Data Warehousing und Data Mining

Materialisierte Sichten II: Aktualisierung und Auswahl

Ulf Leser

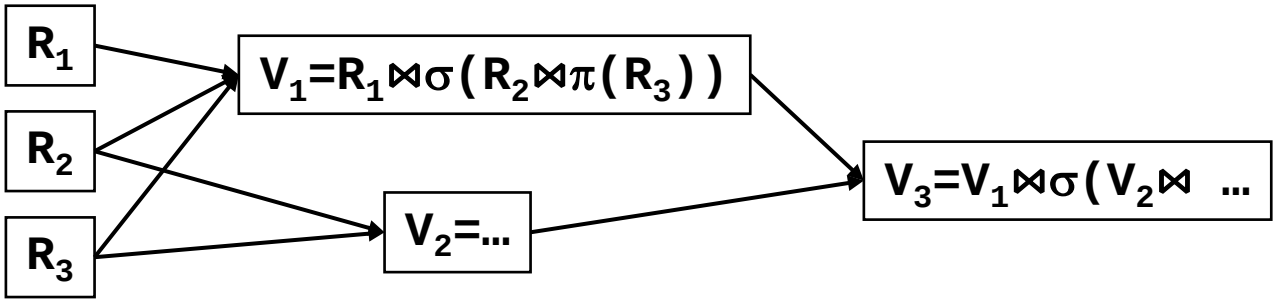
Wissensmanagement in der Bioinformatik

Inhalt dieser Vorlesung

- Aktualisierung materialisierter Sichten
- Konsistenz in DWH
- MV‘s in Oracle
- Auswahl materialisierter Sichten

Das Problem

- Materialisierte Sichten sind Anfragen auf Basisdaten
- Wenn sich Basisdaten ändern, müssen auch die gespeicherten MV angepasst werden
  - Sonst führt die Verwendung zu falschen Ergebnissen
- Kritisch ist es auch, wenn unterschiedliche MVs zu unterschiedlichen Zeiten aktualisiert werden
  - Verwendung in einer Query führt zu inkonsistenten Ergebnissen



Aktualisierung: Wann und Wie

- Wann?
  - Synchron: Bei allen (comitteten) Änderungen auf Basistabellen
    - Z.B. durch Trigger; laufen in gleicher Transaktion
  - Asynchron: Auf Benutzeranforderung (z.B. zu festen Zeiten)
- Wie?
  - Komplette Neuberechnung
    - Sehr teuer, sollte nicht zu häufig erfolgen
  - Inkrementelle Aktualisierung
    - Nur Änderungen an Basistabellen werden eingespielt
    - Delta zwischen Zeitpunkt T der letzten Aktualisierung und Zeitpunkt X der aktuellen Aktualisierung müssen ermittelt werden
    - Für eine Tabelle: $R^X = R^T + \Delta R$; wir schreiben meist: $R' = R + \Delta R$
    - Unterscheidung zwischen INS und DEL notwendig (UPD=DEL;INS)

Vorsicht

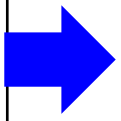
- „R+ΔR“ heißt: Wende ΔR auf R an
  - INS-Tuple werden gefügt
  - DEL-Tupel werden gelöscht
- „+“ ist fast „∪“
- Wir müssen uns die Unterschiede genau ansehen

Beispiel

Basistabelle R		
P	**PN**	**SN**
100	Geroldsteiner	Wedding
120	Spreequelle	Kreuzberg
150	Fanta	Mitte
80	Geroldsteiner	Pankow
250	Geroldsteiner	Kreuzberg

$V=\sigma_{P>110}(R)$		
P	**PN**	**SN**
120	Spreequelle	Kreuzberg
150	Fanta	Mitte
250	Geroldsteiner	Kreuzberg

```
INSERT (120,Limo,Wedding);
INSERT (80, Cola, Dahlem);
DELETE (150, Fanta, Mitte);
…
```

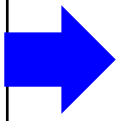


ΔR			
P	**PN**	**SN**	**LOG**
120	Limo	Wedding	INS
80	Cola	Dahlem	INS
150	Fanta	Mitte	DEL

Beispiel

Basistabelle R		
P	**PN**	**SN**
100	Geroldsteiner	Wedding
120	Spreequelle	Kreuzberg
150	Fanta	Mitte
80	Geroldsteiner	Pankow
250	Geroldsteiner	Kreuzberg

```
INSERT (120,Limo,Wedding);
INSERT (80, Cola, Dahlem);
DELETE (150, Fanta, Mitte);
…
```

