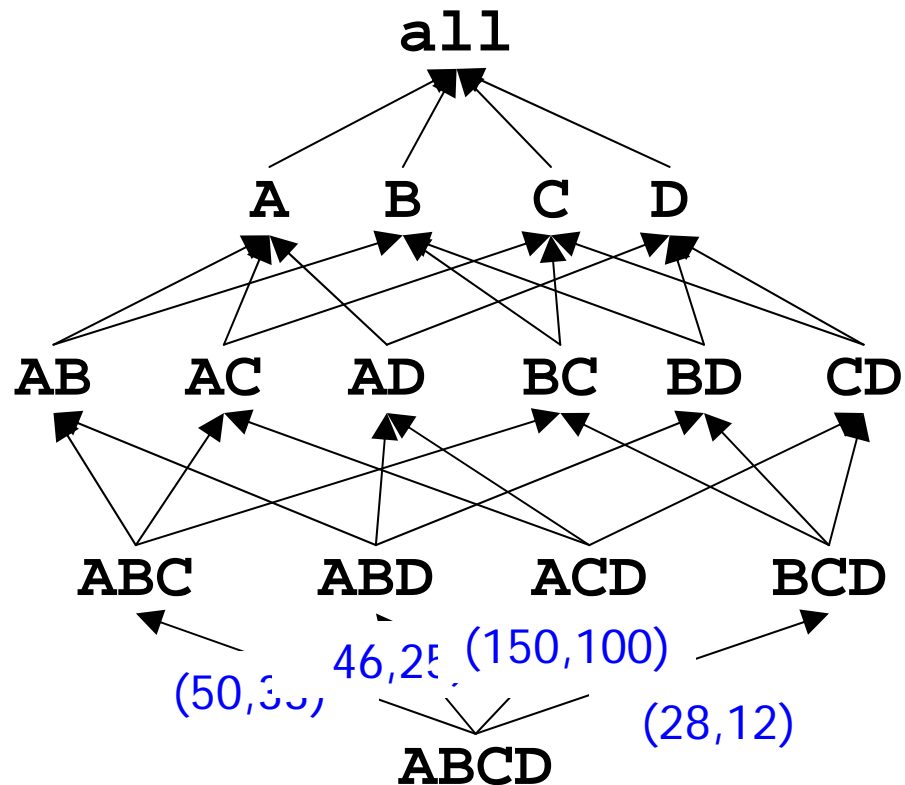
PipeSort [AAD+96]

- Die verschiedenen Tricks stehen **offensichtlich in Konflikt**
 - Der kleinste Vorfahr muss nicht die gleiche Sortierung haben
 - In einem Scan alle Kinder berechnen, obwohl man nicht der kleinste Elternteil ist
 - ...
- Algorithmen müssen sich auf einige der Tricks konzentrieren
- Beispiel: **PipeSort**
 - Eingabe ist das **Aggregationsgitter** der Gruppierung
 - Berechnet Reihenfolge und Sortierung der Gruppierungen
 - Benutzt **Share-Sorts und Smallest-Parent**
 - Benötigt Schätzungen über Kardinalität der Gruppierungen
 - Für Smallest-Parent

Gewichtetes Aggregationsgitter

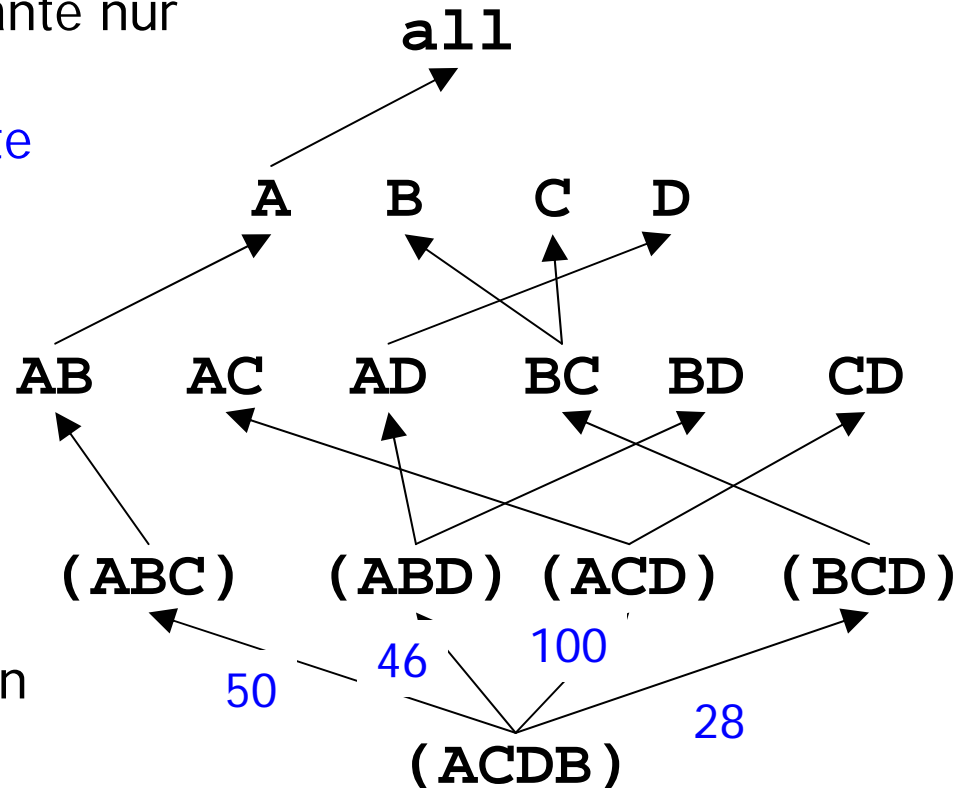
- Jede Kanten $G_i \rightarrow G_j$ hat zwei Kosten
 - S_{ij} : Berechnung von G_j aus G_i mit Umsortierung
 - A_{ij} : Berechnung von G_j aus G_i ohne Umsortierung

Hier ist noch keine Sortierung festgelegt!



Ziel

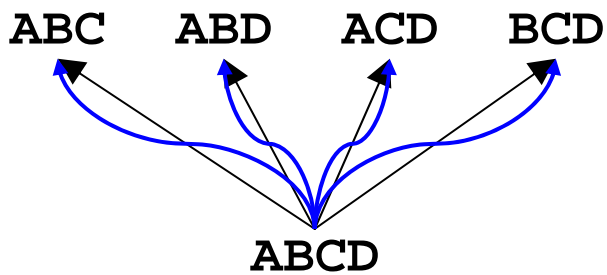
- Wir wollen den Subgraphen finden, für den gilt
 - Jeder Knoten hat nur eine eingehende Kante
 - Jeder Knoten hat eine festgelegte Sortierung
 - Damit ergibt sich für jede Kante nur noch ein Gewicht
 - Der **Subgraph soll die kleinste Kantensumme** haben
- Natürlich NP-hard
- Bemerkung
 - Da nur direkte Ableitungen ausgenutzt werden, kann **nur ein Nachfahre die ganze Sortierung „erben“**
 - Partielle Sortierungen können auch berücksichtigt werden



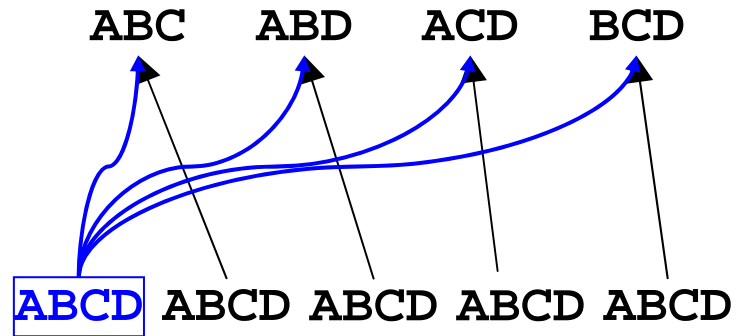
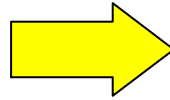
Ebene für Ebene

- Betrachte jeden Ebenenübergang im gewichteten Gitter
- Wir suchen eine optimale Teillösung, die
 - Für jeden Kindknoten genau **eine eingehende Kante** hat
 - Die **Sortierung aller Elternknoten** festlegt
- Reduktion auf **Bipartites Matching**
 - „Kopiere“ Elternknoten
 - \forall Elternknoten mit n ableitbaren Gruppen, jeweils mit A/S Kanten
 - Lege 1 Kopie an, die alle A-Kanten behält
 - Lege n Kopien an, die jeweils eine S-Kante behalten
 - Dies ist ein bipartiter Graph
 - Suche ein **optimales bipartites Matching**
 - Eine maximal „leichte“ Menge von Kanten so, dass nie zwei Kanten einen Knoten gemeinsam haben und alle Kindknoten durch genau eine Kante erreicht werden
 - Verschmelze gleiche Knoten wieder
 - Lege Sortierreihenfolge aller Eltern durch ausgehende A-Kante fest

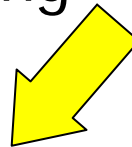
Beispiel – Level 1



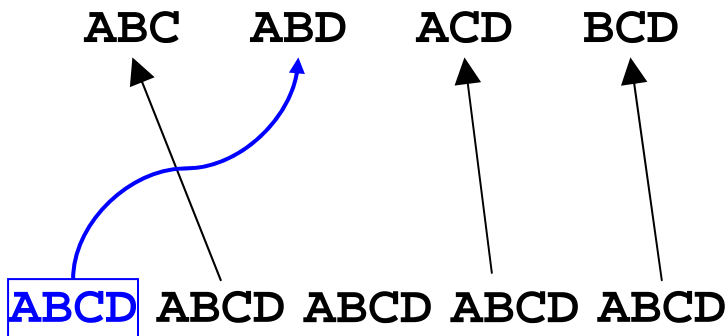
Kopieren



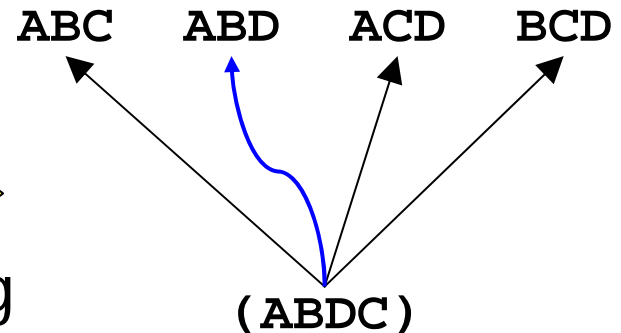
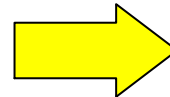
Matching



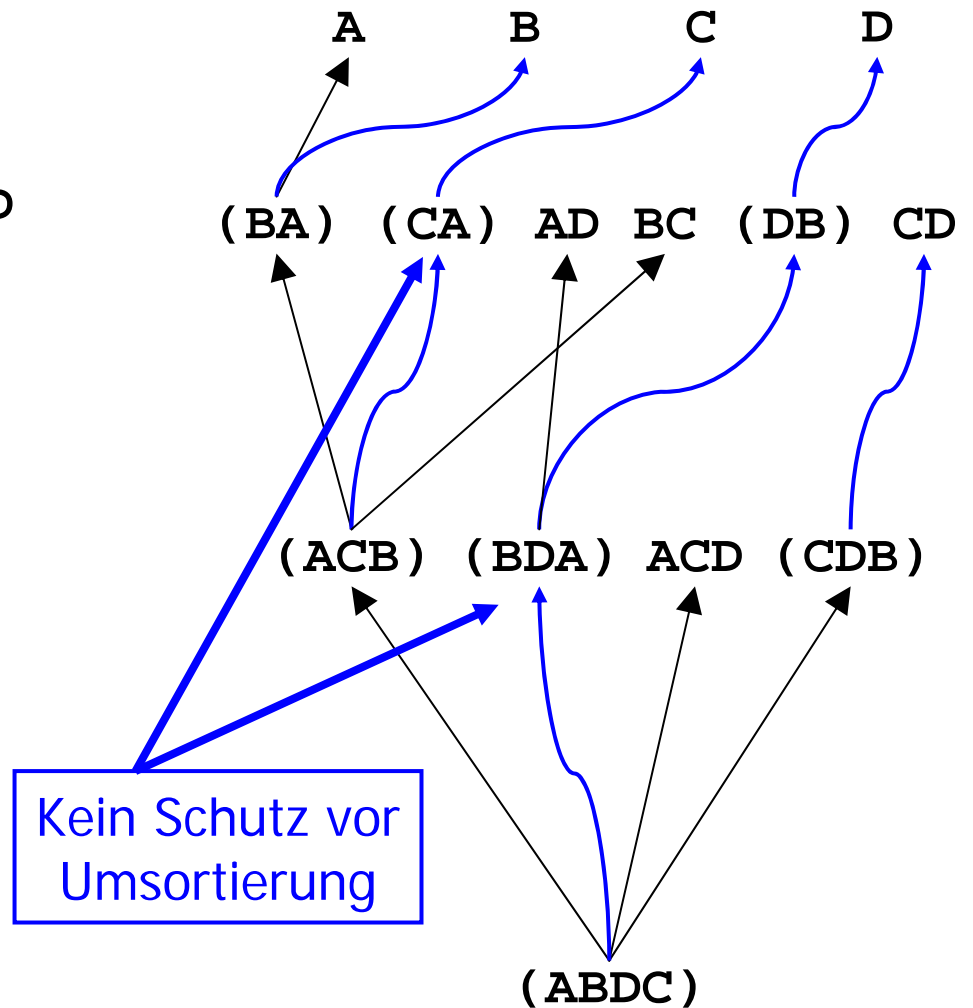
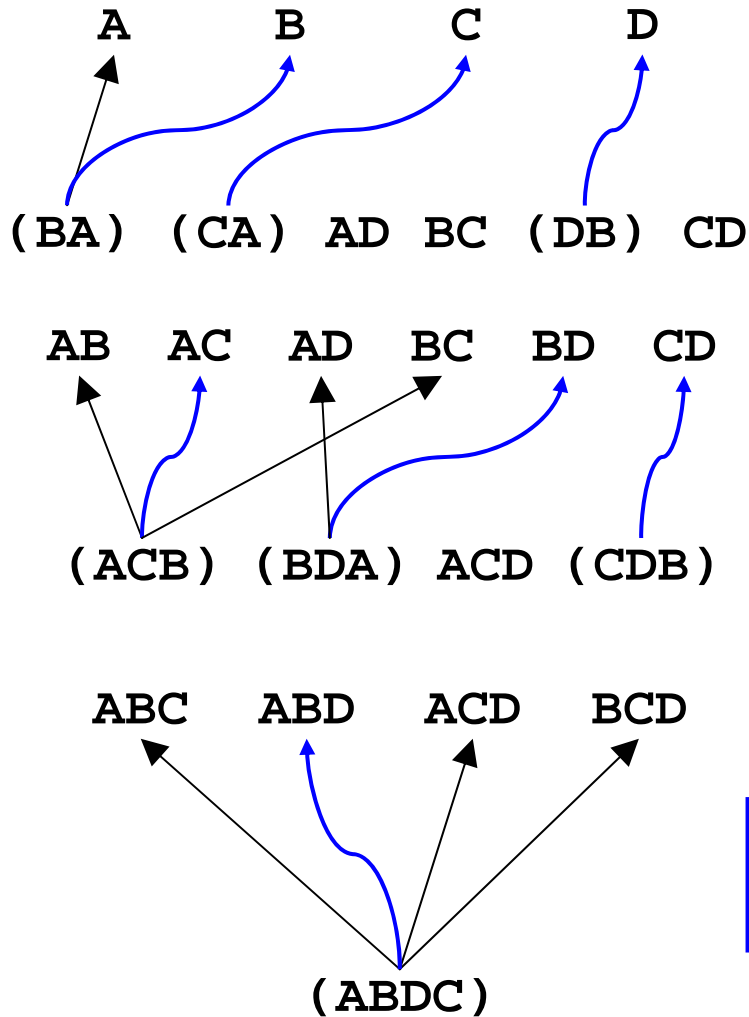
Nur eine A-Kante kann gewählt werden –
Sortierung steht fest



