Data Warehousing

Answering Queries using Views

Ulf Leser

Wissensmanagement in der Bioinformatik

Inhalt dieser Vorlesung

- Materialisierte Sichten
- Answering Queries Using Views
- Query Containment
- Ableitbarkeit
- Query Optimierung mit MVs
- MVs in Oracle

Materialisierte Sichten (MV)

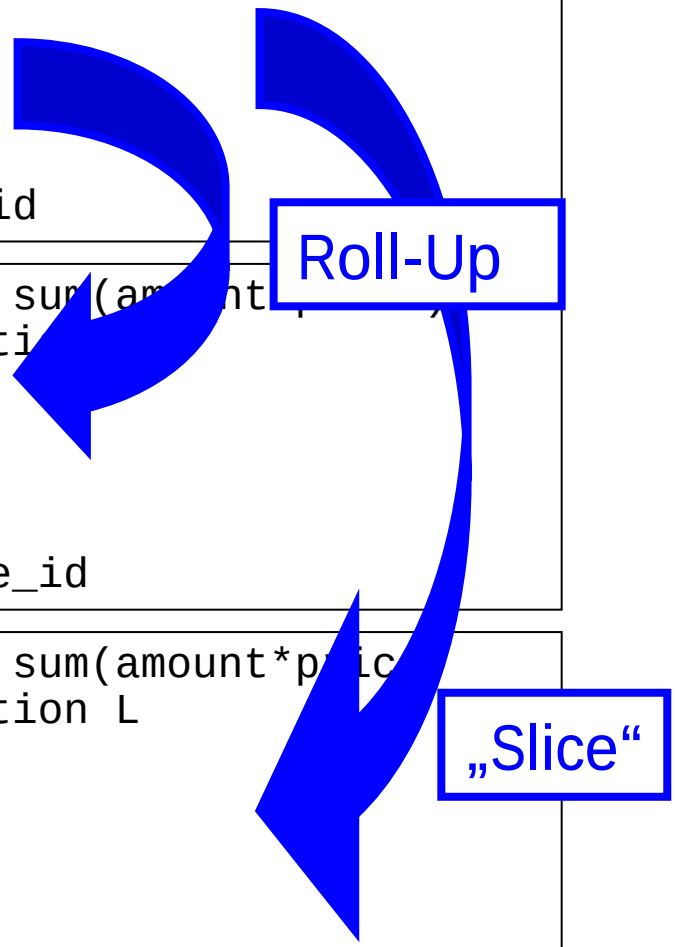
- Grundidee
  - Viele Anfragen wiederholen sich häufig
    - Bilanzen, Bestellungsplanung, Produktionsplanung
    - Kennzahlenberechnung
  - Viele Anfragen sind Variationen derselben Anfrage
    - Alle Verkäufe nach Produkt, Monat und Region
    - Alle Verkäufe in Region X nach Produkt, Monat und Shop
    - Alle Verkäufe in Region Y nach Produkt, Monat und Shop
- ➢ Materialisierter View
  - Einmaliges Berechnen und Speichern einer Anfrage
  - Transparente Verwendung in folgenden Anfragen

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 AND
        P.pg_id = 159
GROUP BY L.shop_id, P.product_id, S.time_id
```

```
SELECT L.shop_id, P.product_id, T.day_id, sum(am
FROM sales S, time T, product P, localizati
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
GROUP BY L.region_id, P.product_id, S.time_id
```

```
SELECT L.shop_id, P.product_id, T.day_id, sum(amount*pric
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
```



Materialisierter View

```
CREATE MATERIALIZED VIEW all_groups
AS
SELECT L.shop_id, P.product_id, T.day_id, T.year, P.pg_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 * FROM all_groups A
WHERE  A.pg_id = 159
```

OK

```
SELECT * FROM all_groups A
WHERE  A.pg_id = 159
Group by A.region_id
```

Nein