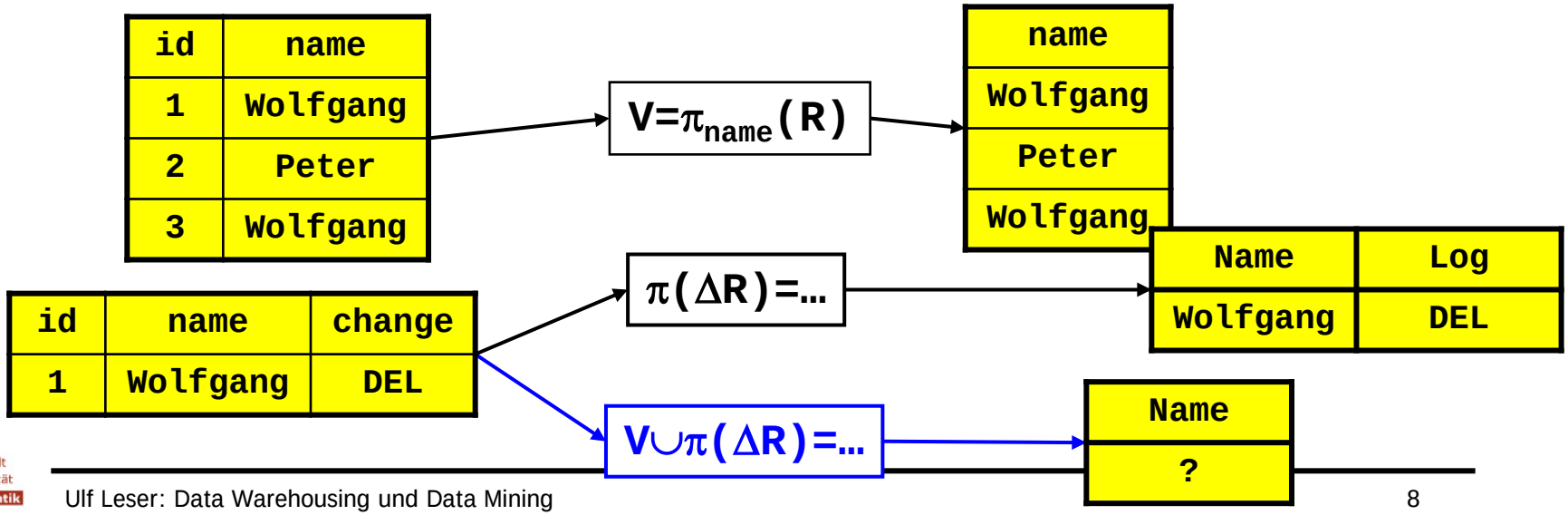


ΔR			
P	**PN**	**SN**	**LOG**
120	Limo	Wedding	INS
80	Cola	Dahlem	INS
150	Fanta	Mitte	DEL

$V=\sigma_{P>110}(R)$		
P	**PN**	**SN**
120	Spreequelle	Kreuzberg
150	Fanta	Mitte
250	Geroldsteiner	Kreuzberg
120	Limo	Wedding

MV: Selektion und Projektion

- Zunächst: MVs ohne Join aber mit Selektion / Projektion
- Selektion: Ausnutzung der Distributivität der Selektion
  - Sei $V=\sigma(R)$
  - $V' = \sigma(R') = \sigma(R+\Delta R) = \sigma(R) \cup \sigma(\Delta R) = V \cup \sigma(\Delta R)$
  - Sehr schön: Wir müssen nur das Delta betrachten
- Projektion
  - Gilt $V' = \pi(R') = \pi(R+\Delta R) = \pi(R) \cup \pi(\Delta R) = V \cup \pi(\Delta R)$?



MV mit Projektionen

- Projektion ist nicht distributiv bei
  - Set-semantik (die reale Systeme nicht verwenden)
  - Anfragen mit DISTINCT
- Lösung
  - Materialisierte Sichten sollten immer einen Schlüssel mitführen
    - Keine Probleme mit Duplikaten mehr
  - Sonst: Zählerspalte Z pro Tupel im MV einführen
    - Bei INS erhöhen
    - Bei DEL erniedrigen
    - Tupel erst löschen, wenn Z=0

MV mit Joins

- Wir beginnen mit nur einem Join und Änderungen an beiden Basistabellen
- Relationale Algebra würde sagen (wenn „+“=„∪“ wäre):

$$
\begin{aligned}
V' &= R' \bowtie S' = (R \cup \Delta R) \bowtie S' \\
&= (R \bowtie S') \cup (\Delta R \bowtie S') \\
&= (R \bowtie (S \cup \Delta S)) \cup (\Delta R \bowtie S') \\
&= (R \bowtie S) \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie S') \\
&= V \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie S') \\
&= V \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie (S \cup \Delta S)) \\
&= V \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie S) \cup (\Delta R \bowtie \Delta S)
\end{aligned}
$$

- Aber: Das stimmt so nicht für „+“

Gegenbeispiel

$$V \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie S) \cup (\Delta R \bowtie \Delta S)$$

id	Name	Change
1	A	
2	B	
3	C	
2	A	DEL
3	B	DEL
4	D	INS
5	E	INS

$\bowtie$

id	Age	Change
1	21	
2	22	
4	24	
2	22	DEL
3	23	INS
4	24	DEL
5	25	INS

$=$

id
1
2

$\cup$

id	Change
2	DEL
3	INS

$\cup$

id	Change
2	DEL
4	INS

$\cup$

id	Change
2	DEL
5	INS

id
1
5

$\neq$

id	Change
1	
2	
2	DEL
3	INS
4	INS
5	INS

$=$

Berechnung von ΔV bei Joins

ΔR \ ΔS	INS	DEL	Schon in S
INS	INS	?	INS
DEL	?	DEL	DEL
Schon in R	INS	DEL	Ignorieren

- Problem: DEL/INS Kombinationen
  - Tupel 3 im Beispiel: Wird eingefügt durch (R⋈ΔS) und nicht gelöscht durch (ΔR⋈ΔS)
  - Problem sind die Joins von Deltas mit Deltas
  - Für korrekte Lösung bräuchte man die Zeitpunkte aller Änderungen - teuer

MV mit Joins

- Also: Joins von Deltas vermeiden

$$
\begin{aligned}
V' &= R' \bowtie S' = (R \cup \Delta R) \bowtie S' \\
&= R \bowtie S' \cup \Delta R \bowtie S' \\
&= R \bowtie (S \cup \Delta S) \cup \Delta R \bowtie S' \\
&= V \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie S')
\end{aligned}
$$

Beispiel

$$V \cup (R \bowtie \Delta S) \cup (\Delta R \bowtie S')$$

id	Name	Change
1	A	
2	B	
3	C	
2	A	DEL
3	B	DEL
4	D	INS
5	E	INS

$\bowtie$

id	Age	Change
1	21	
2	22	
4	24	
2	22	DEL
3	23	INS
4	24	DEL
5	25	INS

=

id	Change
1	
2	

$\cup$

id	Change
2	DEL
3	INS

$\cup$

id	Change
3	DEL
5	INS

=

id	Change
1	
5	INS

MV mit Joins

- Das würde klappen, aber effizient ist es nicht
- Wir benötigen den alten Zustand R (bzw. S) zur Berechnung von **V'**
  - Das ist teuer – alle Relationen müssen doppelt gehalten werden
  - Wunsch: R/S „as usual“ aktualisieren und nur die Logs zusätzlich halten
  - Wir dürfen zur Berechnung von V' nur V, R', S', Δ**R** und Δ**S** verwenden