Merging



Ergebnis



Bewertung

- **Viele Heuristiken**, keine optimale Lösung
 - Keine Beachtung von Sortierungseffekte über mehrere Ebenen hinweg
 - Keine Garantie für die Anzahl notwendiger Sortierungen
- Platzverbrauch: Wenn ein Teilbaum nur einmal sortiert werden muss, braucht man nur **ein Tupel pro Gruppierung**
 - Bei distributiven Aggregatfunktionen, sonst ...
 - Tupel werden sortiert durch eine Pipeline von Gruppierungen geschickt
- Viele weitere Vorschläge (PipeHash, Multiway Aggregation, ...)

Iceberg Cubes

- Immer noch: Es gibt exponentiell viele Gruppierungen in einem CUBE Operator
 - 10 Dimensionen a 10 Ausprägungen = 1024 Gruppierungen mit zwischen 1 und 10^{10} Partitionen
 - „High dimensionality“ – die allermeisten Kombinationen müssen leer sein
 - So viel kann man gar nicht verkaufen
- Iceberg Cubes
 - Oftmals interessieren nur Aggregate oberhalb einer gewissen Grenze
 - ```
SELECT product_id, day_id, shop_id, SUM(amount)
FROM sales S
GROUP BY CUBE(product_id, day_id, shop_id)
HAVING COUNT(*) > threshold
```

# Berechnung

---

- Möglichkeit 1: CUBE normal berechnen, „leere“ Partitionen dann filtern
  - Ineffizient
- Beobachtung
  - (Gilt nicht für alle Aggregatfunktionen!)
  - Wenn für eine Partition  $P$  einer Gruppierung  $G$  und Aggregatfunktion  $f$  gilt, dass  $f(P) < t$ , dann muss für alle Partitionen  $P'$  jeder Gruppierung  $G'$  mit  $G \subseteq G'$  und  $P \subseteq P'$  gelten, dass  $f(P') < t$ 
    - A-Priori Eigenschaft
  - Durch die Hinzunahme weiterer Attribute in die Partitionierung werden die Partitionen höchstens kleiner
  - Das kann man zum **Prunen** ausnutzen

# Construction of Iceberg Cubes [BR99]

---

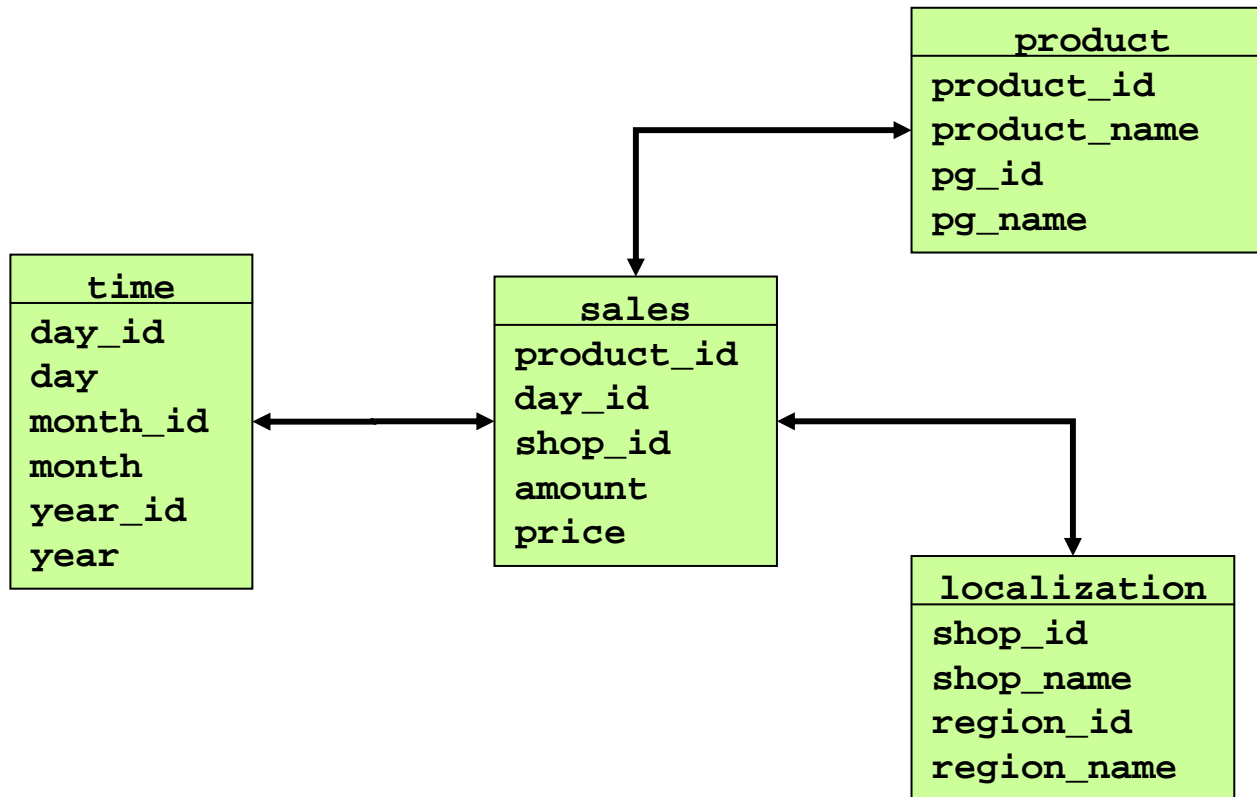
- Baut die Gruppierungen von ALL nach (ABCD)
- Iteriere über alle Dimensionen d
  - Partitioniere nach d
  - Iteriere über alle Partitionen P
    - Wenn  $\text{COUNT}(P) < t$ : Wegwerfen
    - Sonst; Gib  $(P, \text{COUNT}(P))$  aus; Steige rekursiv ab
      - Bilde alle Kombinationen  $(D, \dots)$
- Beobachtung
  - Benötigt viele Scans der gesamten Datenbasis (äußere Schleife)
  - Sinnvoll: Stufenweises Sortieren
  - Geschickte Implementierungen „switchen“ ab einem variablen Punkt in jeder Dimension auf Hauptspeichervarianten
- Viele weitere Algorithmen

# Inhalt dieser Vorlesung

---

- Materialisierte Sichten
- Logische Optimierung mit MV
- Kostenbasierte Optimierung mit MV
- Optimierung mit Aggregaten

# Beispiel



# Materialisierte Sichten

---

- Grundidee
  - Viele **Anfragen wiederholen sich häufig**
    - Bilanzen, Bestellungsplanung, Produktionsplanung
    - Kennzahlenberechnung
  - Viele Anfragen sind Variationen derselben Anfrage
    - Alle Verkäufe nach Produkt, Monat und Region
    - Alle Verkäufe in Region X nach Produkt, Monat und Shop
    - Alle Verkäufe in Region Y nach Produkt, Monat und Shop
- **Materialisierte Views (MV)**
  - Berechnen und **Speichern** einer Sicht
  - **Transparente Verwendung** in späteren Anfragen
    - Transparent = Benutzer muss die MV nicht direkt adressieren

# Arbeiten mit einem Cube ...

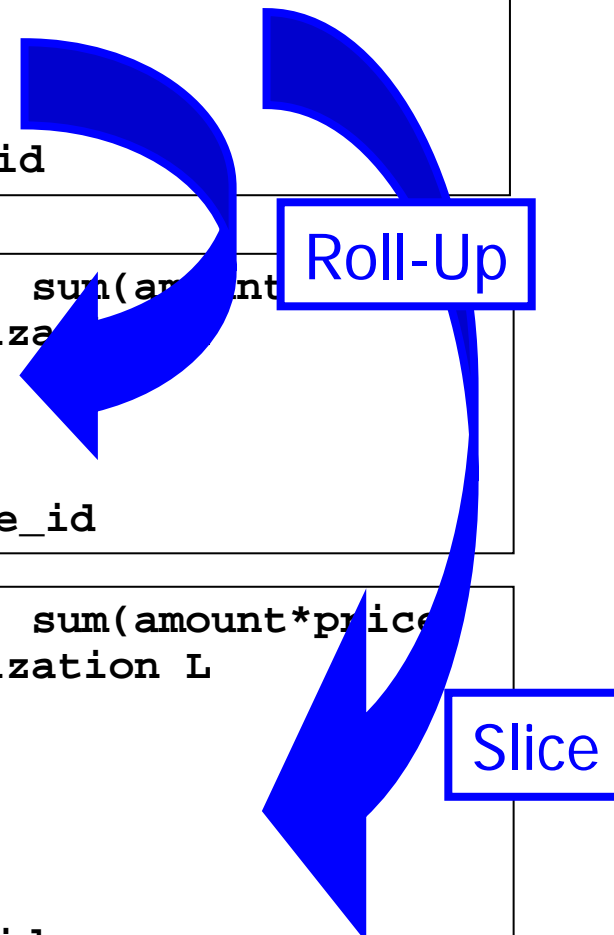
```
SELECT L.shop_id, P.product_id, T.day_id, sum(amount*price)
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id
GROUP BY L.shop_id, P.product_id, S.time_id
```

```
SELECT L.shop_id, P.product_id, T.day_id, sum(amount*price)
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id
GROUP BY L.region_id, P.product_id, S.time_id
```

```
SELECT L.shop_id, P.product_id, T.day_id, sum(amount*price)
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id AND
 P.pg_id = 159 AND
 T.year = ,1999`
GROUP BY L.shop_id, P.product_id, S.time_id
```

Roll-Up

Slice



# MV und Slicing

```
CREATE MATERIALIZED VIEW all_groups AS
SELECT L.shop_id, P.product_id, T.day_id, T.year, P.pg_id,
 sum(amount*price) as total
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id
GROUP BY L.shop_id, P.product_id, S.time_id
```

```
SELECT L.shop_id, P.product_id, T.day_id, sum(amount*price)
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id AND
 P.pg_id = 159 AND
 T.year = ,1999`
GROUP BY L.shop_id, P.product_id, S.time_id
```



```
SELECT shop_id, product_id, day_id, total
FROM all_groups A
WHERE pg_id = 159 AND
 year = ,1999`
```

# MV und Roll-Up

```
CREATE MATERIALIZED VIEW all_groups AS
SELECT L.shop_id, P.product_id, T.day_id, T.year, P.pg_id,
 sum(amount*price) as total
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id
GROUP BY L.shop_id, P.product_id, S.time_id
```

```
SELECT L.shop_id, P.product_id, T.day_id, sum(amount*price)
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id
GROUP BY L.region_id, P.product_id, S.time_id
```



```
SELECT shop_id, product_id, day_id, total
FROM all_groups A
GROUP BY region_id, P.product_id, S.time_id
```

# MV und Roll-Up

```
CREATE MATERIALIZED VIEW all_groups AS
SELECT L.shop_id, P.product_id, T.day_id, T.year, P.pg_id,
 min(region_id) as region_id, sum(amount*price) as total
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id
GROUP BY L.shop_id, P.product_id, S.time_id
```

```
SELECT L.shop_id, P.product_id, T.day_id, sum(amount*price)
FROM sales S, time T, product P, localization L
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id
GROUP BY L.region_id, P.product_id, S.time_id
```



```
SELECT shop_id, product_id, day_id, total
FROM all_groups A
GROUP BY region_id, P.product_id, S.time_id
```

# Themen

---

- Welche Views soll man materialisieren?
  - MVs kosten: Platz und Aktualisierungsaufwand
  - Wahl der optimalen MVs hängt von Workload ab
  - Algorithmen zur **Auswahl der optimalen Menge**
- Wie hält man MV aktuell?
  - MV nachführen, wenn sich Basistabellen ändern
  - U.U. schwierig: Aggregate, Joins, Outer-Joins, ...
    - Algorithmen zur **inkrementellen Aktualisierung**
- **Wann kann/soll man welche MV verwenden?**
  - **Wann** kann man MV verwenden?
    - Query Containment und Ableitbarkeit
  - **Welche** MV kann man verwenden?
    - Logische Optimierung, Query Rewriting
  - Wann **soll** man MV verwenden?
    - Kostenbasierte Optimierung