```
SELECT * FROM all_groups A
WHERE  P.pg_id = 159 AND
       // S.shop_id(+) = L.shop_id AND
       T.year = ‚1999‘
```

Nein

Materialisierte Sichten – Themen

- Wahl von Views zur Materialisierung
  - MVs kosten: Platz und Aktualisierungsaufwand
  - Wahl der optimalen MVs hängt von Workload ab
  - ➢ Auswahl der „optimale“ Menge von MV
- Automatische Aktualisierung von MV
  - Aktualisierung bei Änderungen
  - U.U. schwierig: Aggregate, Joins, Outer-Joins, ...
  - ➢ Algorithmen zur inkrementellen Aktualisierung
- Automatische Verwendung von MV
  - „Answering Queries using Views“
  - Umschreiben der Anfrage notwendig
  - Schwierigkeit hängt von Komplexität der Anfrage / Views ab
  - ➢ Algorithmen zur transparenten und kostenoptimalen Verwendung der materialisierten Sichten

Answering Queries using Views

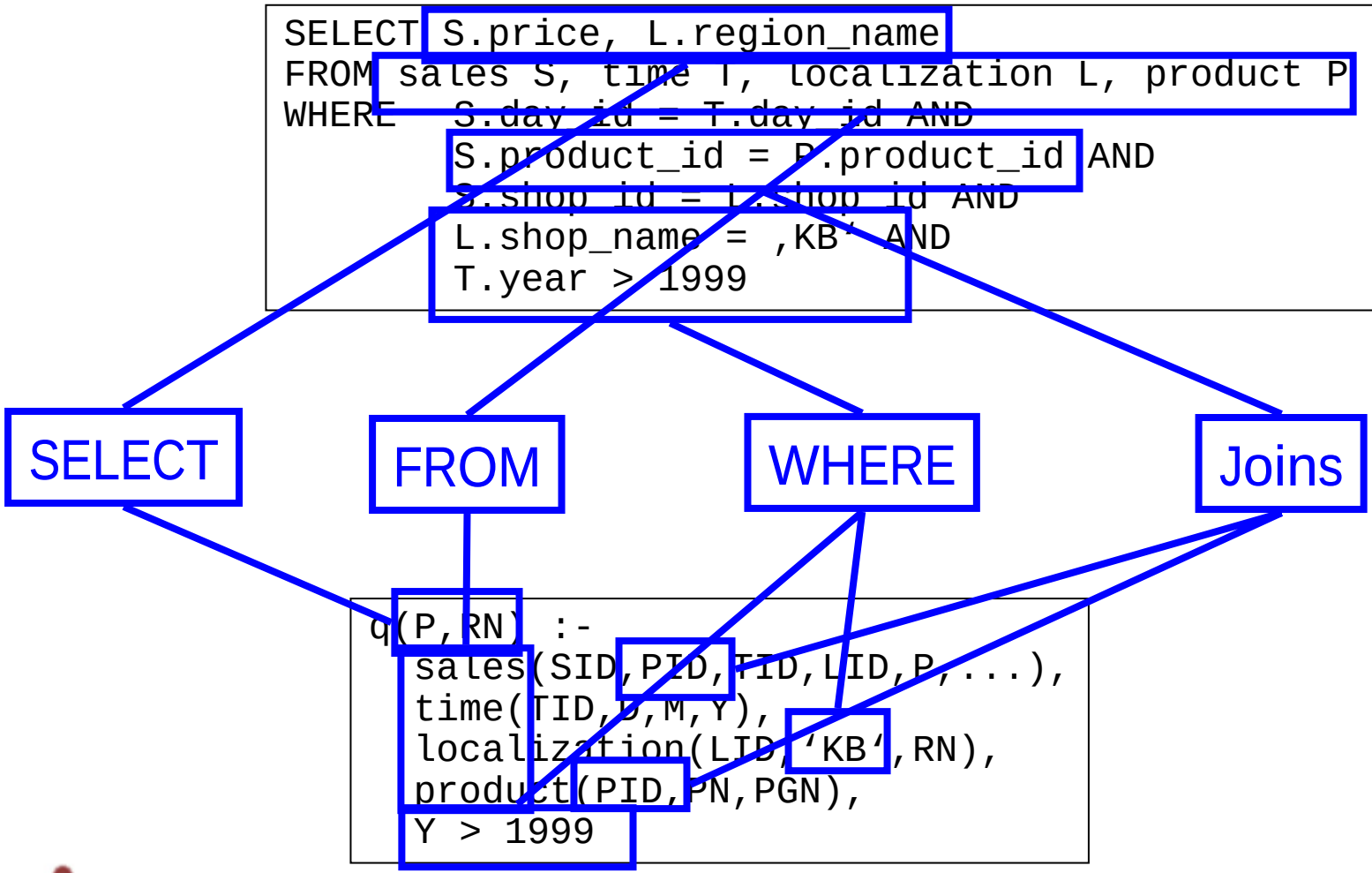
- Gegeben
  - Eine Query q
  - Eine Menge V von materialisierten Views
- Fragen
  - Kann man Antworten für q unter Verwendung von V berechnen?
  - Kann man q nur mit Views aus V beantworten?
  - Kann man q unter Verwendung von V korrekt beantworten?
  - Ist es günstig, Views aus V zur Beantwortung von q zu verwenden? Welche?
  - ...

➢ Viele und schwierige Probleme

Datalog Notation

- Schreibweise: Datalog
  - `q(X,Y):-`
    `sales(X,A,B,C),time(A,Y,D),D>1999`
  - SELECT Klausel
    - Regelkopf, „Exported Variables“
  - FROM Klausel
    - Relationen werden zu Prädikaten
    - Attribute werden über Position statt Name adressiert
  - WHERE Klausel
    - Joins werden durch gleiche Variablen in verschiedenen Prädikaten angezeigt
    - 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
SELECT
FROM
WHERE
Joins
q(P,RN) :-
sales(SID,PID,TID,LID,P,...),
time(TID,D,M,Y),
localization(LID,‘KB‘,RN),
product(PID,PN,PGN),
Y > 1999

Begriffe

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

- q ist der Kopf (oft synonym für ganze Query)
- Variablen sind in GROSSBUCHSTABEN
- `sales, time`, .. sind Prädikate
  - Relationen des Schemas
- `sales(SID,...), time(TID,...)` sind Literale
  - Eine Query kann ein Prädikat mehrmals enthalten - mehrere Literale desselben Prädikats
  - Literale sind eindeutig

Query Containment

- Gegeben
  - Query q
  - View v (zunächst nur einer)
- Anfrageäquivalenz
  - Ist Ergebnis von v identisch dem Ergebnis von q?
  - Kurz: Ist v äquivalent zu q, $v \Leftrightarrow q$
- Anfragecontainment („enthalten in“)
  - Ist das Ergebnis von v im Ergebnis von q enthalten?
  - Kurz: Ist v in q enthalten, $v \subseteq q$
- Rückführung
  - $v \subseteq q,\ q \subseteq v \Rightarrow v \Leftrightarrow q$
- Frage: Definition von „äquivalent“, „enthalten in“ für Queries?

Definition

- Definition
  - *Sei S ein Schema. Seien q_1 und q_2 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 (zur genauen Semantik siehe Domänen / Tupelkalkül)*
  - *q_1 ist äquivalent zu q_2, geschrieben $q_1 \Leftrightarrow q_2$, gdw. $q_1(D^S) = q_2(D^S)$ für jede mögliche Instanz D^S von S*
  - *q_1 ist enthalten in q_2, geschrieben $q_1 \subseteq q_2$, gdw. $q_1(D^S) \subseteq q_2(D^S)$ für jede mögliche Instanz D^S von S*
- Semantische Definition: Es zählt das Ergebnis einer Query, nicht die Syntax

Beispiele

Regelkopf wird hier unterschlagen

q	$\Leftrightarrow$	q
product(PID,PN,PGID, ,Wasser‘)	$\subseteq$	product(PID,PN,PGID,PGN)
product(PID,PN,PGID,PGN)	$\not\subseteq$	localization(SID,SN,RID,RN)
product(PID,PN,PGID,PGN), PGN>,Wasser‘	$\subseteq$	product(PID,PN,PGID,PGN)
sales(SID,PID,...,P,...), P>80, P<150	$\not\subseteq$	sales(SID,PID,...,P,...), P>100, P<150
sales(SID,PID,...,P,...), P>100,P<=150,P<170,P>=150	$\Leftrightarrow$	sales(SID,PID,...,150,...)
sales(SID,PID,...,P,...), product(PID,PN,...)	$\subseteq$	sales(SID,PID,...,P,...) *(Bei Projektion auf sales)*

Weitere Beispiele

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

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

q_1(A,C) :- path(A,B), path(B,C)
q_2(A,C) :- threeNodePath(A,B,C)
$q_1 \subseteq q_2$

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

?

Wie kann man Containment beweisen?

- *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:
  - *$\forall s \in constants(q_2)$ gilt: $h(s) = s$*
    - *Jede Konstante in q_2 wird auf dieselbe Konstante in q_1 abgebildet*
  - *$\forall l \in q_2$ gilt: $\exists\, l' \in q_1$ mit $h(l) = l'$*
    - *Jedes Literal in q_2 wird auf (mindestens) ein Literal in q_1 abgebildet*
  - *$h(\, head(q_2)) = head(q_1)$*
    - *Der Kopf von q_2 wird auf den Kopf von q_1 abgebildet*
  - *Die Bedingungen von q_1 implizieren die Bedingungen von $h(q_2)$*
- Bemerkung
  - Jede exportierte Variable in q_2 wird als auf eine exportierte Variable in q_1 abgebildet, und zwar die erste auf die erste, die zweite auf die zweite, etc.

Vom Containment Mapping 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
- Richtungen beachten!
  - Containment Mapping von q_2 nach q_1
  - Bedingungen von q_1 implizieren Bedingungen von q_2

Aufgabe

- Gegeben
  - Query q
  - View(s) v
- Gesucht
  - Eine Methode, q mit Hilfe von v zu beantworten
- Ist das Query Containment?
  - Nur fast – v muss eventuell modifiziert werden
- Trick
  - Was nicht passt, wird passend gemacht

Anwendung

- Möglichkeit 1: q nicht in v, v nicht in q
  - Kein (erkennbarer) Zusammenhang zwischen Queries
  - v wird nicht für q benutzt

- Möglichkeit 2: v und q äquivalent
  - v als Ergebnis von q ausgeben
  - Seltener Spezialfall
  - Test auf Äquivalenz sehr teuer (2 x Containment)
  - Benutzung nur bei „syntaktischer Äquivalenz“

Anwendung

- Möglichkeit 3: v in q aber nicht umgekehrt
  - v ist ein partielles Ergebnis für q
  - Jedes Tupel von v ist korrekte Antwort für q, aber v berechnet nicht alle korrekten Antworten
  - Erweiterung von v i.d.R. nicht hilfreich
    - Berechnung von q\v nicht trivial
    - Nur selten einfache (und damit schnelle) Lösungen
  - Keine Verwendung (in Datenbanken), aber
    - Informationsintegration
    - Datentransformationen

Anwendung 2

- Möglichkeit 4: q in 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 Filter F auf v, so dass nur die richtigen Tupel selektiert werden, also $F(v) \Leftrightarrow q$?

2. Ableitbarkeit

- Jetzt hat man alle Attribute, aber zu viele Tupel
- Wie findet man die richtigen?

➢ Ableitbarkeit

  - Wenn $q \subseteq v$, dann gilt: $\forall t: q(t) \Rightarrow v(t)$
  - Gesucht: Formel F: $\forall t: q(t) \Leftrightarrow v(t) \wedge F(t)$
  - Der allgemeine Fall ist unentscheidbar wegen Unentscheidbarkeit der Prädikatenlogik
  - Deswegen viele Einschränkungen auf den zugelassenen Operationen in q und v
    - Konjunktive Anfragen

Beispiel

Annahme: q und v enthalten dieselben Literale und Joins