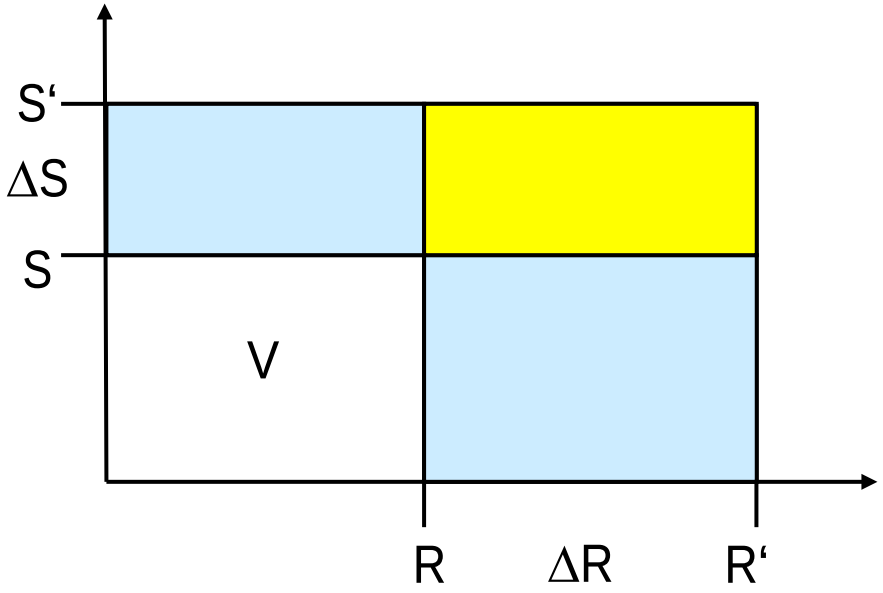
Vorhalten zweier Zustände?

- Möglichkeit 1: R bei Aktualisierung von V rück-berechnen
  - **R = R‘ - ΔR**
- Möglichkeit 2: Auf Löschoperationen verzichten
  - Das ist in DWH durchaus realistisch
  - Dann gilt das Distributivgesetz für „+“ und damit zunächst

    V‘ = V ∪ (R⋈ΔS) ∪ (ΔR⋈S) ∪ (ΔR⋈ΔS)
  - Dann kann man R und S wie folgt eliminieren

V = R⋈S = (R‘-ΔR)⋈(S‘-ΔS) =
= (R‘⋈S‘) - (R‘⋈ΔS) - (ΔR⋈S‘) ∪ (ΔR⋈ΔS)
V‘ = R‘⋈S‘
ΔV = V‘ – V =
= (R‘⋈ΔS) ∪ (ΔR⋈S‘) - (ΔR⋈ΔS)

Verdeutlichung



- Vorwärts: **V‘ = V ∪ (R⋈ΔS) ∪ (ΔR⋈S) ∪ (ΔR⋈ΔS)**
  - Alle Quadrate werden einzeln addiert
- Rückwärts: **ΔV = (R‘⋈ΔS) ∪ (ΔR⋈S‘) - (ΔR⋈ΔS)**
  - Gelbes Quadrat wird zwei mal addiert und einmal subtrahiert

MV mit vielen Joins (ohne Löschen)

- Das Distributivgesetz gilt genauso
- Zwei Joins: $V = R \bowtie S \bowtie T$
  - $V = R \bowtie S \bowtie T$
    $= (R' - \Delta R) \bowtie (S' - \Delta S) \bowtie (T' - \Delta T)$
    $= (R' \bowtie S' \bowtie T') - (R' \bowtie S' \bowtie \Delta T) - (R' \bowtie \Delta S \bowtie T') \ldots - (\Delta R \bowtie \Delta S \bowtie \Delta TS)$
- Viele Joins: $V = R \bowtie S \bowtie T \ldots \bowtie Z$
  - Jeder Term ist eine Anfrage, die man auswerten muss
  - Wie viele Terme bei n Joins?
    - 2^{n+1}

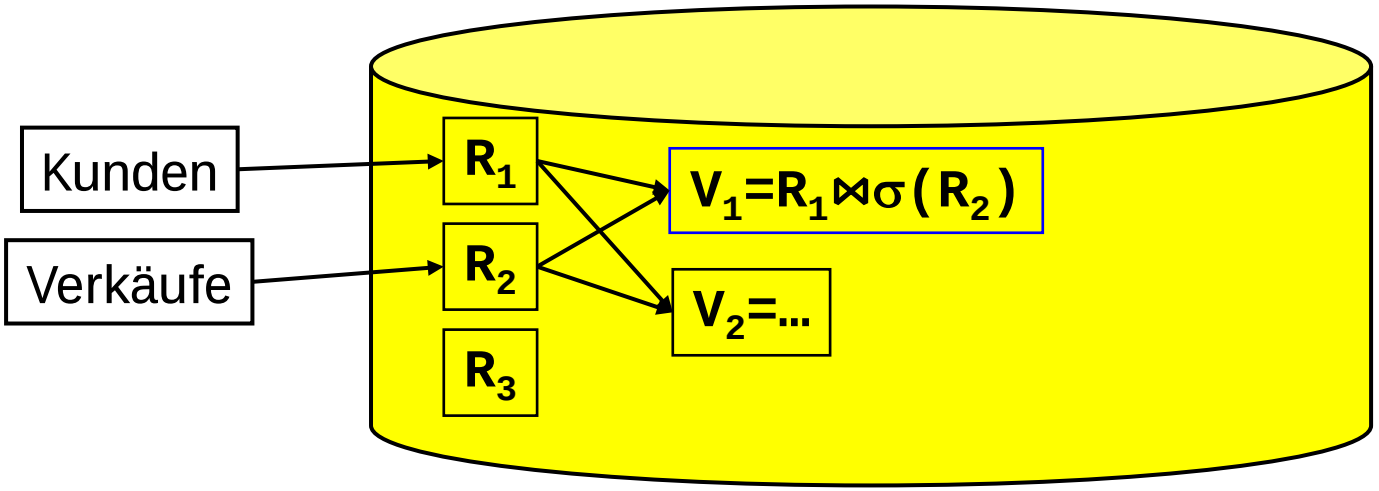
Fazit

- Wenn man Löschen verbietet, sind MV bestehend aus SPJ (ohne DISTINCT) effizient aktualisierbar
- Auch Gruppierungen und Aggregation sind gut handhabbar
- Wenn man immer PK‘s mitnimmt, ist auch DISTINCT OK
- Schwierig sind komplexere MV
  - EXIST Klausel, NOT
  - Holistische Aggregatfuntionen
  - Geschachtelte Queries, die man nicht in Joins umformen kann
    - Unkorreliert / korreliert
  - ...