# Inhalt dieser Vorlesung

---

- Materialisierte Sichten
- **Logische Optimierung mit MV**
  - Einschub: Datalog Notation
  - Query Containment
  - Depth-First Algorithmus
  - Ableitbarkeit und Query Rewriting
- Kostenbasierte Optimierung mit MV
- Optimierung mit Aggregaten

# Kürzere Schreibweise

---

- Wir betrachten nur **konjunktive Anfragen**
  - Equi-joins und Bedingungen mit  $=, <, >$  zwischen Attribut und Wert
  - Kein NOT, EXISTS, GROUP BY,  $\neq$ ,  $X > Y$ , ...
- Schreibweise: **Datalog**
  - **`q(X,Y) :- sales(X,A,B,C), time(A,Y,D), D>1999;`**
  - SELECT Klausel
    - Regelkopf, exportierte Variable
  - FROM Klausel
    - Relationen werden zu Literalen in Prädikatenschreibweise
    - Attribute werden über **Position statt Name** adressiert
  - WHERE Klausel
    - Joins: gleiche Variablen an mehreren Stellen
    - Bedingungen mit „ $>, <$ “ werden explizit angegeben
    - Gleichheitsbedingungen „Attribut = Wert“ werden durch Konstante in Literal angegeben

# SQL – Datalog

```
SELECT S.price, L.region_name
FROM sales S, time T, localization L, product P
WHERE S.day_id = T.day_id AND
 S.product_id = P.product_id AND
 S.shop_id = L.shop_id AND
 L.shop_name = 'KB' AND
 T.year > 1999
```



```
q(P, RN) :-
 sales(SID, PID, FID, LID, P, ...),
 time(TID, D, M, Y),
 localization(LID, 'KB', RN),
 product(PID, PN, PGN),
 Y > 1999
```

# Begriffe

## Definition 2.2

Sei  $V$  eine Menge von Variablensymbolen und  $C$  eine Menge von Konstanten. Eine *konjunktive Datalog-Anfrage*  $q$  ist eine Anfrage der Form:

$$q(v_1, v_2, \dots, v_n) \quad :- \quad r_1(w_{1,1}, \dots, w_{1,n_1}), r_2(w_{2,1}, \dots, w_{2,n_2}), \dots, \\ r_m(w_{m,1}, \dots, w_{m,n_m}), k_1, \dots, k_l;$$

mit extensionalen Prädikaten  $r_1, r_2, \dots, r_m$ ,  $v_i \in V$ ,  $w_{i,j} \in V \cup C$  und  $\forall v \in V : \exists i, j : w_{i,j} = v$  und  $\forall c \in C : \exists i, j : w_{i,j} = c$ . Alle  $k_i$  haben für beliebige  $v_1, v_2 \in V$  und  $c \in C$  die Form  $v_1 < c$ ,  $v_1 > c$ ,  $v_1 = c$  oder  $v_1 = v_2$ . Dann ist:

- $head(q) = q(v_1, v_2, \dots, v_n)$  der Kopf von  $q$ ,
- $body(q) = r_1(w_{1,1}, \dots), r_2(w_{2,1}, \dots), \dots, r_m(w_{m,1}, \dots)$  der Rumpf von  $q$ ,
- $exp(q) = \{v_1, v_2, \dots, v_n\}$  die Menge der exportierten Variablen von  $q$ ,
- $var(q) = V$  die Menge aller Variablen von  $q$ ,
- $const(q) = C$  die Menge aller Konstanten von  $q$ ,
- $sym(q) = C \cup V$  die Menge aller Symbole von  $q$ ,
- $r_1, r_2, \dots, r_m$  sind die *Literale* von  $q$ , und
- $cond(q) = k_1, \dots, k_l$  sind die Bedingungen von  $q$ .

# Beispiel

---

```
q(P,RN) :-
 sales(SID,PID,TID,LID,P,_,_),
 time(TID,D,M,Y),
 localization(LID,`KB`,RN),
 product(PID,PN,PGN),
 Y > 1999
```

- `sales`, `time`, .. sind **Prädikate**
  - Relationen des Schemas
- `sales(SID,PID,...)`, `time(TID,D,M,Y)` sind **Literale**
  - Eine Anfrage kann ein Prädikat mehrmals enthalten - mehrere Literale desselben Prädikats
  - Literale sind eindeutig in einer Anfrage, Prädikate nicht
- Variable, die nicht interessieren, kürzt man mit „\_“ ab
- `q` ist **sicher**, wenn jede exportierte Variable im Rumpf vorkommt

# Kein echtes Datalog

---

- Datalog kennt noch mehr
  - Disjunktion und Vereinigung
  - Andere Joins als Equi-Joins
  - Rekursive Anfragen
    - **Extensional predicates**: Prädikate, deren Extension in der Datenbank vorliegen
    - **Intensional predicates**: Prädikate, die zur Laufzeit berechnet werden
      - SQL: Views
    - Verwendet ein intensionales Prädikat sich selber im Rumpf, wird dadurch eine rekursive Anfrage definiert
      - „Normales“ SQL: Verboten
      - Rekursives SQL: Views mit Namen

# Was müssen wir tun?

---

- Wir betrachten den einfachen Fall – eine Query  $q$  und einen MV  $v$ 
  - Erweiterung: Verwendung mehrerer MV für eine Query
    - Nicht viel schwieriger
  - Erweiterung: Beantwortung der Anfrage **nur mit MV**
    - „Answering queries using views“
  - Siehe: HK Informationsintegration
- Frage: Kann man  $v$  verwenden, um  $q$  zu beantworten?
  - Dazu müssen wir eine Aussage über die Ergebnismengen von  $v$  und  $q$  machen
  - Die wollen wir aber aus den Definitionen von  $v$  und  $q$  ableiten
  - **Query Containment**

# Query Containment

---

- Gegeben  $q$  und  $v$
- Anfrageäquivalenz
  - Ist Ergebnis von  $v$  identisch dem Ergebnis von  $q$ ?
  - Kurz: Ist  $v$  äquivalent zu  $q$ ,  $v \equiv q$
- Anfragecontainment („enthalten in“)
  - Ist das Ergebnis von  $v$  im Ergebnis von  $q$  enthalten?
  - Kurz: Ist  $v$  in  $q$  enthalten,  $v \subseteq q$
- Offensichtlich gilt:  $v \subseteq q, q \subseteq v \Rightarrow v \equiv q$
- Definition von „äquivalent“, „enthalten in“ für Queries?

# Definition

---

- Definition

*Sei  $S$  ein Schema und  $q, v$  Anfragen gegen  $S$ :*

- Eine **Instanz** von  $S$  ist eine beliebige Datenbank  $D$  mit Schema  $S$ , geschrieben  $D^S$
- Das **Ergebnis** einer Query  $q$  in einer Datenbank  $D^S$ , geschrieben  $q(D^S)$ , ist die Menge aller Tupel, die die Ausführung von  $q$  in  $D^S$  ergibt
- $q$  ist **äquivalent zu  $v$** , geschrieben  $q \equiv v$ , gdw.  
 $q(D^S) = v(D^S)$  für jede mögliche Instanz  $D^S$  von  $S$
- $q$  ist **enthalten in  $v$** , geschrieben  $q \subseteq v$ , gdw.  
 $q(D^S) \subseteq v(D^S)$  für jede mögliche Instanz  $D^S$  von  $S$

- Bemerkung

- **Semantische Definition**: Es zählt das Ergebnis einer Query
  - Natürlich können wir nicht alle Instanzen aufzählen
- Wir wollen Enthalten-Sein beweisen, indem wir nur auf die **Definition** der Anfragen sehen

# Beispiele

---

## Regelköpfe werden unterschlagen

$q \equiv q$

$\text{product}(\text{PID}, \text{PN}, \text{PGID}, \text{Wasser}) \subseteq \text{product}(\text{PID}, \text{PN}, \text{PGID}, \text{PGN})$

$\text{product}(\text{PID}, \text{PN}, \text{PGID}, \text{PGN}) \not\subseteq \text{localization}(\text{SID}, \text{SN}, \text{RID}, \text{RN})$

$\text{product}(\text{PID}, \text{PN}, \text{PGID}, \text{PGN}), \text{PGN} > \text{Wasser} \subseteq \text{product}(\text{PID}, \text{PN}, \text{PGID}, \text{PGN})$

$\text{sales}(\text{SID}, \text{PID}, \dots, P, \dots), P > 80, P < 150 \not\subseteq \text{sales}(\text{SID}, \text{PID}, \dots, P, \dots), P > 100, P < 150$

$\text{sales}(\text{SID}, \text{PID}, \dots, P, \dots), P > 100, P \leq 150, P < 170, P \geq 150 \equiv \text{sales}(\text{SID}, \text{PID}, \dots, 150, \dots)$

$\text{sales}(\text{SID}, \text{PID}, \dots, P, \dots), \text{product}(\text{PID}, \text{PN}, \dots) \subseteq \text{sales}(\text{SID}, \text{PID}, \dots, P, \dots)$   
*(Bei Projektion auf sales)*

# Weitere Beispiele

$q_1(B,D) :- \text{path}(A,B), \text{path}(B,C), \text{path}(C,D), \text{path}(D,E)$   
 $q_2(A,C) :- \text{path}(A,B), \text{path}(B,C), \text{path}(C,D)$   
 $q_1 \subseteq q_2 ?$



$q_1(C,B) :- \text{path}(A,B), \text{path}(C,A), \text{path}(B,C), \text{path}(A,D)$   
 $q_2(X,Z) :- \text{path}(X,Y), \text{path}(Y,Z)$   
 $q_1 \subseteq q_2 ?$



$q_1(A,C) :- \text{path}(A,B), \text{path}(B,C)$   
 $q_2(A,C) :- \text{threeNodePath}(A,B,C)$   
 $q_1 \subseteq q_2 ?$



$q_1(B,D) :- \text{path}(A,B), \text{path}(B,C), \text{path}(C,D), \text{path}(A,E), \text{path}(E$   
 $q_2(A,C) :- \text{path}(A,C), \text{path}(C,E), \text{path}(E,A), \text{path}(A,B), \text{path}(D$   
 $q_1 \subseteq q_2 ?$



# Vorarbeiten

---

- Definition

- Ein *Symbol Mapping*  $h$  von einer Anfrage  $q_2$  in eine Anfrage  $q_1$  ist eine Funktion  $h: \text{sym}(q_2) \mapsto \text{sym}(q_1)$
- Für ein Literal  $l \in q_2$ ,  $l = \text{rel}(A_1, \dots, A_m)$  ist  $h(l)$  definiert als
$$h(l) := \text{rel}(h(A_1), \dots, h(A_m))$$

- Bemerkung

- Jedes Symbol Mapping ist eine Funktion, bildet also jedes Symbol aus  $q_2$  auf exakt ein Symbol aus  $q_1$  ab

# Containment Mappings

---

- Definition

*Ein **Containment Mapping** (CM)  $h$  von Anfrage  $q_2$  nach Anfrage  $q_1$  ist ein Symbol Mapping von  $q_2$  nach  $q_1$  für das gilt:*

1.  $\forall c \in \text{const}(q_2)$  gilt:  $h(c) = c$ 
  - Jede Konstante in  $q_2$  wird auf dieselbe Konstante in  $q_1$  abgebildet
2.  $\forall l \in q_2$  gilt:  $h(l) \in q_1$ 
  - Jedes Literal in  $q_2$  wird auf (mindestens) ein Literal in  $q_1$  abgebildet
3.  $\forall e \in \text{exp}(q_2)$  gilt:  $h(e) \in \text{exp}(q_1)$ 
  - Der Kopf von  $q_2$  wird auf den Kopf von  $q_1$  abgebildet
4.  $\text{cond}(q_1) \rightarrow \text{cond}(h(q_2))$ 
  - Die Bedingungen von  $q_1$  sind logisch restriktiver als die von  $q_2$

# Vom CM zum Query Containment

---

- *Theorem*
  - $q_1 \subseteq q_2$  gdw. es ein CM von  $q_2$  nach  $q_1$  gibt
- *Lemma*
  - $q_1 \Leftrightarrow q_2$  gdw. es ein CM von  $q_2$  nach  $q_1$  und ein CM von  $q_1$  nach  $q_2$  gibt
- *Beweise*
  - Nicht schwierig; über die Semantik von Anfragen
  - Literatur [CM77]
  - Ursprünglich zur **Anfrageminimierung** entwickelt
- **Richtungen beachten**
  - Containment Mapping von  $q_2$  nach  $q_1$
  - Bedingungen von  $q_1$  implizieren Bedingungen von  $q_2$

# Beispiel

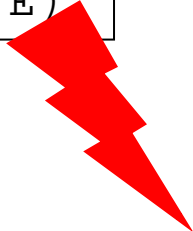
$q_1 \subseteq q_2$  ?