```
v1(P,PN,SN) :- s(SID,PID,LID,P),p(PID,PN),l(LID,SN)
v2(P,PN,SN) :- s(SID,PID,LID,P),p(PID,PN),l(LID,SN)
               P>100, P<300, SN=‚Kreuzberg‘

q1(P,PN,SN) :- s(SID,PID,LID,P),p(PID,PN),l(LID,SN),
               PN=‚Gerolsteiner‘
q2(P,PN,SN) :- s(SID,PID,LID,P),p(PID,PN),l(LID,SN),
               P>100, P<200, SN=‚Kreuzberg‘
q3(P,PN,SN) :- s(SID,PID,LID,P),p(PID,PN),l(LID,SN),
               SN=‚Kreuzberg‘
```

$v_1 \wedge$ PN=‚Gerolsteiner‘ $\Leftrightarrow q_1$

$v_1 \wedge ... \Leftrightarrow q_x$

$v_2 \wedge ... \not\Leftrightarrow q_1$

$v_2 \wedge$ P<200 $\Leftrightarrow q_2$

$v_2 \wedge ... \not\Leftrightarrow q_3$

2.3 Ableitbarkeit von Gruppierungen

- 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: $q \Leftrightarrow v$
    - Bzw. $q \Leftrightarrow h(v) \wedge B \wedge J$
- Wir betrachten nur die Aggregatsfunktion SUM
  - Bei anderen Funktionen Summierbarkeit über Hierarchien beachten
- $G_1, G_2, \ldots, G_n$ sind die Gruppierungsattribute
  - Nur hier seien Unterschiede zwischen q und v

Intuition

- Sei v

```
SELECT  T.year_id, T.month_id,
        T.day_id, sum(...)
FROM    Sales S, Time T
WHERE   ...
GROUP BY T.year_id, T.month_id,
          T.day_id
```

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

Year	Month	Day	SUM
1997	1	1	150
1997	1	2	130
1997	1	3	145
1997	1	4	122
...	...	...	...
1997	1	31	145
1997	2	1	133
1997	2	2	122
...	...	...	...
1997	3	10	180
1997	12	31	480
1998	1	1	240
...	...	...	...
2003	6	18	345

Beobachtung

- In v sind alle Kombinationen von G = {YEAR_ID, MONTH_ID, DAY_ID} mit Summe vorhanden
- Aufsummierung für jede Untermenge von G möglich

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

Year	Month	SUM
1997	1	15000
1997	2	13300
...	...	...
1997	12	48000
1998	1	24000
...	...	...
2003	6	34500

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

Year	Day	SUM
1997	1	1500
1997	2	1330
...	...	...
1997	31	4800
1998	1	2400
...	...	...
2003	30	3450

Orthogonale Gruppierungen

- Sei v

```
SELECT  T.month_id, P.pg_id,
        R.shop_name, sum(...)
FROM    Sales S, Time T, Product P,
        Region R
WHERE   ...
GROUP BY T.month_id, P.pg_id,
         R.shop_name
```

Month	PG_ID	Shop_ name	SUM
1	22245	KB	150
1	22245	Mitte	130
...	...	...	145
1	22246	KB	133
1	22246	Mitte	144
...	...	...	122
2	22245	KB	122
2	22245	Mitte	...
...	...	...	145

```
SELECT  P.pg_id, R.shop_name,
        sum(...)
FROM    v
GROUP BY P.pg_id,
         R.shop_name
```