Inhalt dieser Vorlesung

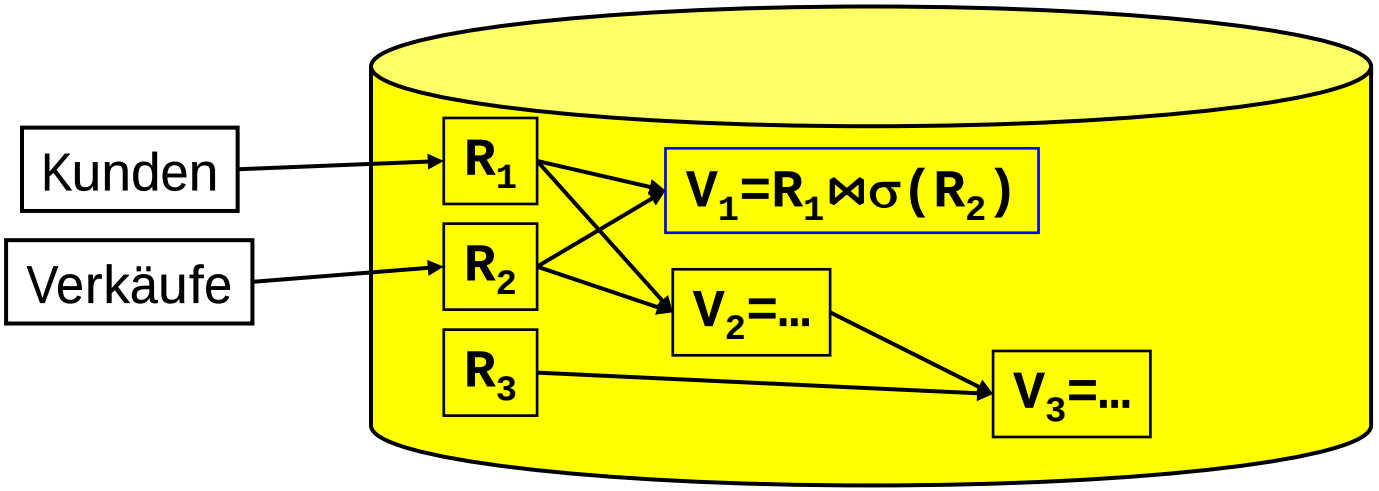
- Aktualisierung materialisierter Sichten
- Konsistenz in DWH
- MV‘s in Oracle
- Auswahl materialisierter Sichten

Inkonsistente Daten 1



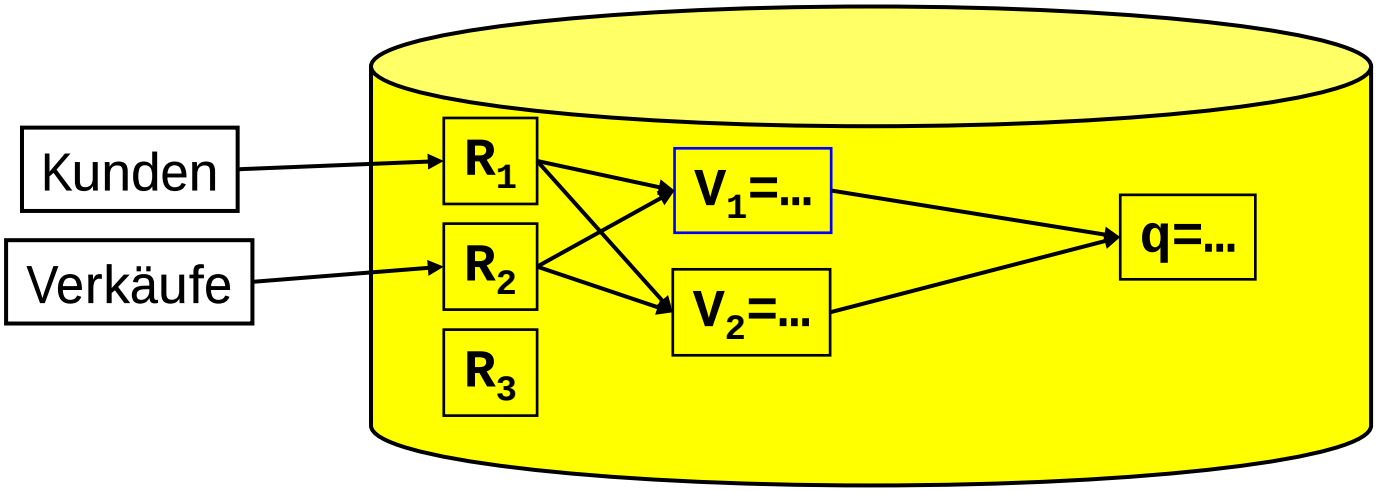
- Zwei Aktualisierungen: Quellen – Basisdaten – MVs
- Beispiel 1
  - Daten für R_2 bis 31.3.2007 werden eingespielt
  - Aber R_1 ist noch auf dem Stand 28.2.2007
  - Aktualisierung von V_1 führt zu „dangling“ Verkäufen
    - Verkäufe mit nicht-existenten Kunden
  - Inkonsistenter Zustand

Inkonsistente Daten 2



- Beispiel 2
  - R1 – R3 sind auf gleichem Zustand (gleicher Zeitpunkt)
  - V_3 wird vor V_2 aktualisiert (auf Basis des alten V_2)
  - Inkonsistenter Zustand