$q_2(A,C) :- \text{path}(A,B), \text{path}(B,C), \text{path}(C,D)$   
 $q_1(B,D) :- \text{path}(A,B), \text{path}(B,C), \text{path}(C,D), \text{path}(D,E)$

Mapping:  $A \rightarrow A, B \rightarrow B, C \rightarrow C, D \rightarrow D$

$q_2(A,C) :- \text{path}(A,B), \text{path}(B,C), \text{path}(C,D)$   
 $q_1(B,D) :- \text{path}(A,B), \text{path}(B,C), \text{path}(C,D), \text{path}(D,E)$

Mapping:  $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$



# Beispiel

$q_2(TID, P) \text{ :- sales}(SID, TID, P, \dots), \text{time}(TID, D, \dots), D > 28, D < 31$

$q_1(Y, Z) \text{ :- sales}(X, Y, Z, \dots), \text{time}(Y, U, \dots), U > 1, U < 30$

CM:  $SID \rightarrow X, TID \rightarrow Y, P \rightarrow Z, D \rightarrow U$   
 $h(D) = U$

Aber:  $U > 1 \wedge U < 30 \not\rightarrow h(D) > 28 \wedge h(D) < 31$

$q_1 \not\subseteq q_2$

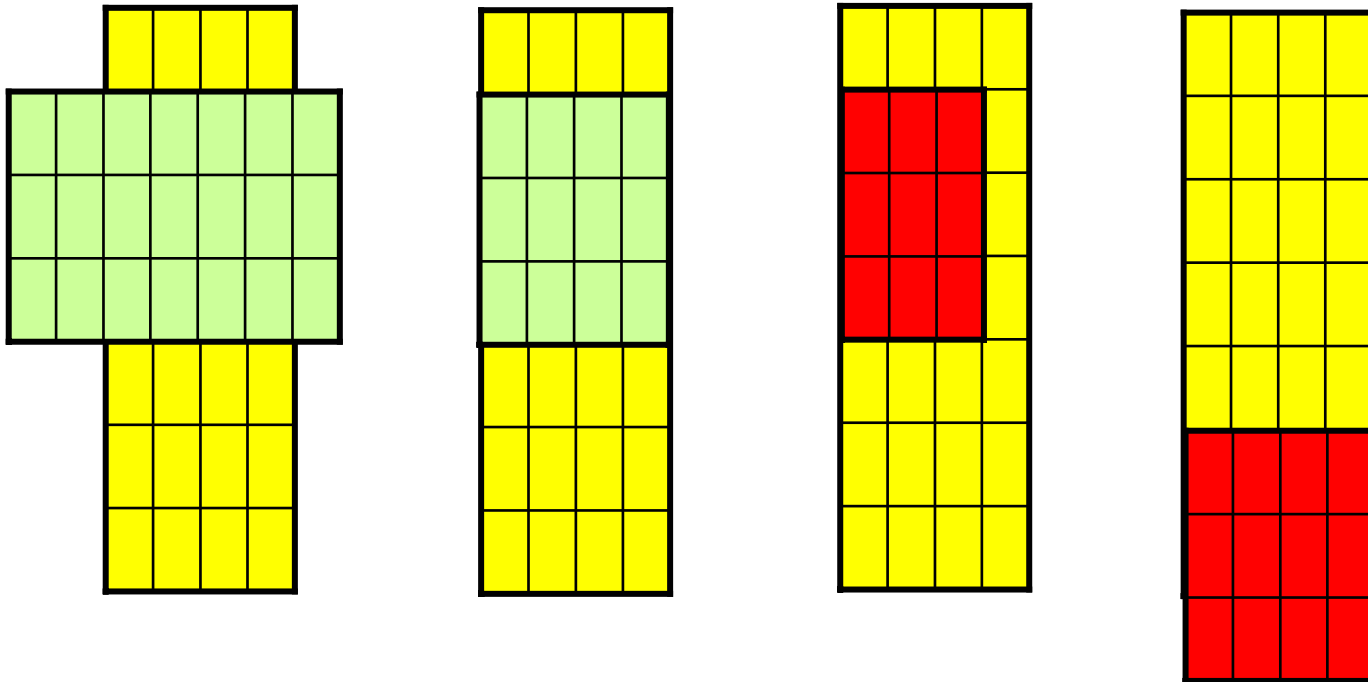
# Intuition

---

- Containment Mapping (CM) von  $q_2$  nach  $q_1$  ...
  1.  $\forall c \in \text{const}(q_2)$  gilt:  $h(c) = c$ 
    - Konstante dürfen sich nicht ändern (gleiche Selektionsbedingungen)
  2.  $\forall l \in q_2$  gilt:  $h(l) \in q_1$ 
    - **Zusätzliche Literale** in  $q_1$  sind nur Filter auf dem Ergebnis; Containment wird dadurch nicht beeinflusst
  3.  $\forall e \in \text{exp}(q_2)$  gilt:  $h(e) \in \text{exp}(q_1)$ 
    - Es müssen auch die **richtigen Variablen** ausgegeben werden;  $q_1$  darf weitere Variable exportieren, aber nicht weniger
  4.  $\text{cond}(q_1) \rightarrow \text{cond}(h(q_2))$ 
    - Bedingungen in  $q_1$  müssen **äquivalent oder strikter** sein (also höchstens Tupel wegfiltern) als die von  $q_2$

# Intuition 2

- $q_1$  darf nur
  - **Weniger Tupel** berechnen – mehr Joins, strengere Bedingungen
  - **Mehr Spalten** berechnen – andere Projektion



# Set-Semantik

---

- Alles gesagte gilt nur unter **Set-Semantik**
  - Das ist Standard in SQL, aber nicht in RDBMS
  - „Containment of **real** conjunctive queries“
- Beispiel
  - $q_1(X,Y) :- r(X,Y)$      $q_2(X,Y) :- r(X,Y), s(Y,Z)$
  - Set-Semantik:  $q_2 \subseteq q_1$
  - Aber: Im Ergebnis ist jedes Tupel  $(X,Y)$  aus  $r$  **so oft** enthalten, wie  $(Y, \_)$  in  $s$  enthalten ist
  - Unter Bag-Semantik gilt das Containment daher nicht
- **Bag-Semantik**
  - Anfragen sind nur dann äquivalent, wenn sie isomorph sind
    - Homomorphismen reichen nicht
  - Containment bei Anfragen mit Ungleichheit ist unentscheidbar (PODS2006)

# Inhalt dieser Vorlesung

---

- Materialisierte Sichten
- **Logische Optimierung mit MV**
  - Einschub: Datalog Notation
  - Query Containment
  - **Depth-First Algorithmus**
  - Ableitbarkeit und Query Rewriting
- Kostenbasierte Optimierung mit MV
- Optimierung mit Aggregaten

# Wie findet man Containment Mappings?

---

- $q \subseteq v$  gwd. es ein Containment Mapping von  $v \rightarrow q$  gibt
- Naives Verfahren
  - Für jedes Symbol Mapping  $s$  testen, ob  $s$  ein Containment Mapping ist
  - Sei  $m = |\text{sym}(q)|$ ,  $n = |\text{sym}(v)| \Rightarrow m^n$  Symbol Mappings
- Besser
  - **Literale müssen auf Literale** abgebildet werden
  - Also müssen alle Symbole jedes Literals in  $v$  auf die Symbole eines Literals in  $q$  der gleichen Relation abgebildet werden
  - Wir zählen mögliche Ziele für Literale auf
  - Dabei können wir gleich Bedingung 1 (und 3) testen
  - Übrig bleibt der Test, ob die **Teilabbildungen kompatibel sind**

# Suchraum

$q \subseteq v ?$

$$\begin{aligned} v &= a(\dots), b(\dots), b(\dots), c(\dots) \\ q &= b(\dots), c(\dots), a(\dots), b(\dots), d(\dots) \end{aligned}$$

Nummerieren

$$\begin{aligned} v &= a(\dots), b^1(\dots), b^2(\dots), c(\dots) \\ q &= b^1(\dots), c(\dots), a(\dots), b^2(\dots), d(\dots) \end{aligned}$$

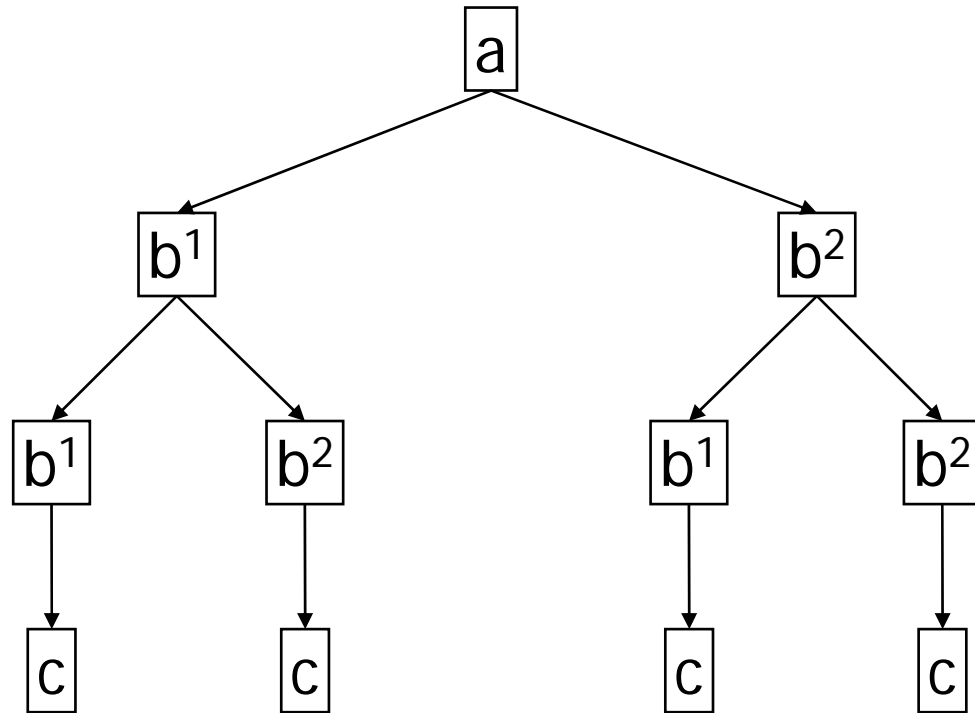
... jedes Literal von  $v$  muss auf mindestens ein Literal in  $q$  abgebildet werden ...

Möglichkeiten:

$$\begin{aligned} a &\rightarrow a \\ b^1 &\rightarrow b^1 \\ b^1 &\rightarrow b^2 \\ b^2 &\rightarrow b^1 \\ b^2 &\rightarrow b^2 \\ c &\rightarrow c \end{aligned}$$

# Suchraum

$v =$   
 $a(\dots),$   
 $b^1(\dots),$   
 $b^2(\dots),$   
 $c(\dots)$



# Partielle CM

---

- Definition

*Seien  $q$  und  $v$  Anfragen mit Literalen  $k \in q$  und  $l \in v$ . Ein **partielles Containment Mapping**  $h_{lk}$  von  $v$  nach  $q$  für  $l$  und  $k$  ist ein Symbol Mapping (von  $v$  nach  $q$ ), für das gilt*

- 1.  $\forall c \in \text{const}(l)$  gilt:  $h_{lk}(c) = c$*
- 2.  $h_{lk}(l) = k$*
- 3.  $\forall e \in \text{exp}(l)$  gilt:  $h_{lk}(e) \in \text{exp}(k)$*

- Bemerkung

- Abbildungen von Literalen auf Literale
- Den Bedingungsteil (Bedingung 4) verschieben wir auf später
- Der Test ist für feste  $l$  und  $k$  offensichtlich sehr einfach
- Überträgt sich natürlich auf **Gruppen aus mehreren Literalen**

# Kompatibilität und Vereinigung

---

- Definition

*Kompatibilität von partiellen CM*

Gegeben seien zwei Anfragen  $p$  und  $q$  mit  $l, i \in p$  und  $j, k \in q$  und  $l \neq i$  und  $j \neq k$ . Seien  $h_{l,j}$  und  $h_{i,k}$  zwei partielle Containment-Mappings. Man nennt  $h_{l,j}$  und  $h_{i,k}$  kompatibel, wenn  $\nexists v : v \in h_{l,j} \wedge v \in h_{i,k} \wedge h_{l,j}(v) \neq h_{i,k}(v)$ . ■

- Definition

*Vereinigung kompatibler partieller CM:*  $\mathbf{h} = \mathbf{h}_1 \oplus \mathbf{h}_2$

Gegeben seien zwei kompatible Teilmappings  $h_{l,j}$  und  $h_{i,k}$ . Das kombinierte Teilmapping  $h = h_{l,j} \circ h_{i,k}$  ist definiert als:

- $\forall v \in \text{sym}(h_{l,j}) : h(v) := h_{l,j}(v)$
- $\forall v \in \text{sym}(h_{i,k}) \setminus \text{sym}(h_{l,j}) : h(v) := h_{i,k}(v)$  ■

Das ist eindeutig, wenn CM kompatibel sind

# Algorithmus: Grobablauf

---

- Im Suchbaum entspricht der Level  $i$  dem Literal  $l_i$  von  $v$
- Alle potentiellen Zielliterale von  $l_i$  stehen in den Knoten des Levels  $i$  des Suchbaums
- $h_i$  : (wachsendes) partielles CM für die ersten  $i$  Literale von  $v$
- Algorithmus durchläuft den Suchbaum Depth-First und versucht, ein wachsendes CM aufzubauen
  - Starte mit einem leeren CM  $h_0$
  - Bei Wahl eines Zielliterals auf Level  $i$  wird versucht, das „größere“ CM  $h_i$  zu berechnen
  - $h_i$  wird berechnet durch Vereinigung des bisherigen  $h_{i-1}$  und des Mappings  $h$  von  $l_i$  zum gewählten Zielliteral
    - Wenn  $h$  und  $h_{i-1}$  kompatibel sind
  - Wenn man Level  $|v|=n$  erreicht hat und  $h_n$  berechnen konnte, hat man ein **Containment Mapping von  $v$  nach  $q$  gefunden**
    - Und damit  $q \subseteq v$  bewiesen

# Datenstruktur

---

- Positionen im Suchbaum wird in einem Stack gespeichert
- Stackelemente sind Tripel:  $(X, Y, Z)$ 
  - X: CM bis zu dieser Position
    - $=h_{i-1}$
  - Y: Weitere Mappingkandidaten im aktuellen Zweig
    - noch nicht untersuchte Geschwister von t
  - Z: Literale aus  $v$ , die noch nicht gemapped sind
    - Noch nicht besuchte Level des Suchbaums