```
SELECT  P.month_id, sum(...)
FROM    v
GROUP BY P.month_id
```

Direkte Ableitbarkeit

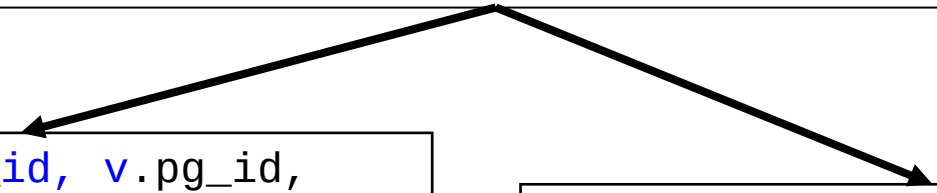
- Theorem
  - *Sei v ein View mit Gruppierungsattributen G_v und q eine Query mit Gruppierungsattributen G_q und q und v haben äquivalente SPJ Klauseln*
    - *SPJ: „Select – Project – Join“*
  - *q ist direkt ableitbar aus v, $q \rangle_1 v$, gdw.*
    - *$G_q \subset G_v$ und $|G_q| = |G_v| - 1$ oder*
    - *$G_q = G_v \mid a_x \cup a_y$ mit $a_x \in G_v$, $a_y \notin G_v$ und a_y ist funktional abhängig von a_x*
- Funktionale Abhängigkeit
  - Foreign Key Constraints
  - Zusicherung wie „CREATE DIMENSION ... ATTRIBUTE DETERMINES ...“

Ableitbarkeit

- Lemma
  - *q ist ableitbar aus v, q ⟩ v, gdw.*
    - *q ist direkt ableitbar aus v oder*
    - *Es existieren Views $v_1 \ldots, v_n$ so, dass: $q \rangle v_n \rangle \ldots \rangle v_1 \rangle v$*
- Also
  - Jede Untermenge einer Menge von Gruppierungsattributen ist aus dieser ableitbar

Ableitbarkeit bei funkt. Abhängigkeiten

```
SELECT  T.month_id, P.pg_id, R.shop_name, sum(...)
FROM    Sales S, Time T, Product P, Region R
WHERE   ...
GROUP BY T.month_id, P.pg_id, R.shop_name
```

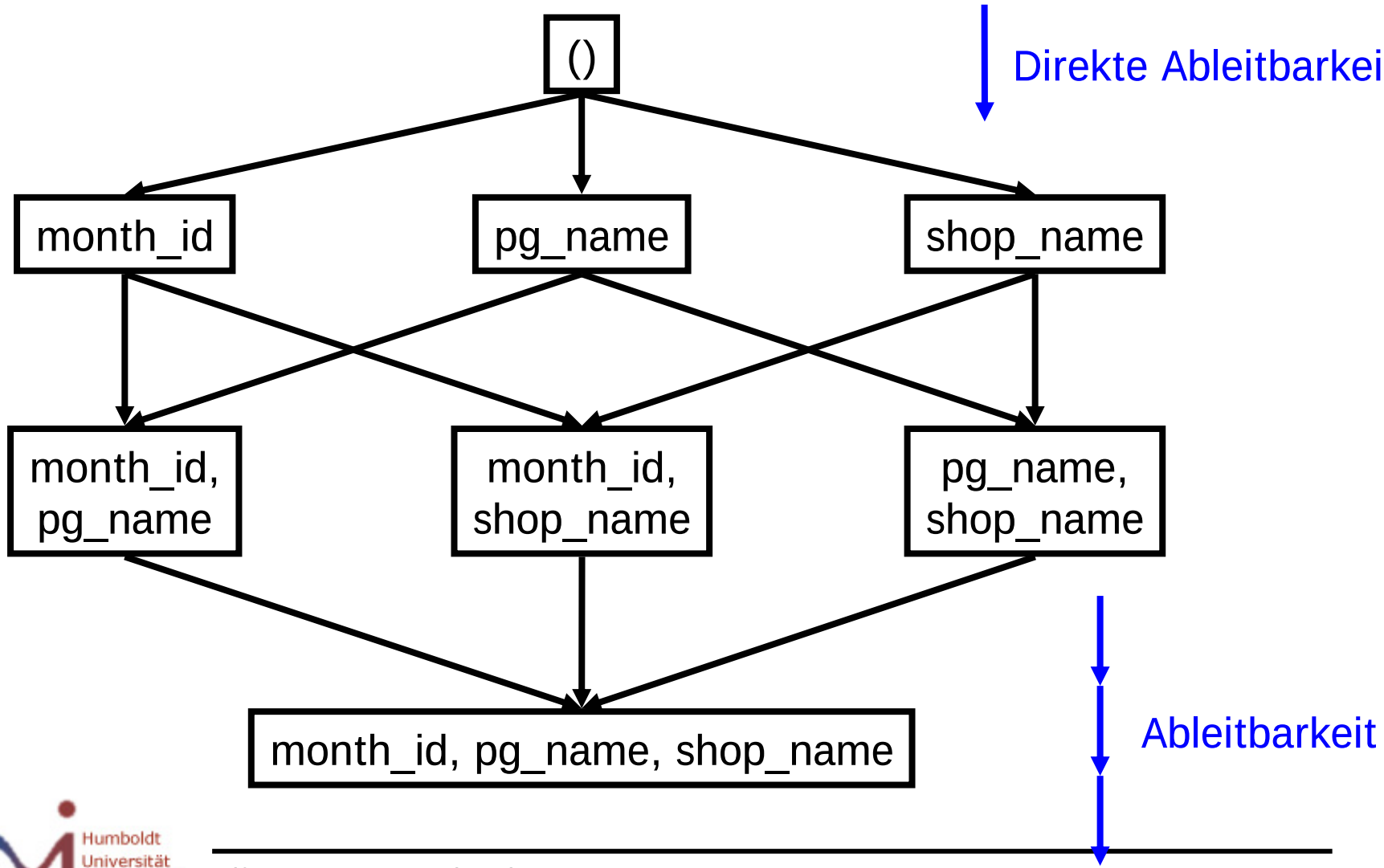


```
SELECT  T.year_id, v.pg_id,
        v.shop_name, v.sum
FROM    v, Time T
WHERE   v.month_id = T.month_id
GROUP BY T.year_id, v.pg_id,
          v.shop_name
```