Inkonsistente Daten 3



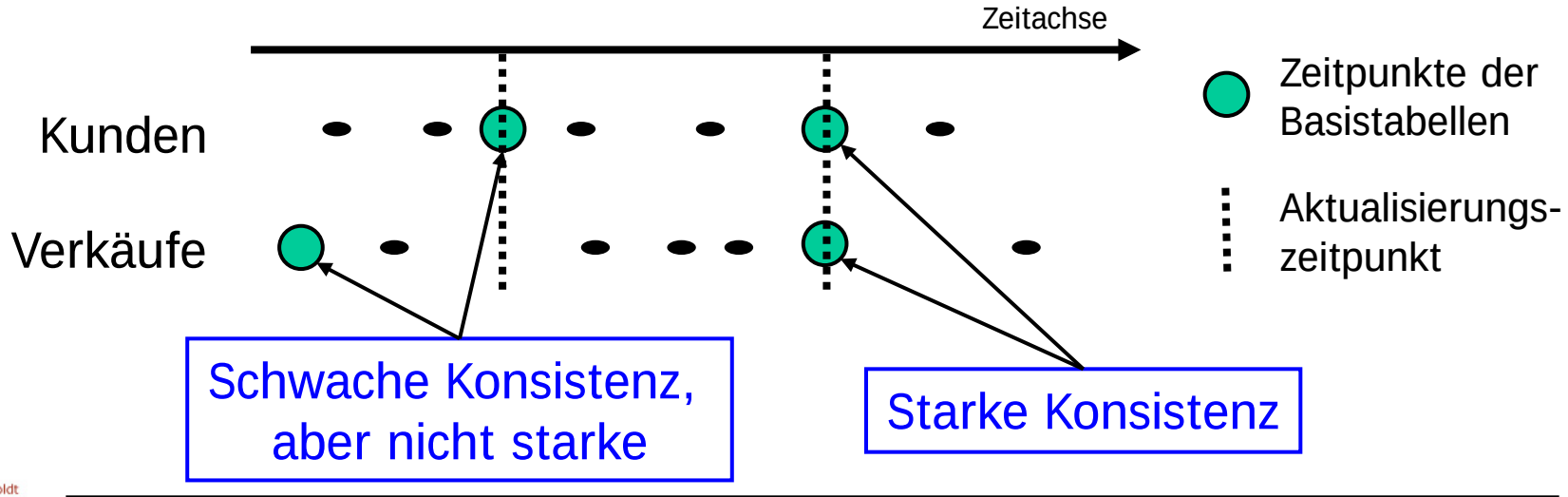
- Beispiel 3
  - R_1 – R_3 sind auf gleichem Zustand (gleicher Zeitpunkt)
  - V_1 wird aktualisiert, V_2 ist noch auf altem Zustand
  - Query wird umgeschrieben in $V_1 \bowtie V_2$
  - Inkonsistentes Ergebnis

Lokale und globale Konsistenz

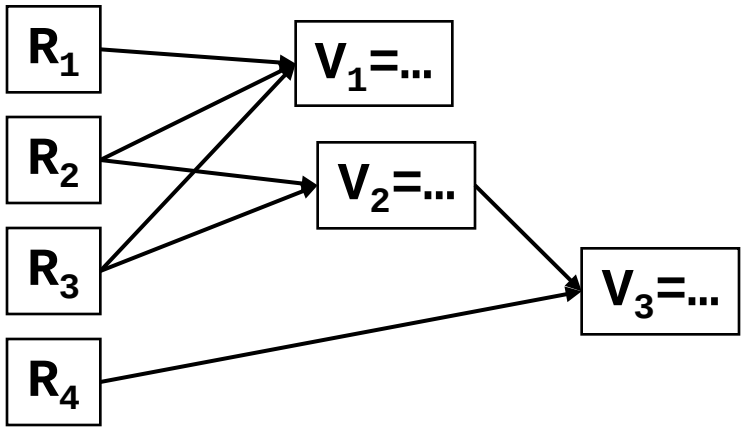
- Idealfall
  - Alle Quellen exportieren nur in-sich konsistente Zustände
    - Das nehmen wir implizit immer an
  - Alle Quellen exportieren zum gleichen Zeitpunkt X
    - Globale Konsistenz
    - Das DWH kann diese Bedingung kaum überwachen
      - Plausibilitätstest über Zeitstempel möglich, aber keine Garantie
  - Voraussetzung: Vor Export müssen alle laufenden Transaktionen in allen Quellen abgeschlossen sein
- Realität
  - Quellen exportieren zu unterschiedlichen Zeitpunkten
    - Die sie selber festlegen
  - TX-Management einer Quelle kann nicht auf andere warten
    - Es kommen schon global inkonsistente Daten ins DWH

Konsistenzgrade

- Schwache Konsistenz
  - Views bzgl. ihrer Basisrelationen und Basisrelationen bzgl. ihrer Quellen konsistent
  - Das ist nicht notwendigerweise ein global gültiger Zustand
- Starke Konsistenz
  - Alle Views entsprechen einem global gültigen Zustand
  - Erfordert synchronisierte Exports / MV-Aktualisierungen



Aktualisierung vieler Views



- Erfolgt die Aktualisierung jedes Views als eigene Transaktion, sind die Views zwischenzeitlich zueinander inkonsistent
- Alles in einer Transaktion führt zu einer langen Sperre
- Besser: Simple Painting [ZWG97]
  - Zerlegung der Deltas in „Mini-Deltas“
  - Einspielen eines Mini-Deltas in alle Views ist eine Transaktion

Inhalt dieser Vorlesung

- Aktualisierung materialisierter Sichten
- MV‘s in Oracle
  - Anlegen von materialisierten Sichten
  - Aktualisierungstechniken
  - Optimierung / Rewriting mit materialisierten Sichten
- Auswahl materialisierter 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.
- „Query Rewrite“ ist per default TRUE
- Globaler Session-Parameter „query_rewrite_integrity“
  - Wenn „enforced“ – kein Rewrite mit „stale“ MVs