# Depth-First Algorithmus

**Input:** Two queries  $q, v$ ;

**Output:** True, if  $q \subseteq v$ , else false;

```
1: R = new stack();
2: L = {l | l ∈ v}; % Alle Literale in v
3: l = L.first(); % Erstes Literal von v
4: H = {h | ∃ l' ∈ q ∧ l →h l'} % Alle pot. Ziele von l in q
5: R.push([], H, L \ l); % Startpunkt
6: repeat
7: (h, H, L) = R.pop(); % Hier weitermachen
8: if H = ∅ then % Keine Kandidaten - aufhören
9: return false;
10: else
11: g = H.first();
12: R.push(h, H \ g, L); % Geschwister später untersuchen
13: if compatible(h, g) then % CM kann erweitert werden
14: if L = ∅ then
15: return true; % Blatt erreicht - fertig
16: else
17: l = L.first(); % Innerer Knoten, Kinder pushen
18: H' = {h | ∃ l' ∈ q ∧ l ≥h l'};
19: R.push(h ⊕ g, H', L \ l);
20: end if;
21: end if;
22: end if;
23: until R = ∅; % Irgendwo Kand. ausgegangen
```

# Erklärung

---

- Sei  $h_{i-1}$  das aktuelle Mapping bis Level  $i-1$ ,  $l$  das  $i$ .te Literal aus  $v$
- Berechne die Menge  $H$  aller **potentiellen Zielliterale** für  $l$ 
  - Das sind die, für die es ein partielles CM von  $l$  aus gibt
- Teste für alle  $t \in H$ 
  - Bestimme  $h$ , das partielle CM von  $l$  nach  $t$ 
    - Wenn es das nicht gibt: springe zu 1;
  - Wenn:  $h$  und  $h_{i-1}$  sind kompatibel
    - Ist  $l$  das letzte Literal aus  $v$ ?
      - Ja, terminiere und melde Erfolg
    - $h_i = h_{i-1} \oplus h$
    - Untersuche nächstes Literal von  $v$ 
      - Abstieg im Baum
  - 1: Sonst
    - Sind noch Geschwister von  $t$  vorhanden?
      - Ja – testen
      - Nein – terminiere und melde Misserfolg

# Beispiel

$$v(A, B) = a(A, C), b(C, B), c(B, A)$$

$$q(X, X) = a(X, Y), b^1(Y, Z), b^2(Y, X), c^1(Z, X), c^2(X, X)$$

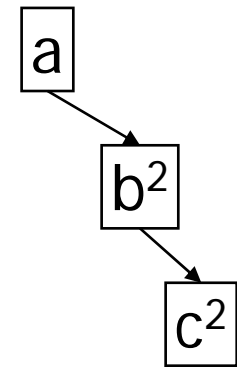
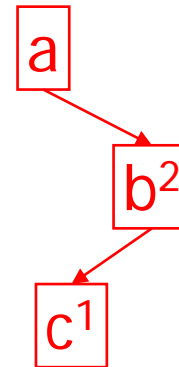
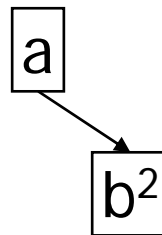
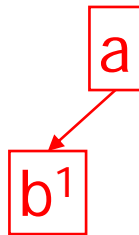
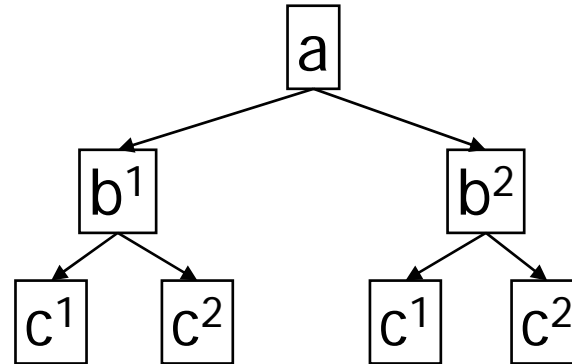
| CM bis Position<br>Aktuelles CM                                                            | Mapping der Literale von<br>v auf Zielliterale in q                        |
|--------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| $A \rightarrow X, C \rightarrow Y$                                                         | $a(A, C) \rightarrow$<br>$a(X, Y)$                                         |
| $A \rightarrow X, C \rightarrow Y,$<br>$C \rightarrow Y, B \rightarrow Z$                  | $a(A, C), b(C, B) \rightarrow$<br>$a(X, Y), b^1(Y, Z)$                     |
| $A \rightarrow X, C \rightarrow Y,$<br>$C \rightarrow Y, B \rightarrow X$                  | $a(A, C), b(C, B) \rightarrow$<br>$a(X, Y), b^2(Y, X)$                     |
| $A \rightarrow X, C \rightarrow Y, B \rightarrow X,$<br>$B \rightarrow Z, A \rightarrow X$ | $a(A, C), b(C, B), c(B, A) \rightarrow$<br>$a(X, Y), b^2(Y, X), c^1(Z, X)$ |
| $A \rightarrow X, C \rightarrow Y, B \rightarrow X,$<br>$B \rightarrow X, A \rightarrow X$ | $a(A, C), b(C, B), c(B, A) \rightarrow$<br>$a(X, Y), b^2(Y, X), c^2(X, X)$ |

Z nicht exportiert

B  $\rightarrow$  X, B  $\rightarrow$  Z  
nicht kompatibel

Fertig

# Beispielbaum



$a(A, C) \rightarrow$   
 $a(X, Y)$

$a(A, C), b(C, B) \rightarrow$   
 $a(X, Y), b_2(Y, X)$

$a(A, C), b(C, B), c(B, A) \rightarrow$   
 $a(X, Y), b^2(Y, X), c^2(X, X)$

$a(A, C), b(C, B) \rightarrow$   
 $a(X, Y), b^1(Y, Z)$

$a(A, C), b(C, B), c(B, A) \rightarrow$   
 $a(X, Y), b^2(Y, X), c^1(Z, X)$

# Was haben wir vergessen?

---

- Kompatibilität von Containment Mappings muss auch **Bedingungen in Queries** beachten

- (Richtige) Definition

*Seien  $q, v$  Anfragen und  $h_1, h_2$  partielle CM über Symbolmengen  $s_1, s_2$ , mit  $s_1 \subseteq \text{sym}(q)$ ,  $s_2 \subseteq \text{sym}(v)$ . Sei  $Z_1$  ( $Z_2$ ) die Menge von Bedingungen aus  $q$  ( $v$ ), die nur Symbole aus  $s_1$  ( $s_2$ ) beinhalten*

*Dann sind  $h_1$  und  $h_2$  **kompatibel**, wenn gilt:*

1.  $\forall X: (X \rightarrow A) \in h_1 \wedge (X \rightarrow B) \in h_2 \rightarrow A=B$
2.  $Z_1 \rightarrow (h_1 \cup h_2)(Z_2)$

- Algorithmus bleibt unberührt

# Komplexität

---

- Lemma  
*Seien  $q$  und  $v$  Anfragen an Schema  $S$  mit  $m=|q|$  und  $n=|v|$ . Die Suche nach einem Containment Mapping von  $v$  nach  $q$  durch Aufzählen möglicher Zielliterale benötigt  $O(n^m)$  Kompatibilitätstests von partiellen CM.*
- Beweis
  - Im Worst-Case entsprechen alle Literale beider Anfragen der gleichen Relation
  - Für jedes der  $n$  Literale aus  $v$  gibt es dann  $m$  mögliche Ziele in  $q$
  - Tests auf Kompatibilität und Berechnung der Vereinigung von partiellen CM ist polynomial
- Problem ist **NP vollständig**
  - Beweis: Reduktion auf „Exakt Cover“ [CM77]
- Bemerkung
  - Das Problem ist linear, wenn **keine Relation zweimal** in einer Anfrage vorkommt, quadratisch, wenn keine Relation dreimal vorkommt, ...
  - Ein (und nicht alle) CM zu finden kann viel schneller gehen (Depth-first)

# Wo sind die Ergebnisse?

---

- Ein Containment Mapping  $h$  von  $q$  nach  $v$  bestimmt auch das Ergebnis von  $q$  in den Ergebnissen von  $v$ 
  - Für jedes Tupel  $t$  im Ergebnis von  $v$
  - Baue ein Tupel  $t'$  gemäß der **umgedrehten Mappings  $h^{-1}$**  der Variablen in  $\text{exp}(q)$
- Wenn es mehrere CM von  $q$  nach  $v$  gibt, wiederhole das für jedes solche Mapping

# Zusammenfassung

---

- Query Containment ist NP-vollständig schon für konjunktive Anfragen
  - Aber linear, wenn Prädikate nicht mehrmals vorkommen
- Diverse Erweiterungen bekannt
  - Containment mit UNION, Negation, Aggregation, Rekursion, ...
    - **Höhere Komplexitätsklassen oder unentscheidbar**
- „Theoretisches“ Problem mit vielen Anwendungen
  - Anfrageoptimierung mit materialisierten Sichten
  - Informationsintegration
  - Semantic Caching
  - Anfrageminimierung

# Inhalt dieser Vorlesung

---

- Materialisierte Sichten
- Logische Optimierung mit MV
  - Einschub: Datalog Notation
  - Query Containment
  - Depth-First Algorithmus
  - **Ableitbarkeit und Query Rewriting**
    - Ableitbarkeit von Bedingungen
    - Ableitbarkeit von Joins
    - Ableitbarkeit von Aggregaten
- Kostenbasierte Optimierung mit MV
- Optimierung mit Aggregaten

# Anwendung

---

- Wie können nun zeigen, ob  $q \subseteq v$
- **Wie wenden wir das an?**
  - Sei  $q$  die Anfrage und  $v$  der materialisierte View
- **Möglichkeit 1:  $v \equiv q$  äquivalent**
  - Fertig:  $v$  als Ergebnis von  $q$  ausgeben
  - Test auf Äquivalenz erfordert 2 x Containment
  - Benutzung nur bei „**syntaktischer Äquivalenz**“
- **Möglichkeit 2:  $v \subseteq q$  (aber nicht umgekehrt)**
  - **$v$  ist ein partielles Ergebnis für  $q$**
  - $v$  berechnet nur korrekte, aber nicht alle Antworten von  $q$
  - Um  $q$  zu beantworten, müsste man  $v'$  berechnen so dass  $q \equiv v \cup v'$ 
    - Im Allgemeinen schwierig
  - **Keine Verwendung** außerhalb der Forschung

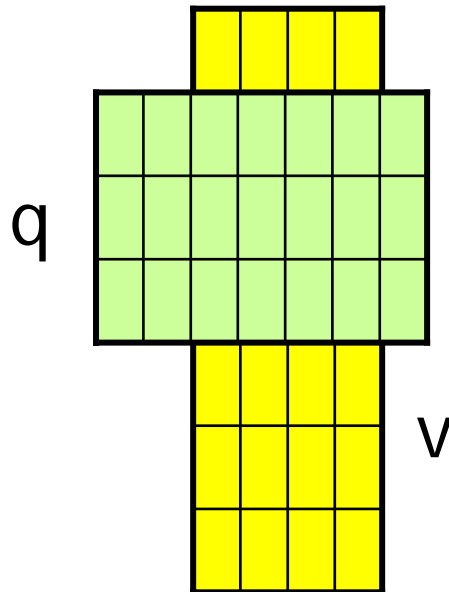
# Anwendung 2

---

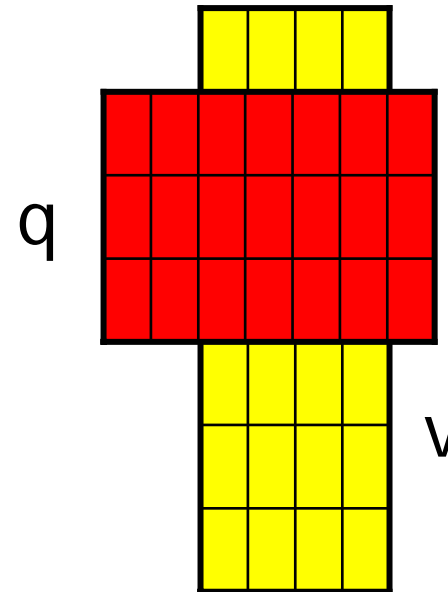
- Möglichkeit 3:  $q \subseteq v$  (aber nicht umgekehrt)
  - Ergebnis von  $q$  vollständig enthalten in  $v$
  - Nicht alle Tupel von  $v$  sind korrekte Ergebnisse für  $q$ , aber  $v$  berechnet alle Ergebnisse von  $q$
  - Manche Tupel müssen aus dem Ergebnis von  $v$  entfernt werden
- Probleme
  - **Vollständigkeit**:  $v$  enthält alle Tupel – auch die richtigen Attribute?
  - **Ableitbarkeit**: Wie findet man einen „Filter“  $F$  auf  $v$ , so dass nur die richtigen Tupel selektiert werden, also  $F(v) \equiv q$  ?