```
SELECT  v.month_id, P.pg_name,
        v.shop_name
FROM    v, Product P
WHERE   v.pg_id = P.pg_id
```

- Erfordert Rewriting mit zusätzlichen Joins
- Eventuell weitere Aggregationen
  - Abh. innerhalb einer Klassifikationsstufe: Keine Agg.
  - Abh. zwischen K-stufen: Agg. entlang der Hierarchie

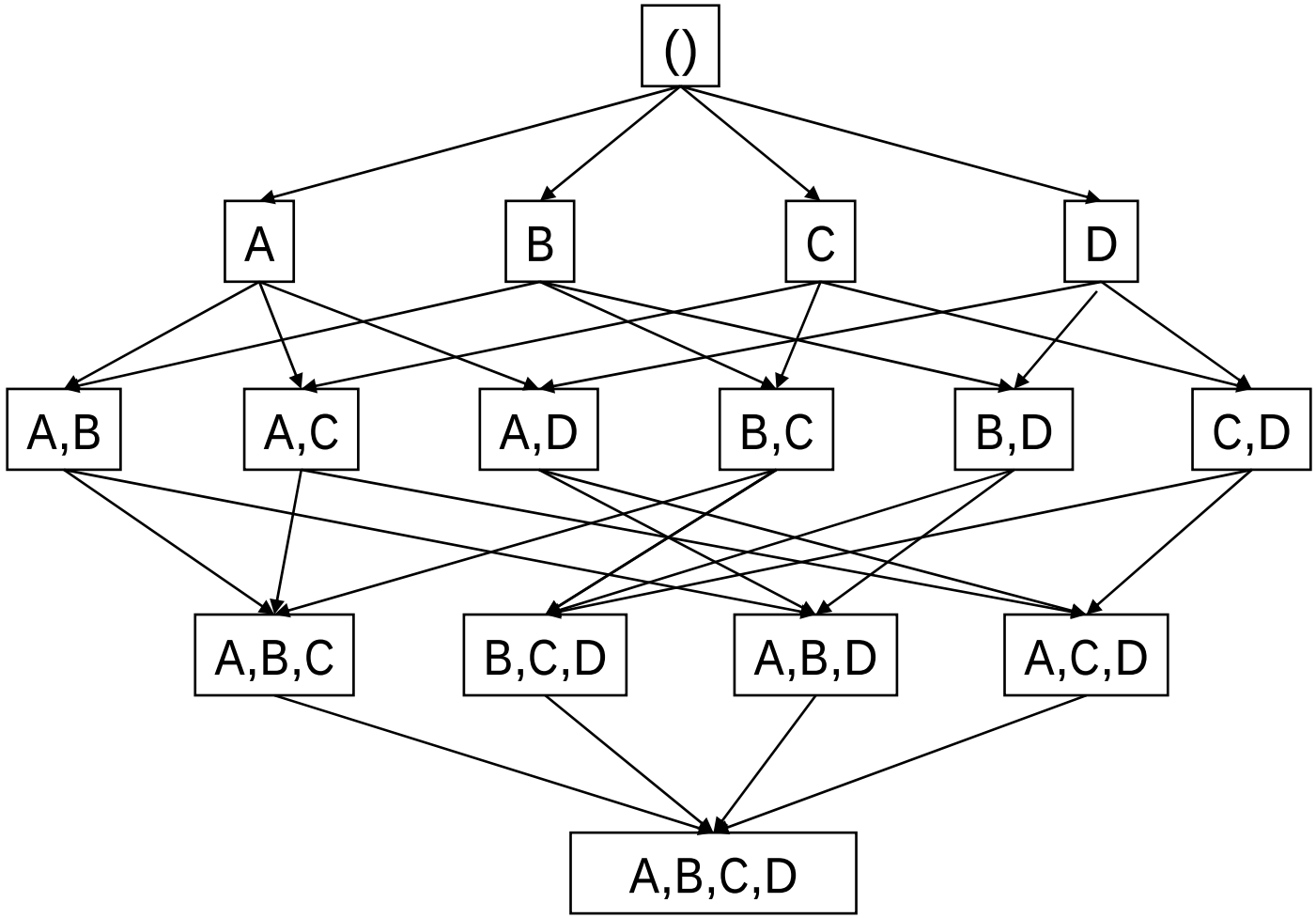
Aggregationsgitter



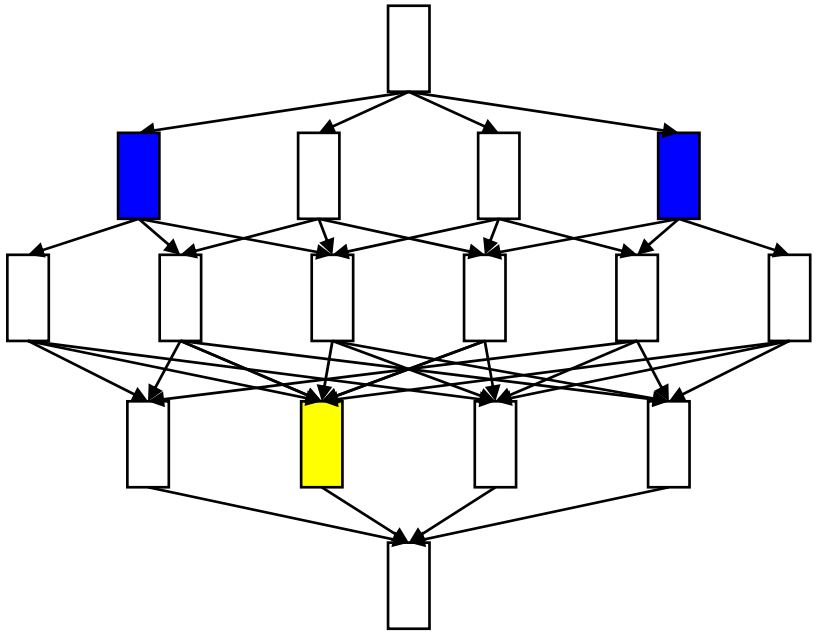
Aggregationsgitter: Verwendung

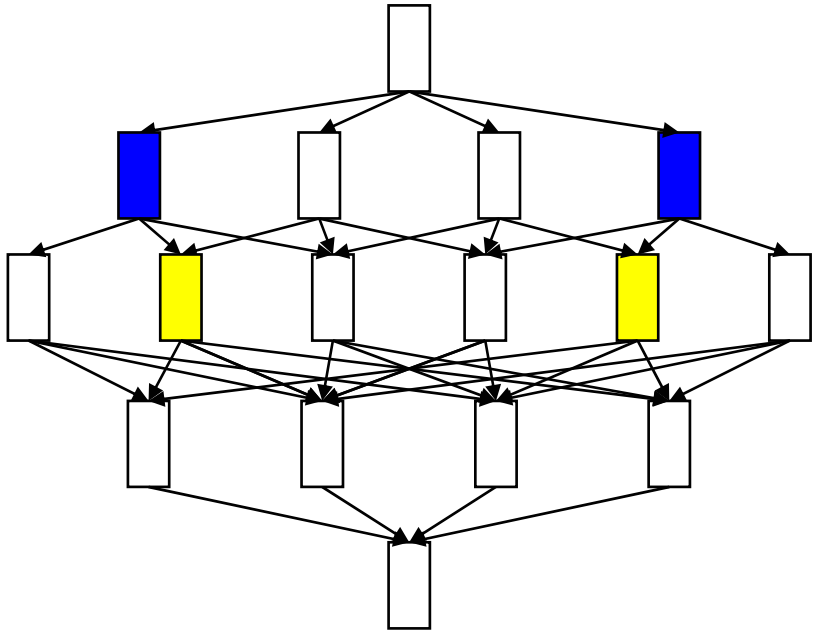
- Gegeben eine Menge Z von Queries mit Gruppierung
- Gesucht: „Optimale“ Menge Z‘ von Views, mit der sich alle Queries berechnen lassen
  - Optimal: Maximale Zeitersparnis
  - Optimal: Minimaler Platzverbrauch/ Aktualisierungszeit
- Mögliches Vorgehen
  - Erstellen des kompletten Aggregationsgitter des Models
  - Markierung der Gruppierungsmengen aus Z
  - Wahl von Z‘
    - Kleinster gemeinsamer Vorfahr aller Knoten aus Z
      - Hoher Platzverbrauch, da alles aus kleiner Granularität berechnet
      - U.U. hoher Nach-Berechnungsaufwand
    - Menge von Vorfahren, so dass alle Z erreichbar sind
      - U.U. hoher Platzverbrauch, da mehrere Views materialisiert werden

4 Klassifikationsstufen



Auswahlmöglichkeiten





Kleinster gemeinsamer Vorfahr

- Mehrere zur Auswahl
- Materialisierung mit hoher Auflösung
- Zwei Aggregationen pro Query

Menge von Vorfahren

- Mehrere zur Auswahl
- Mehrere Materialisierungen zu aktualisieren
- Eine Aggregation pro Query

Platzverbrauch