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

- Wann wird aktualisiert
  - ON DEMAND: Nur bei Aufruf von speziellen Funktionen im Package DBMS_MVIEW
    - **`dbms_mview.refresh (mv), refresh_all, refresh_dependent(tab)`**
  - ON COMMIT: Beim Abschluss jeder Transaktion, die mindestens eine Basistabelle verändern
    - Bestmögliche Aktualität
    - Schwieriger Konsistenzbegriff
  - On Statement:A Aktualisierung des MV synchron mit Basistabs
- Wie wird aktualisiert
  - COMPLETE: Neuberechnung des Views bei Änderungen an mindestens einer Basistabelle
  - FAST: Inkrementelles Nachführen von Änderungen
  - FORCE: Ausführung von FAST, wenn möglich; sonst COMPLETE

Kombinationen

	On commit	On demand
Complete		OK
Fast	OK	OK

Inkrementelle Aktualisierung (Fast)

- 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 zum Speichern der Deltas an
- Diverse Einschränkungen
  - Benötigt immer ein count(*) bei Gruppierung – Zählvariable
  - Rowids müssen immer dabei sein
- Aktualisierung kann auf Partitionen über ganze MV-Bäume laufen (synch_refresh)

Nested MV

- Die Definition eines MV kann einen anderen MV als Basistabelle benutzen
- Dann muss (für FAST refresh) ein MV Log auf dem MV angelegt werden
- Vorsicht: Unterschiedliche Refresh-Optionen innerhalb eines solchen MV Baums führen zu „unvorhersehbaren“ Effekten

Rewrite Methoden

- Oracle versucht Rewriting auf mehrere Arten
  - Vorsicht: Der benutze MV muss nicht aktuell sein
  - Optionen STALE_TOLERATED etc.
- Zwei Optionen
  - Text Match: Rein syntaktischer Match, erkennt nicht Reihenfolge von Bedingungen, andere Tabellenaliase, etc.
  - 10g: 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
- Seit 10gR2 können auch mehrere MVs für eine Query verwendet werden

2017: Query Rewrite Types

Query Rewrite Types	Dimensions	Primary Key/Foreign Key/Not Null Constraints
Matching SQL Text	Not Required	Not Required
Join Back	Required OR	Required
Aggregate Computability	Not Required	Not Required
Aggregate Rollup	Not Required	Not Required
Rollup Using a Dimension	Required	Not Required
Filtering the Data	Not Required	Not Required
PCT Rewrite	Not Required	Not Required
Multiple Materialized Views	Not Required	Not Required

- Read the manual!

Inhalt dieser Vorlesung

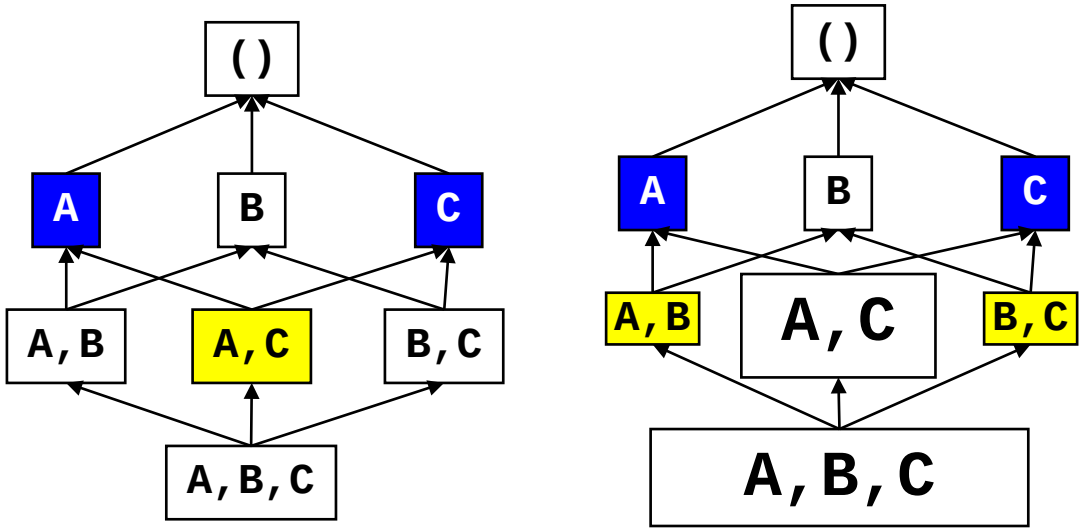
- Aktualisierung materialisierter Sichten
- MV‘s in Oracle
- Auswahl materialisierter Sichten
  - Statische Auswahl: Gewinn versus Platzverbrauch
  - Aktualisierungskosten

Auswahl materialisierter Sichten

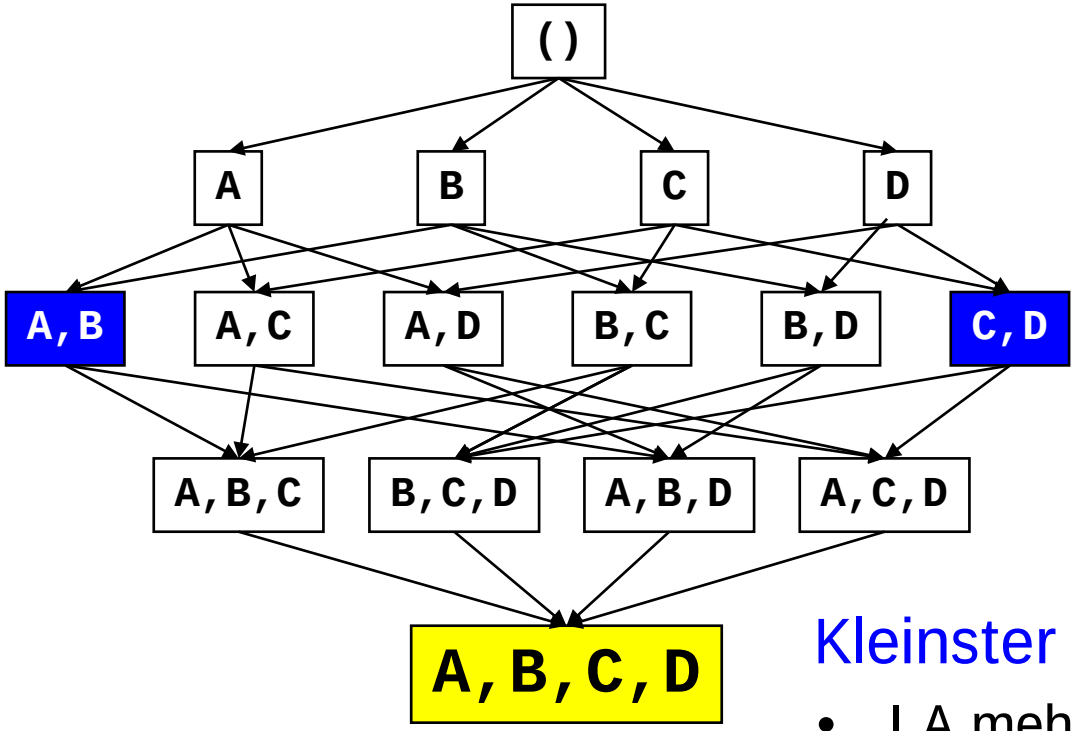
- Wir betrachten nur MV mit Aggregation über verschiedene Gruppierungen
  - Wiederholtes Scannen sehr großer Tabellen
  - Vorberechnung ist daher attraktiv
- Der vollständige CUBE benötigt zu viel Platz
  - Beispiel: 5 Dimensionen a 50 Knoten, keine Hierarchie
    - Gleichverteilung der Werte auf alle Knoten
  - Die höchste Granularität hat potentiell ~50^5=$3*10^8$ Tupel
  - Dazu: ~ $5*50^4+20*50^3+20*50^2+5*50 = 3.3*10^7$ Tupel mit weniger Gruppierungsattributen
  - Tatsächliche Zahlen abhängig vom Füllgrad des Cube ab