# Vollständigkeit

Query Containment



Ableitbarkeit von q



- Wir wollen **q beantworten**, in dem wir v filtern
- Daher muss v alle Attribute exportieren, die q exportiert
- Bedingungen für Containment Mappings müssen geändert werden

# Erweitertes Containment Mapping

---

- Definition

*Ein **erweitertes Containment Mapping** (CM)  $h$  von Anfrage  $v$  nach Anfrage  $q$  ist ein Symbol Mapping von  $v$  nach  $q$  für das gilt:*

*1.  $\forall c \in \text{const}(v)$  gilt:  $h(c) = c$*

*2.  $\forall l \in v$  gilt:  $h(l) \in q$*

*3.  $\forall e \in \text{exp}(q)$  gilt:  $\exists e' \in \text{exp}(v)$  mit  $h(e') = e$*

*• Der Kopf von  $q$  ist im Bild des Kopfes von  $v$  enthalten*

*4.  $\text{cond}(q) \rightarrow \text{cond}(h(v))$*

- In Zukunft: CM = Erweitertes Containment Mapping

# Ableitbarkeit

---

- Jetzt hat man alle Attribute, aber zu viele Tupel
- Wie findet man die richtigen?
- **Ableitbarkeit**
  - Wenn  $q \subseteq v$ , dann gilt:  $q \rightarrow h(v)$ 
    - $h$ : Containment Mapping von  $v$  nach  $q$
    - $\rightarrow$  bezeichnet hier die **logische Implikation** zwischen Formeln in Prädikatenlogik
  - Gesucht: Ausdruck  $F$  für den gilt:  $q \equiv h(v) \wedge F$
  - Im Allgemeinen **unentscheidbar**
    - Wegen Unentscheidbarkeit der Prädikatenlogik
  - Aber wir betrachten **nur konjunktive Anfragen**

# Beispiel

$v_1(P, PN, SN) :- s(SID, PID, LID, P), p(PID, PN), l(LID, SN)$

$v_2(P, PN, SN) :- s(SID, PID, LID, P), p(PID, PN), l(LID, SN)$   
 $P > 100, P < 300, SN = ,Kreuzberg`$

$q_1(P, PN, SN) :- s(SID, PID, LID, P), p(PID, PN), l(LID, SN),$   
 $PN = ,Gerolsteiner`$

$q_2(P, PN, SN) :- s(SID, PID, LID, P), p(PID, PN), l(LID, SN),$   
 $P > 100, P < 200, SN = ,Kreuzberg`$

$q_3(P, PN, SN) :- s(SID, PID, LID, P), p(PID, PN), l(LID, SN),$   
 $SN = ,Kreuzberg`$

$v_1, PN = ,Gerolsteiner` \equiv q_1$

$v_1, \dots \equiv q_x$

$v_2, \dots \not\equiv q_1$

$v_2, P < 200 \equiv q_2$

$v_2, \dots \not\equiv q_3$

# Inhalt dieser Vorlesung

---

- Materialisierte Sichten
- Logische Optimierung mit MV
  - Einschub: Datalog Notation
  - Query Containment
  - Depth-First Algorithmus
  - Ableitbarkeit und Query Rewriting
    - [Ableitbarkeit von Bedingungen](#)
    - Ableitbarkeit von Joins
    - Ableitbarkeit von Aggregaten
- Kostenbasierte Optimierung mit MV
- Optimierung mit Aggregaten

# Ableitbarkeit von Bedingungen

---

- Annahmen
  - $q \subseteq v$  (mit erweitertem CM)
  - $q$  und  $v$  beinhalten dieselben Literale, Joins und Gruppierungen
- Damit können wir **cond(q) als Filter** verwenden
  - Per Definition CM gilt:  $\text{cond}(q) \rightarrow \text{cond}(h(v))$
  - $\text{cond}(q)$  sind also schärfere Bedingungen; mit denen müssen wir die Tupel aus  $v$  filtern
- Beispiel
  - $v(A,B) :- r(A,B,C,D), B < 40, C > 60$
  - $q(A,B) :- r(A,B,C,D), B < 30, C > 70$
- Ist **q aus v ableitbar?**
  - Verschärfung  $B < 30$  auf Ergebnis von  $v$  berechenbar
  - Verschärfung  $C > 70$  nicht auf Ergebnis von  $v$  berechenbar
- Wir müssen also auf **exportierte Variable** aufpassen

# Algorithmus

---

- Annahmen
  - $q \subseteq v$  mit CM  $h: v \rightarrow q$
  - Seien  $\text{cond}(q) = \{B_1, B_2, \dots, B_n\}$  (ohne Joins)
- Algorithmus
  - Für alle Bedingungen  $B_i$  mit  $h(v) \rightarrow B_i$
  - Wenn  $B_i$  Variablen enthält, deren Urbild bzgl.  $h$  in  $v$  nicht exportiert ist: Abbruch
    - Unzureichende Bedingung auf einer nicht-exportierten Variable
  - Sonst:  $q \equiv h(v) \wedge \text{cond}(q)$
- Komplexität des einzelnen Tests
  - $O(n)$  für Bedingungen der Art:  $X=5, X<5, X>5$
  - $O(n^3)$  für Bedingungen der Art:  $X=Y, X<Y, X>Y$

# Beispiele

```
v(P,PN,TID) :- s(SID,PID,LID,TID,P),p(PID,PN),l(LID,SN);
q1(P,PN,TID) :- s(SID,PID,LID,TID,P),p(PID,PN);
q2(P,PN,TID) :- s(SID,PID,LID,TID,P),p(PID,PN),l(LID,SN),
 t(TID,D,M,Y);
q3(P,TID) :- s(SID,PID,LID,TID,P),p(PID,P),l(LID,SN);
```

$q_1 \subseteq v$  ?

- **Nein**: Inner Join mit  $l(LID,SN)$  wirkt als Filter in  $v$

$q_2 \subseteq v$  ?

- **Ja**: Inner Join mit  $t()$  wirkt als Filter in  $q_2$
- Also:  $q_2 \equiv v, t(TID,D,M,Y)$

$q_3 \subseteq v$  ?

- **Ja**: Join  $s(...,P), p(PID,P)$  ist Filter in  $q_3$
- Also:  $q_3 \equiv v, PN=P$

# Ableitbarkeit von Joins

- Annahme
  - $q \subseteq v$  mit CM  $h: v \rightarrow q$
  - $q$  und  $v$  beinhalten dieselben Bedingungen und Gruppierungen
  - Aber **unterschiedliche Literale bzw. Joins**
- Potentielle Probleme
  - $q$  enthält Literale, die  $v$  nicht enthält
  - $q$  enthält Joins, die  $v$  nicht enthält
- Einfachste Idee: Wende  $q$  auf  $v$  an:  $q \equiv q(v(D))$ 
  - In typischen RDBMS **sehr schwierig zu implementieren**
- Wir wollen lieber die Anfrage umschreiben

$$\begin{array}{l} v(\dots) \quad :- \quad l(X, Y), k(Y, Z) \\ q(F_1, \dots) \quad :- \quad l(\mathbf{B}, \mathbf{A}), k(\mathbf{A}, \mathbf{B}), m(\mathbf{A}, \mathbf{C}) \end{array}$$

# Algorithmus für Joins

---

- Sei  $q \subseteq v$  mit CM  $h: v \rightarrow q$ 
  - $h$  bildet jedes Literal aus  $v$  auf **mindestens ein** Literal aus  $q$  ab
- Algorithmus
  - Berechne Literale  $L=(l_1, l_2, \dots)$  aus  $q$ , die nicht im Bild von  $h$  sind
  - Prüfe alle Variable  $V \in I_x$ , die als Bild in  $h$  enthalten sind
    - D.h. es gibt ein  $(X \rightarrow V) \in h$  ( $X$  Variable in  $v$ )
    - Wenn  $X \notin \text{exp}(v)$ : Abbruch
      - Da **nicht kompensierbarer Join**
  - Prüfe alle Elemente von  $J = \{ (X=Y) \mid (X \rightarrow V) \in h \wedge (Y \rightarrow V) \in h \}$ 
    - Das sind Joins in  $q$  aber nicht in  $v$
    - Wenn  $X \notin \text{exp}(v)$  oder  $Y \notin \text{exp}(v)$ : Abbruch
- Sonst:  **$q \equiv h(v) \wedge L \wedge J$**

# Beispiel

$$v(P, PN, SN, TID) :- s(SID, PID, LID, TID, P), p(PID, PN), l(LID, SN)$$
$$q_2(P, PN, SN, TID) :- s(SID, PID, LID, TID, P), p(PID, PN), l(LID, SN), \\ t(TID, D, M, Y)$$
$$q_3(P, P, SN, TID) :- s(SID, PID, LID, TID, P), p(PID, P), l(LID, SN)$$

- $q_2 \subseteq v$ 
  - $L = \{t\}, J = \{\}$
  - $(TID \rightarrow TID) \in h$ , aber  $TID \in \text{exp}(v)$
  - Also:  $q_2 \equiv v(P, PN, SN, TID), t(TID, D, M, Y)$
- $q_3 \subseteq v$ 
  - $L = \{\}, J = \{(PN = P)\}$
  - $(P \rightarrow P), (PN \rightarrow P) \in h$ , aber  $P, PN \in \text{exp}(v)$
  - Also:  $q_3 \equiv v(P, PN, SN, TID), P = PN$

# Inhalt dieser Vorlesung

---

- Materialisierte Sichten
- Logische Optimierung mit MV
  - Einschub: Datalog Notation
  - Query Containment
  - Depth-First Algorithmus
  - Ableitbarkeit und Query Rewriting
    - Ableitbarkeit von Bedingungen
    - Ableitbarkeit von Joins
    - **Ableitbarkeit von Aggregaten**
- Kostenbasierte Optimierung mit MV
  - Optimierung mit Aggregaten

# Ableitbarkeit von Aggregaten

---

- Annahmen

- q und v haben die Form

```
SELECT G1, G2, ..., Gn, sum(A1)
FROM ...
WHERE ...
GROUP BY G1, G2, ..., Gn
```

- Ohne Gruppierung / Aggregation soll gelten:  $q \equiv v$ 
  - BZW.  $q \equiv h(v) \wedge B \wedge L \wedge J$

- Wir betrachten nur die Aggregatsfunktion SUM

- Bei anderen Funktionen Anwendbarkeit in Hierarchien beachten (distributiv, algebraisch, holistisch ...)

- $G_1, G_2, \dots, G_n$  heißen **Gruppierungsattribute**

- Nur hier sind Unterschiede zwischen q und v

- Frage: Welche q kann man unter Verwendung eines gegebenen v (schneller) beantworten ?

# Beispiel

- Sei  $v$

```
SELECT pg_id, shop_id, year_id, sum(amount*price)
FROM sales S, ...
GROUP BY S.pg_id, S.shop_id, T.year_id
```

- Welche Gruppierungen in einer Query  $q$  können mit  $v$  berechnet werden?

| PG      | Shop           | Year | SUM |
|---------|----------------|------|-----|
| Pepsi   | Kreuzberg      | 1997 | ... |
| Pepsi   | Charlottenburg | 1997 | ... |
| Pepsi   | Kreuzberg      | 1998 | ... |
| Pepsi   | Charlottenburg | 1998 | ... |
| Bionade | Kreuzberg      | 1997 | ... |
| Bionade | Charlottenburg | 1997 | ... |
| Bionade | Kreuzberg      | 1998 | ... |
| Bionade | Charlottenburg | 1998 | ... |
| ...     | ...            | ...  | ... |

# Beobachtung

- In  $v$  sind alle Kombinationen von  $G = \{pg\_id, shop\_id, year\_id\}$  mit Summe vorhanden
- Aufsummierung für **jede Untermenge** von  $G$  möglich

```
SELECT T.pg_id, T.year_id, sum(...)
FROM v
GROUP BY T.pg_id, T.year_id
```

```
SELECT T.shop_id, sum(...)
FROM v
GROUP BY T.shop_id
```

| PG      | Year | SUM |
|---------|------|-----|
| Pepsi   | 1997 | ... |
| Pepsi   | 1998 | ... |
| Bionade | 1997 | ... |
| Bionade | 1998 | ... |
| ...     | ...  | ... |

| Shop           | SUM |
|----------------|-----|
| Kreuzberg      | ... |
| Charlottenburg | ... |
| ...            | ... |

# Formaler

---

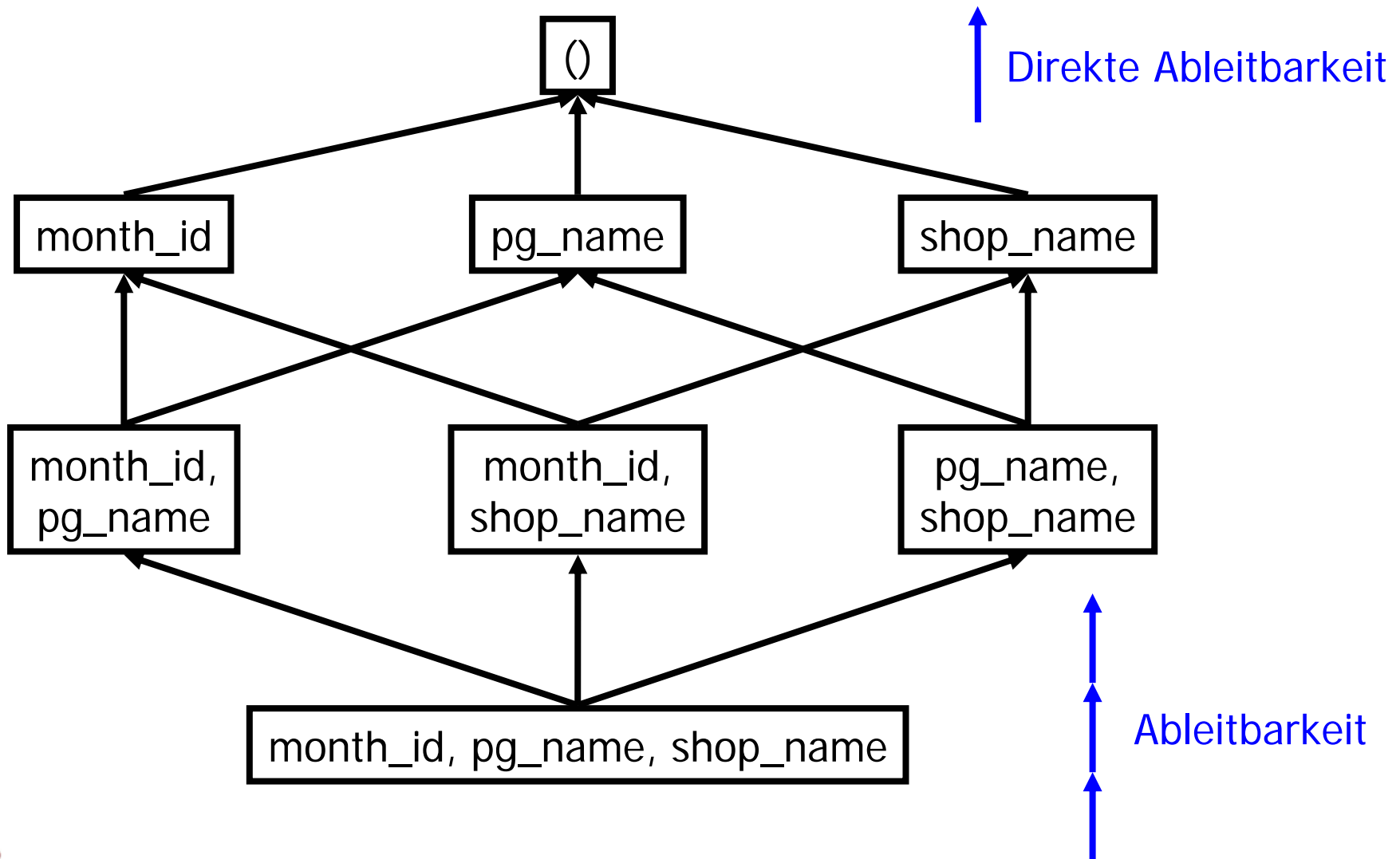
- Erinnerung

*Eine Gruppierung  $G$  ist aus einer Gruppierung  $H$  **ableitbar**, wenn  $H \subseteq G$*