- Größe des Aggregationsgitter wächst exponentiell in der Anzahl möglicher Klassifikationsstufen
  - n Klassifikationsstufen - 2^n mögliche Gruppierungen
  - Identisch zur Anzahl berechneter Gruppierungen des CUBE Operators
- Platzverbrauch einer materialisierten Gruppierung hängt ab von Anzahl Klassifikationsknoten der beteiligten Gruppierungsattribute
  - Durch Nicht-speichern der 0-Summen kann Platz gespart werden („ ... `having SUM() != 0` ...“)
  - Keine Auswirkung auf abgeleitete Summen
  - Ersparnis abhängig von Füllgrad des Würfels. Bedingt die Wahrscheinlichkeit, dass eine konkrete Kombination von Klassifikationsknoten in der Gruppierung vorhanden ist

Auswahl der optimalen Views

- Abgeschätzt werden muss
  - Platzverbrauch
  - Aktualisierungskosten
  - Gewinn für gegebene Workload
- Verschiedene Algorithmen sind vorgeschlagen worden
  - Optimale Lösung ist NP hartes/vollständiges Probleme
  - ➢ Heuristiken (Greedy, Simulated Annealing, ...)
- ➢ Literatur

Inhalt dieser Vorlesung

- Anfrageoptimierung mit MVs
- Materialisierte Sichten in Oracle

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

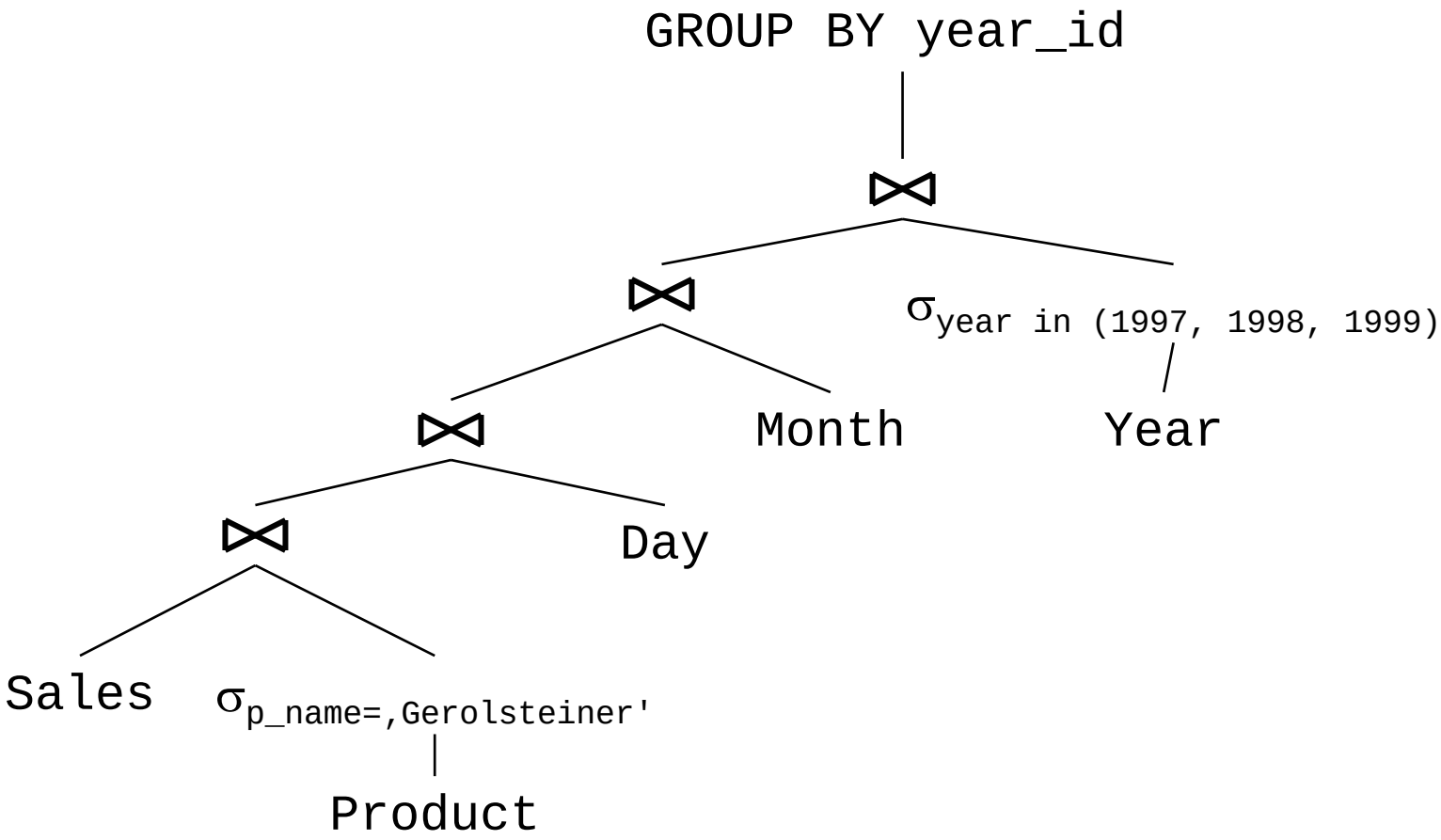
- Wo passen hier materialisierte Sichten?
- Sketch
  - Jede materialisierte Sicht ist ein potentieller Teilplan
  - Bei Bottum-Up Bewertung von Teilplänen auch materialisierte Sichten berücksichtigen
    - Matching: Gibt es für den aktuellen Teilplan einen / mehrere geeignete MVs?
    - Hinzufügen von 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
```