Welche MV soll man vorhalten?

- Die oft in Anfragen verwendet werden
- Aus denen man viele Anfragen schnell ableiten kann
- Die viel Arbeit in Anfragen sparen
- Die wenig Platz brauchen
- ...



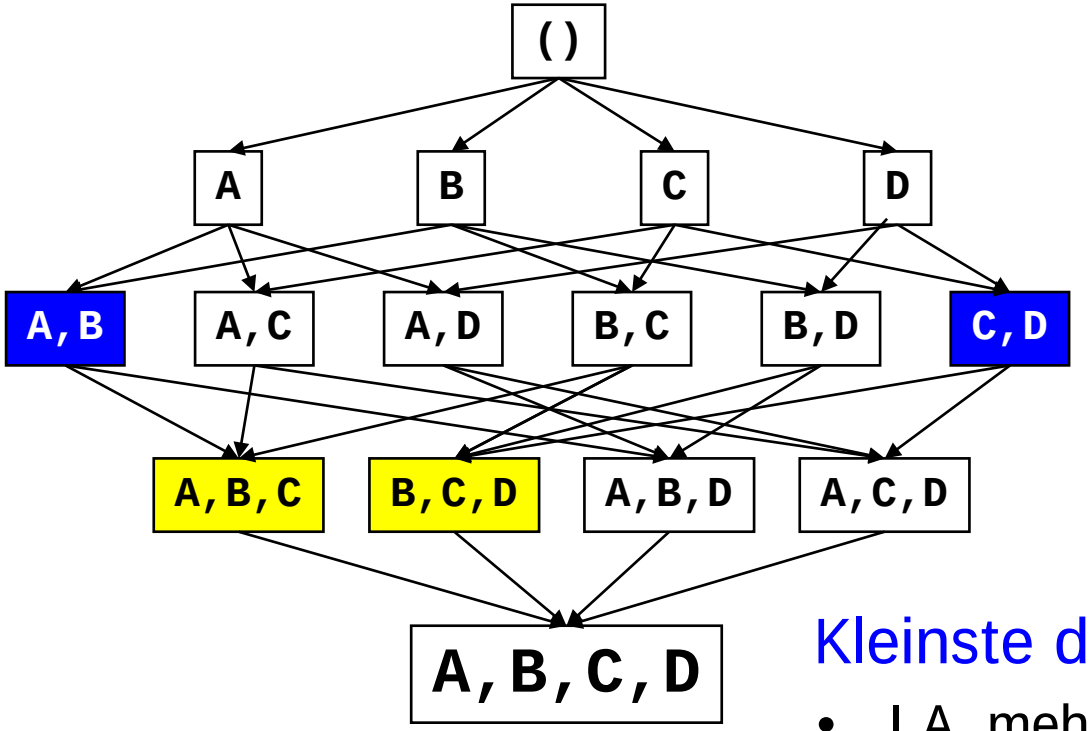
Kleinster gemeinsame Vorfahre?



Kleinster gemeinsamer Vorfahr

- I.A mehrere zur Auswahl
- Verlangt MV mit hoher Auflösung – braucht Platz

Kleinste direkte Vorfahren?



Kleinste direkte Vorfahren

- I.A. mehrere zur Auswahl
- MV mit niedrigerer Auflösung – weniger Platz
- Dafür mehrere MV notwendig

Statische Auswahl mit Platzbeschränkung

- Gegeben
  - Workload: Anfragen $Q=\{q_1,...,q_n\}$ mit Frequenzen $h_1,...,h_n$
  - Aggregationsgitter mit $A=\{a_1,...,a_k\}$ Gruppierungen
    - $|A|=2^d$ für d Dimensionen (ohne Hierarchien)
    - Jede a_i habe $|a_i|$ Tupel
  - Menge an maximal verfügbarem Platz: m
  - Geschätzte Kosten $c(q_i,a_j)$ zur Ableitung von q_i aus MV a_j
  - Wir ignorieren Aktualisierungskosten (später)
- Gesucht: Optimale Menge M von MV

Problem

- Definition

  Gegeben Workload Q, |Q|=n, Anfragefrequenzen h, Aggregationsgitter A, Kostenmatrix c und Konstante m. Das MV-Selektion Problem sucht die Menge M⊆A, die den folgenden Ausdruck minimiert

$$c(Q,M) = \sum_{i=1...n} h_i * \min_{a_j \in M}\left(c(q_i, a_j)\right)$$

  unter Beachtung der Nebenbedingung

$$\sum_{a_i \in M} |\, a_i \,| \leq m$$

Komplexität?