- Lemma

- *Sei  $v$  ein View mit Gruppierungsattributen  $G_v$  und  $q$  eine Query mit Gruppierungsattributen  $G_q$  und  $v$  haben äquivalente SPJ Klauseln*
  - *SPJ: „Select – Project – Join“*
- *$q$  ist **direkt ableitbar** aus  $v$ ,  $v \rightarrow q$ , gdw.*
  - *$G_q \subset G_v$  und  $|G_q| = |G_v| - 1$  oder*
  - *$G_q = G_v \setminus a_x \cup a_y$  mit  $a_x \in G_v$ ,  $a_y \notin G_v$  und  $a_y$  ist funktional abhängig von  $a_x$*
- *$q$  ist **ableitbar** aus  $v$ ,  $v \rightarrow^* q$ , gdw.*
  - *$v \rightarrow q$  oder*
  - *Es existieren Views  $v_1, \dots, v_n$  so, dass:  $v \rightarrow v_1 \rightarrow \dots \rightarrow v_n \rightarrow q$*

# Aggregationsgitter



# Ableitbarkeit bei funkt. Abhängigkeiten

---

```
SELECT T.month_id, P.pg_id, R.shop_id, sum(v.sum)
FROM sales S, time T, product P, region R
WHERE ...
GROUP BY T.month_id, P.pg_id, R.shop_id
```



```
SELECT T.year_id, v.pg_id, v.shop_id, sum(v.sum)
FROM v, time T
WHERE v.month_id = T.month_id
GROUP BY T.year_id, v.pg_id, v.shop_id
```

- Erfordert Rewriting mit zusätzlichen Joins
- Erfordert **weitere Aggregationen**
  - Entlang der Hierarchie

# Zusammenfassung Ableitbarkeit

---

- Im allgemeinen Fall unentscheidbar
- NP-vollständig schon für sehr eingeschränkte Anfrageklassen (konjunktive Anfragen)
- Aber: **Linear für viele Anfragen des täglichen Lebens**
  - Wann braucht man Self-Joins?
- Gruppierungsableitbarkeit (ohne `having`) ist linear
- Vorsicht vor holistischen Aggregatfunktionen (z.B.: `median`)

# Inhalt dieser Vorlesung

---

- Materialisierte Sichten
- Logische Optimierung mit MV
- **Kostenbasierte Optimierung mit MV**
  - Grundprinzip
  - Optimierung mit Aggregaten

# Ziel

---

- Mögliche nächste Frage
  - Gegeben eine Query  $q$  und **mehrere materialisierte Views**, die zur Ableitung von  $q$  dienen können
  - Welcher View soll verwendet werden – wenn überhaupt?
    - Der kleinste
    - Der, bei dem die Kompensationsattribute billig auszuwerten sind
    - ...
- Diese Frage ist **sehr eingeschränkt**
  - Wir verlangen, dass  $q$  vollständig ist  $v$  enthalten ist
- Allgemeine Frage
  - Gibt es einen **Teil von  $q$** , den man mit einem  $v$  berechnen kann?
  - Welche Teile? Kostenbasierte Optimierung?

# Anfrageoptimierung mit MVs [TCL+00]

---

- Kostenbasierte Anfrageoptimierung
  - Anfrage parsen
  - Aufzählen von Query Execution Plans (QEP)
    - Joinreihenfolge
    - Joinmethode
    - Operatorreihenfolge
    - etc.
  - Bottom-Up Bewertung von Teilplänen (Access Paths)
  - **Konstruktion vollständiger Pläne aus besten Teilplänen**
  - Dynamische Programmierung

# Anfrageoptimierung mit MVs

---

- Wo passen hier materialisierte Sichten?
- Sketch
  - Jede materialisierte Sicht ist ein potentieller Teilplan
    - Besser: Jede MV plus Kompensationen
  - Bei Bottom-Up Bewertung von Teilplänen auch materialisierte Sichten berücksichtigen
    - **Matching**: Gibt es für den aktuellen Teilplan einen / mehrere geeignete MVs?
    - Hinzufügen von Kompensationsoperationen notwendig?
  - Bewertung der Kosten (MV + Kompensation)
  - Auswahl des MV als Access Path, wenn er geringere Kosten als andere Teilpläne in Aussicht stellt

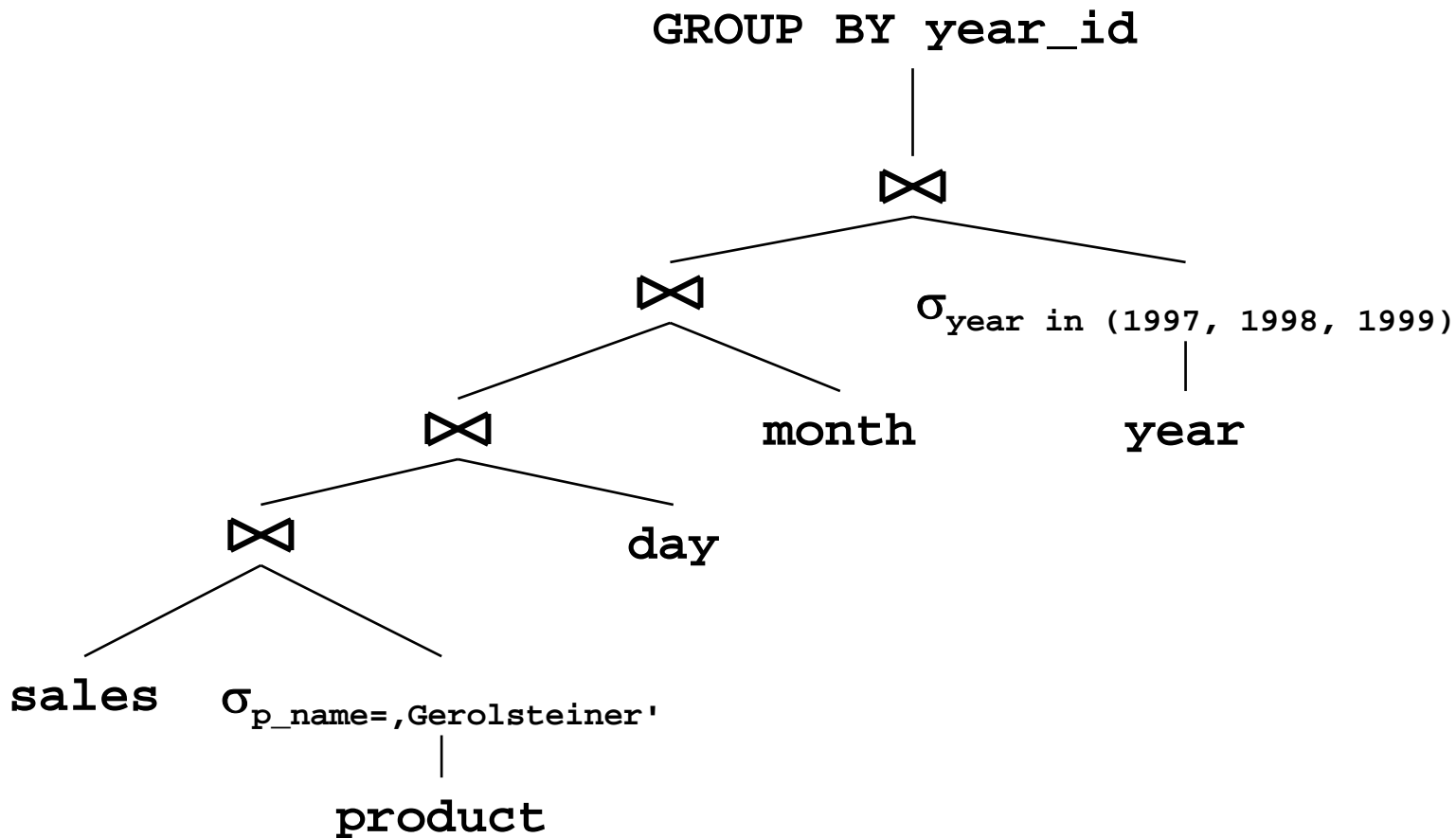
# Beispiel (Snowflakeschema)

---

```
SELECT Y.year, sum(amount)
FROM sales S, product P, days D, months M, years Y
WHERE P.name=„Gerolsteiner“ AND
 P.product_id = S.product_id AND
 S.day_id = D.day_id AND
 D.month_id = M.id AND
 M.year_id = Y.id AND
 Y.year in (1997, 1998, 1999)
GROUP BY Y.year
```

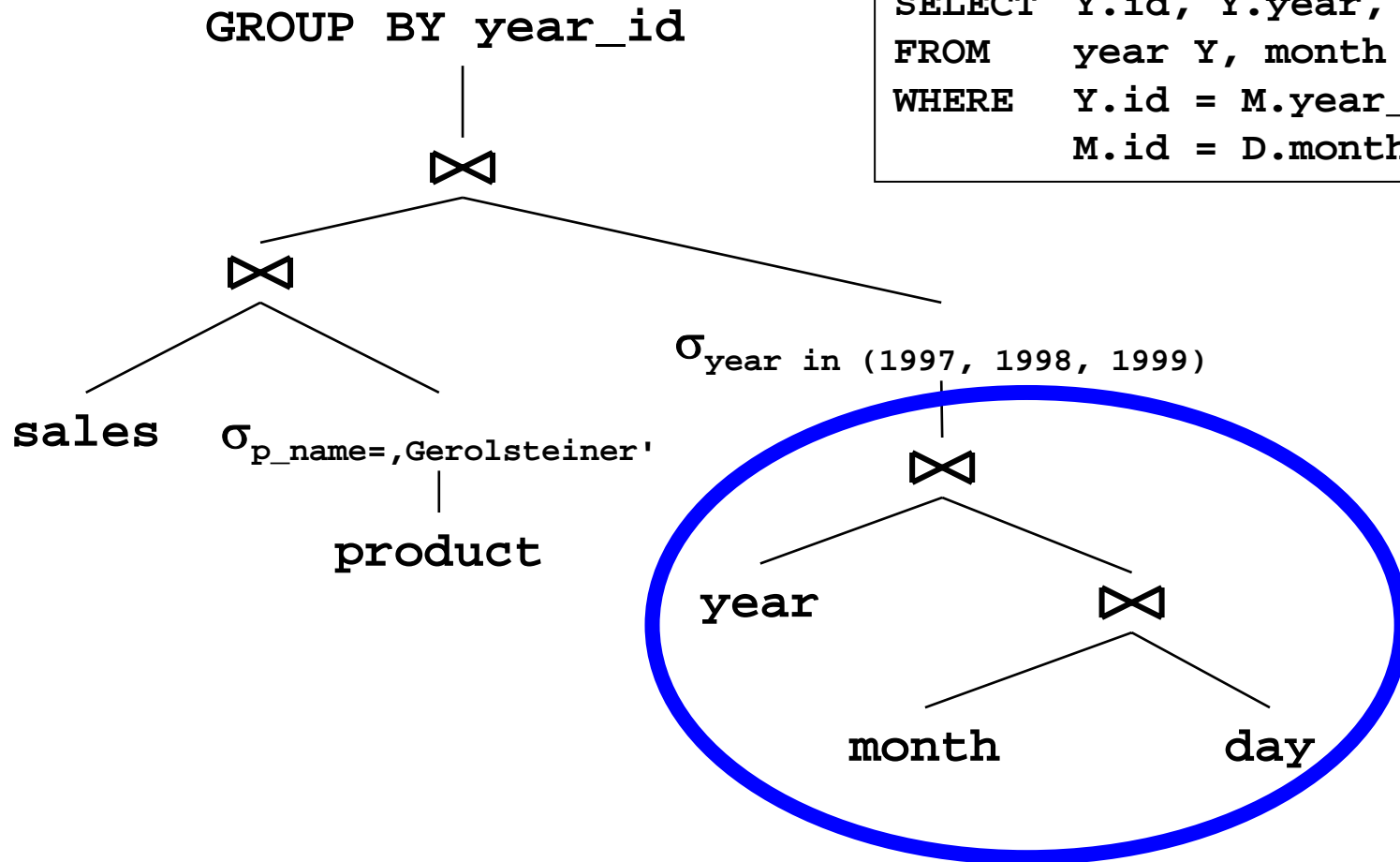
```
CREATE MATERIALIZED VIEW v_time AS
SELECT Y.id, Y.year, M.id, M.month, D.id, D.day
FROM year Y, month M, day D
WHERE Y.id = M.year_id AND
 M.id = D.month_id
```

# Ausführungsplan



```
CREATE MATERIALIZED VIEW v_time AS
SELECT Y.id, Y.year, M.id, M.month, D.id, D.day
FROM year Y, month M, day D
WHERE Y.id = M.year_id AND
 M.id = D.month_id
```