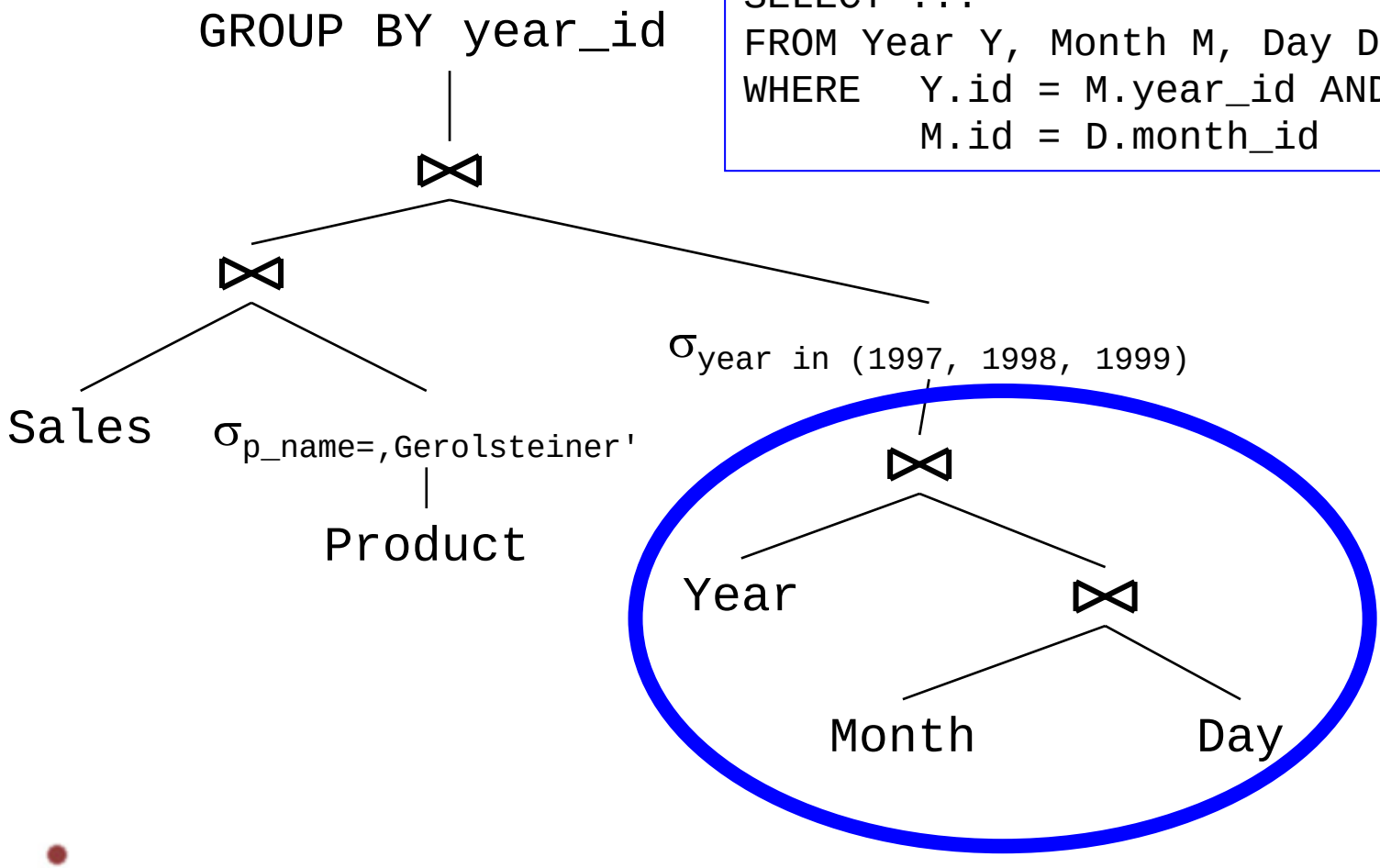
```
CREATE MATERIALIZED VIEW v_time AS
SELECT Y.id, Y.year, M.id, M.month, D.id, D.day
FROM Year Y, Month M, Day D
WHERE  Y.id = M.year_id AND
       M.id = D.month_id
```