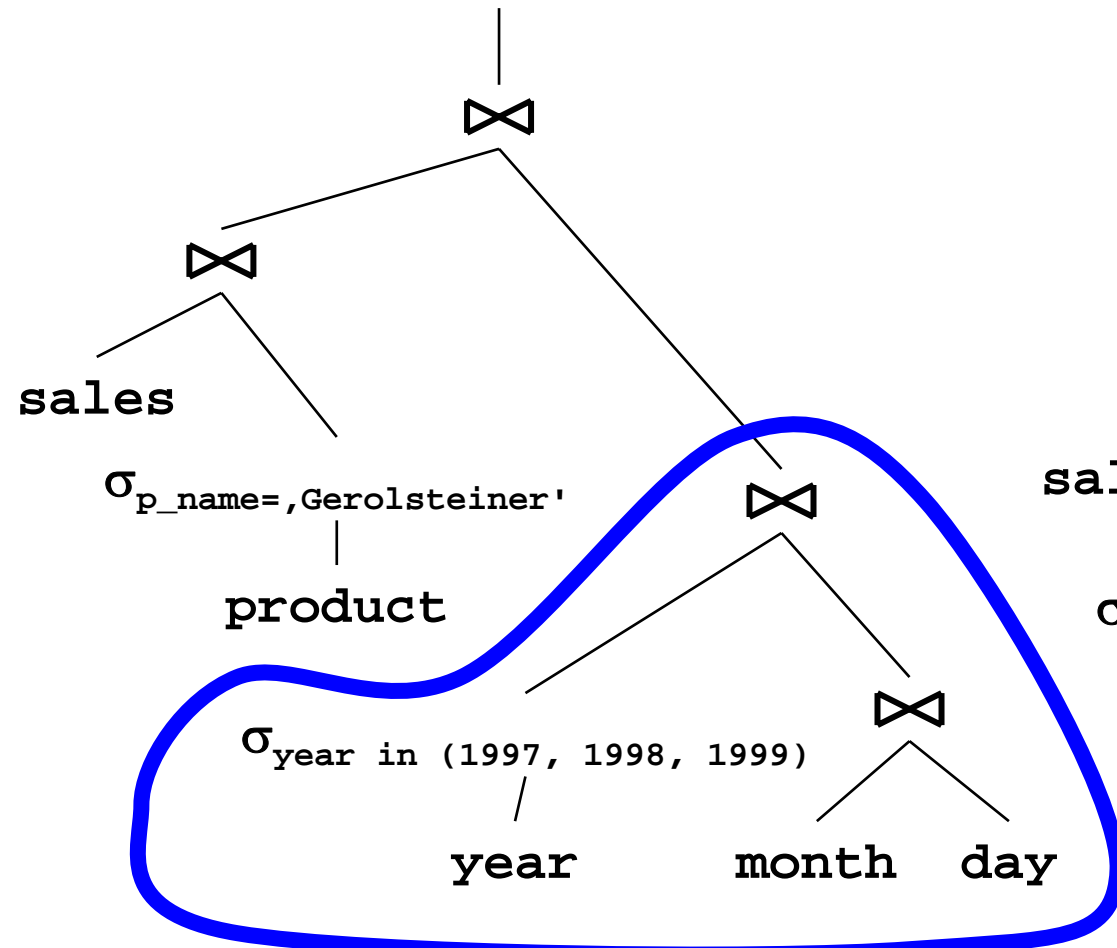
# Alternativplan



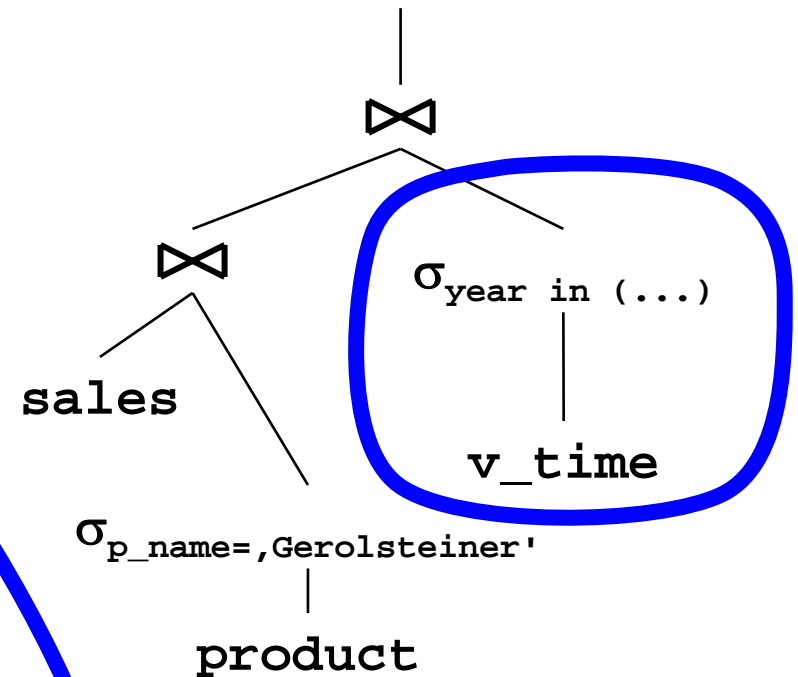
```
CREATE MV v_time AS
SELECT Y.id, Y.year, ...
FROM year Y, month M, day D
WHERE Y.id = M.year_id AND
 M.id = D.month_id
```

# Alternativen bewerten

GROUP BY year\_id



GROUP BY year\_id



# Einschränkungen

---

- „**Matching**“ soll sehr schnell gehen
  - Lieber einen MV übersehen, als zu lange für Auswahl brauchen
  - Exponentielle Algorithmen vermeiden
  - Trick: MVs indexieren (insb. nach enthaltenen Relationen)
- I.d.R. werden vom Optimierer nicht alle Pläne aufgezählt
  - Typischerweise nur Left-Deep Joins
  - **Gezieltes Suchen** nach MVs muss eingebaut werden
- Matching erkennt in echten Systemen **nicht alle Matches**
  - Abhängig von Datenbanksystem
  - Beispiele für Einschränkungen
    - Keine Funktionen in Bedingungen
    - Keine Negation
    - Keine Self-joins
    - ...

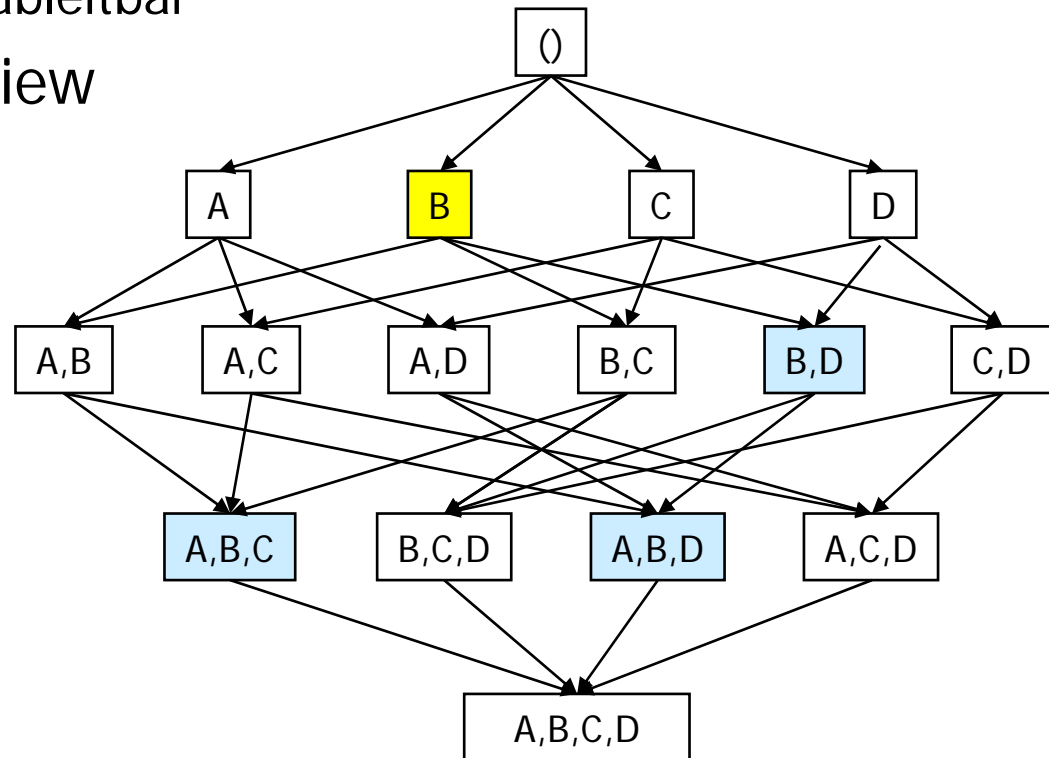
# Einschränkungen – Beispiele [GL01]

---

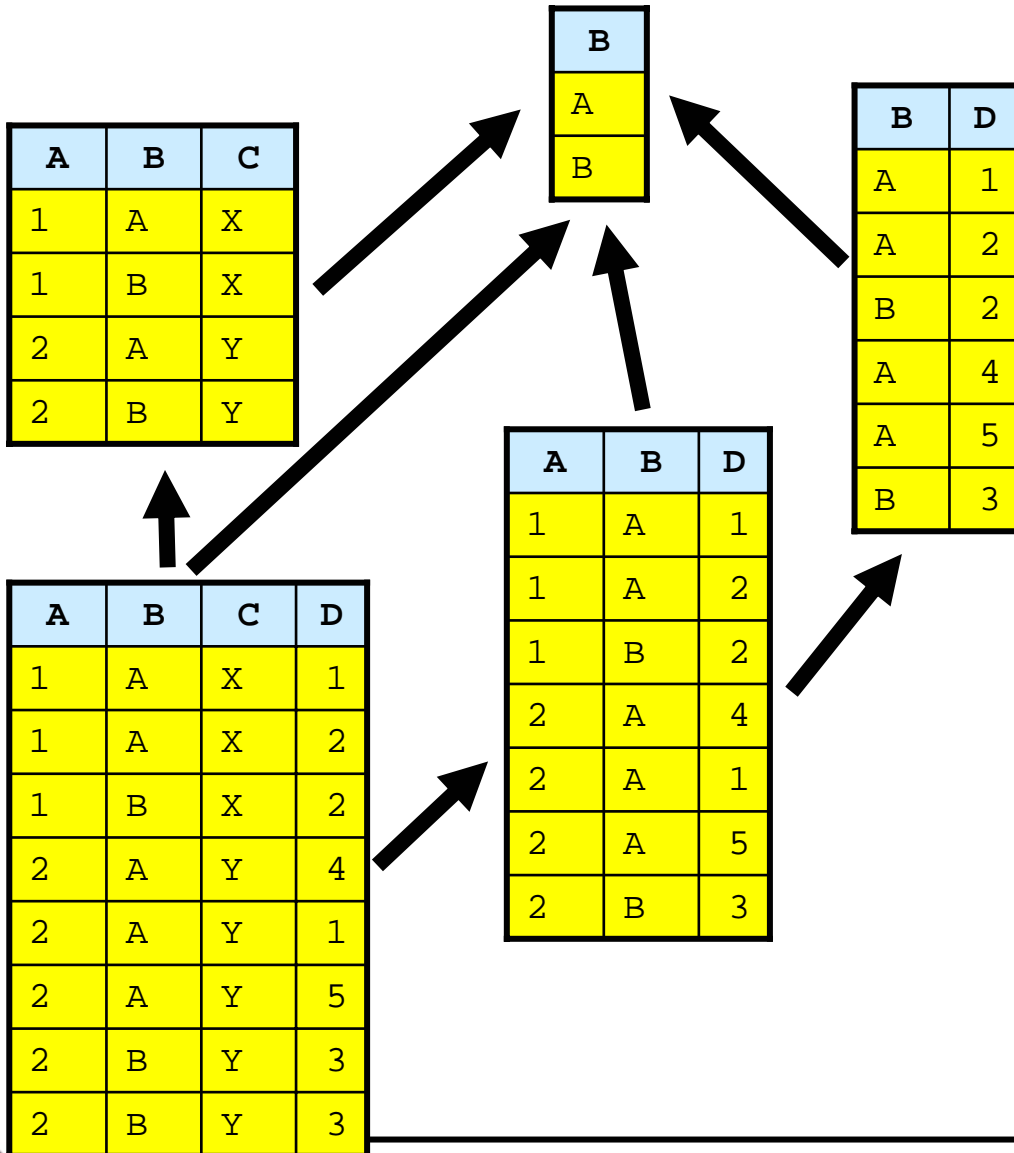
- Möglichkeiten, die SQLServer verpasst
  - Äquivalente Formelausdrücke
    - Q: `WHERE .... A+B=10`
    - V: `WHERE .... (B/2+A/2)*10=50`
  - Induzierbare Äquivalenzen
    - Q: `WHERE .... A=2 and B=2`
    - V: `WHERE .... A=B`
  - Ersetzung konstanter/berechenbarer Output-Columns
    - Q: `SELECT A,B ... WHERE... B=3`
    - V: `SELECT A,3 ... WHERE...`
    - Q: `SELECT... WHERE A*B>50`
    - V: `SELECT A*B ... WHERE`
  - ...

# Optimierung mit Aggregaten

- Annahme
  - Geg. eine Anfrage  $q$  mit Gruppierung
  - Geg. eine Menge materialisierter Views  $V = \{v_1, \dots, v_n\}$
  - $q$  sei aus allen  $v_i$  ableitbar
- Frage: Welchen View benutzt man am besten?



# Den kleinsten



- Berechnung von  $q$  aus MV benötigt:
  - Sortieren des Views nach Gruppierungsattributen von  $q$
  - Lesen, gruppieren und aggregieren in einem Scan
- Da man MV nicht sortiert speichern kann ...
- **Auswahl des MV mit der kleinsten Kardinalität**
- Das ist nicht unbedingt der View mit der kleinsten Menge an Gruppierungsattributen!

# Literatur

---

- [Leh03], Kapitel 7.2, 8.2
- [TCL+00] Zaharioudakis, M., et al. (2000). "Answering Complex SQL Queries Using Automatic Summary Tables". ACM SIGMOD
- [BDD+98] Bello, R. G., et al. (1998). "Materialized Views in ORACLE". 24th VLDB
- [GL01] Goldstein, J. and Larson, P.-A. (2001). "Optimizing Queries Using Materialized Views: A Practical, Scalable Solution". ACM SIGMOD, Seattle
- [LN06] Leser, U. and Naumann, F. (2006). "Informationsintegration". Heidelberg, dpunkt.verlag.