Ausführungsplan Query

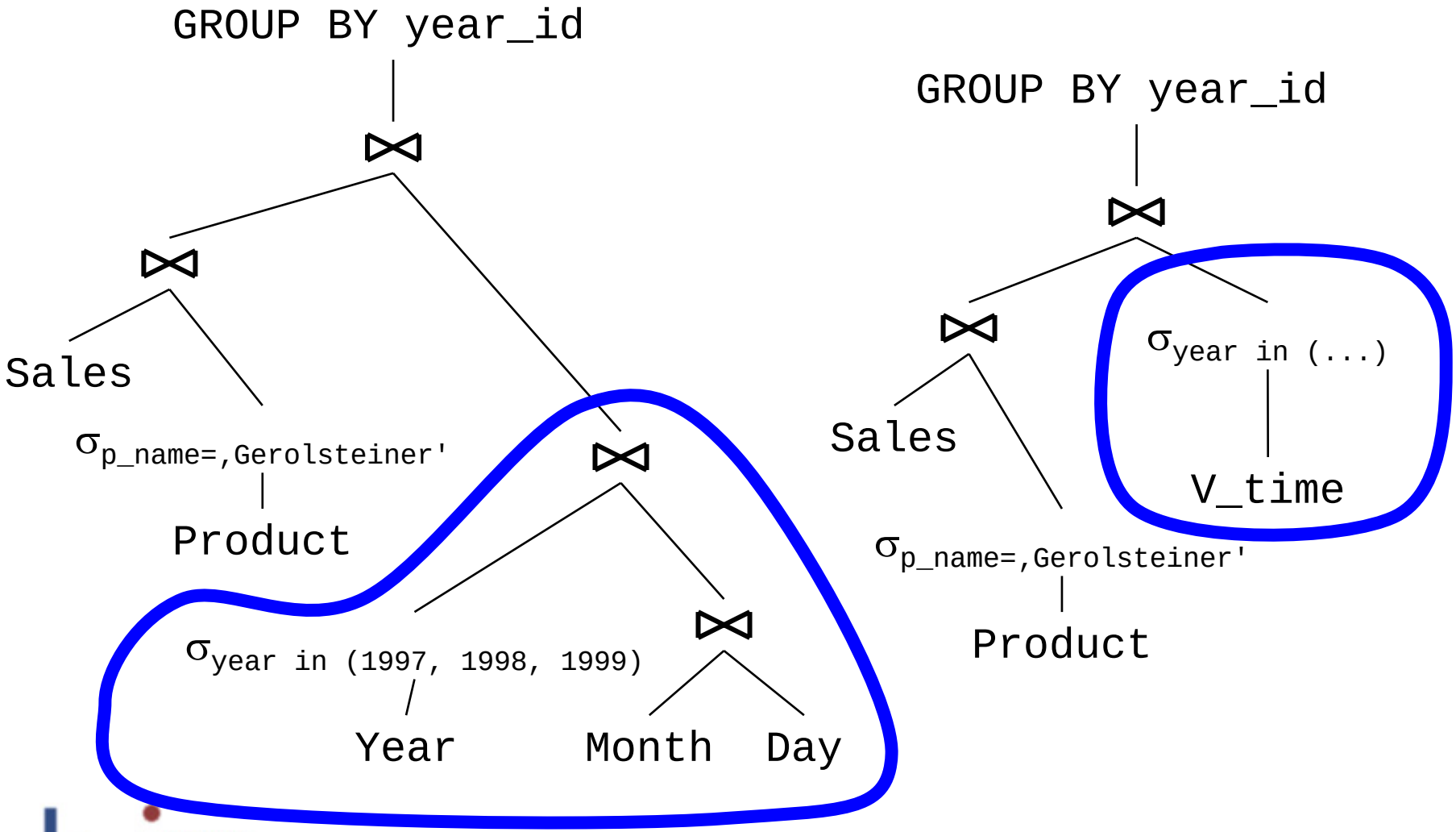


Alternativplan

```
CREATE MATERIALIZED VIEW v_time AS
SELECT ...
FROM Year Y, Month M, Day D
WHERE   Y.id = M.year_id AND
        M.id = D.month_id
```



Alternativen bewerten



Einschränkungen

- Matching muss sehr schnell gehen
  - Exponentielle Algorithmen vermeiden
- 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 (Stand 2001)
  - 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 where ...
    - Q: select ... where A*B>50
    - V: select A*B ... where
  - ...

Materialisierte Sichten in Oracle

- Anlegen von materialisierten Sichten
- Aktualisierungstechniken
- Optimierung / Rewriting mit materialisierten Sichten

Definition von MVs

```
CREATE MATERIALIZED VIEW product_sales_mv
BUILD IMMEDIATE
REFRESH FAST
ENABLE QUERY REWRITE
AS
SELECT  P.prod_name, SUM(S.amount), COUNT(*)
FROM    sales S, products P
WHERE   S.product_id = P.id
GROUP BY P.prod_name;
```

- Materialisierte Sicht wird als Tabelle angelegt
  - Indexierung, Partitionierung, STORAGE Klauseln, etc.
- Aktualisierung erfolgt über Trigger
- „Query Rewrite“ muss explizit erlaubt werden

Reengineering

- Materialisierte Sichten können über existierende Tabellen gelegt werden

```
CREATE MATERIALIZED VIEW sum_sales_tab
ON PREBUILT TABLE
ENABLE QUERY REWRITE
AS
...
```

- Nachträgliche Benutzung der Rewrite / Aktualisierungsmechanismen

Aktualisierungsstrategien

- „Mastertabellen“ – Im MV referenzierte Tabellen
- Wann wird aktualisiert
  - „On Demand“ – Nur bei Aufruf von speziellen Funktionen im Package DBMS_MVIEW
  - „On Commit“ – Beim Abschluss von Transaktionen, die mindestens eine Mastertabelle verändern
- Wie wird aktualisiert
  - „COMPLETE“ – Komplette Neuberechnung des Views bei Änderungen an mindestens einer Mastertabelle
  - „FAST“ – Inkrementelles Nachführen von Änderungen
    - Was nur unter bestimmten Bedingungen geht
  - „FORCE“ – Ausführung von FAST, wenn möglich; sonst COMPLETE

FAST Refresh

- Erfordert das explizite Anlegen eines „MV Logs“

```
CREATE MATERIALIZED VIEW LOG ON sales
WITH SEQUENCE, ROWID
(prod_id, cust_id, time_id, channel_id,
 promo_id, quantity_sold, amount_sold)
INCLUDING NEW VALUES;
```

- Legt Trigger auf Mastertabellen an
- MV-Log speichert Deltas von Änderungen
- FAST Refresh spielt diese Deltas in MV ein
- Diverse Einschränkungen
  - *A materialized view log must be present for each table.*
  - *The rowids of all the tables must appear …*
  - *If there are no outer joins, you …*
  - *If there are outer joins, …*

Rewrite Methoden

- Oracle versucht Rewriting auf mehrere Arten (V9.2)
  - Rewriting erfolgt auf syntaktischer Ebene, nicht im Operatorbaum
- Text Match
  - „Full Text Match“ und „Partial Text Match“
  - Rein syntaktischer Match
    - Keine Unterscheidung von Groß-/ Kleinschreibung und Leerzeichen
    - Aber: Reihenfolge von Bedingungen, andere Tabellenaliase, etc.
  - Keine logische Implikation, kein Mapping, etc.
- General Query Rewrite
  - Nicht anwendbar bei „Complex materialized views“
  - Unterschiedliche Methoden ja nach Art des MV
    - Only join – only aggregate – join and aggregate - complex
  - Viele Einschränkungen - Dokumentation

Zusammenfassung MVs

- Hochkomplexes Thema
- Ultimative Anfragebeschleunigung möglich – keinerlei Berechnung mehr
- Trade-Off zwischen
  - Performancegewinn
  - Platzbedarf, Aktualisierungskosten
- Query Rewrite in RDBMS funktioniert mit Heuristiken