- Kann auf das Rucksack-Problem reduziert werden
- Das Problem ist NP-hart
  - Kann aber gut approximiert werden
- Wir brauchen Heuristiken

Greedy-Heuristik [HRU98]

- Grundidee: Iteratives Greedy-Verfahren
  - MV werden iterativ ausgewählt, solange noch Platz vorhanden
  - Auswahl erfolgt nach dem „Nutzen pro Platz“ des neuen MV
- Definition

  Gegeben eine Anfragemenge Q und eine Menge M von MV. Der Nutzen (Benefit) eines MV $a \notin M$ ist

$$B(a,Q,M) = c(Q,M) - c(Q,M \cup a)$$

Der Benefit-per-Space (BS) von a ist

$$BS(a,Q,M) = \frac{B(a,Q,M)}{|a|}$$

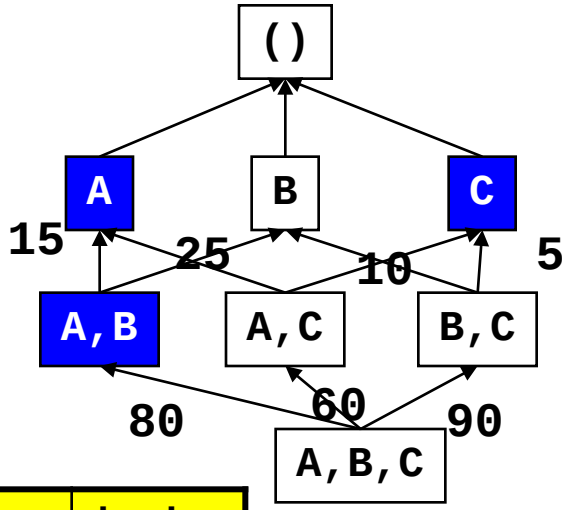
Sketch des Algorithmus

```
Input: Q, A, m;
Output: M;                    \\ Menge ausgewählter MV
while m>0                     \\ Noch verfügbarer Platz
  opt := 0;                   \\ Optimaler BS bisher
  for all a∈A-M               \\ Noch nicht materialisiert
    b := 0;                   \\ Nutzen von a
    for all q∈{Anfragen ableitbar aus a}
      b =+ c(q,M) – c(q,M∪a);  \\ Nutzen von a für q
    end for;
    if ((b\|a|)>opt)
      opt := b;               \\ Neues Optimum
      a_opt := a;
  end for;
  if ((m-|a_opt|)>0)
    M := M ∪ a_opt;           \\ Menge selektierter MV anpassen
    m =- |a_opt|;             \\ Freien Platz anpassen
  else
    m := 0;                   \\ Terminieren
end while;
return M;
```

Eigenschaften

- Nutzt den Platz nicht perfekt
  - Terminiert, wenn der nächste beste MV nicht mehr passt
- Theorem
  Der vorherige Algorithmus liefert eine Menge M von MV, deren Nutzen mindestens (63-x)% des optimalen Nutzen beträgt
  - *X: Größe des größten MV in Prozent der Gesamtgröße aller MV*
- Beweis: Literatur
  - Also: Wenn kein View mehr als 10% der Gesamtgröße aller MV benötigt, ist garantiert der Algorithmus eine Güte von 53%

Beispiel

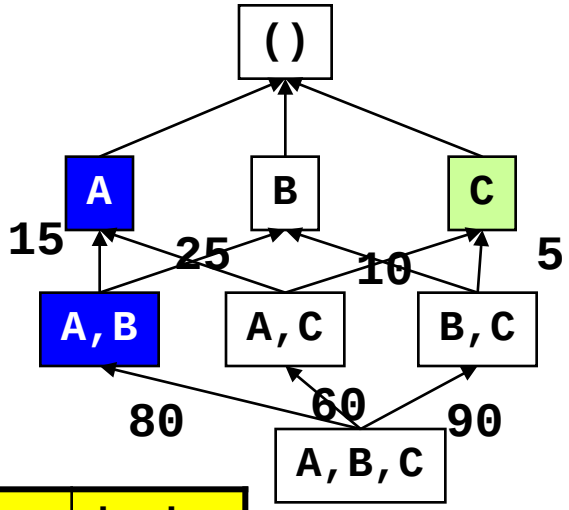


a_i	$\|a_i\|$
()	1
A	24
B	10
C	5
A,B	380
B,C	120
A,C	180
A,B,C	2200

- Annahmen
  - m=200
  - Anfragehäufigkeiten alle 1
  - Kosten zur Berechnung ohne MV: jeweils 200

a_i	B(A)	B(A,B)	B(C)	BS(a_i)
()	0	0	0	0
A	200	0	0	8.3
B	0	0	0	0
C	0	0	200	40
A,B	185	200	0	1
B,C	0	0	195	1.6
A,C	175	0	190	2
A,B,C	115	120	130	0.2

Beispiel 2

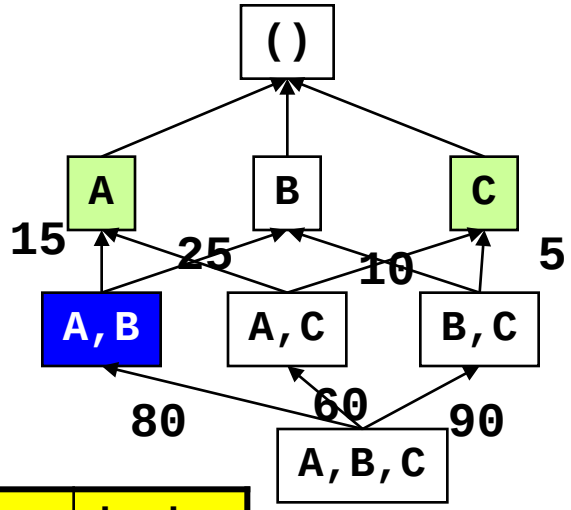


- Annahmen
  - m=200
  - Anfragehäufigkeiten alle 1
  - Kosten zur Berechnung ohne MV: jeweils 200

a_i	$\|a_i\|$
()	1
A	24
B	10
C	5
A,B	380
B,C	120
A,C	180
A,B,C	2200

a_{i1}	B(A)	B(A,B)	B(C)	BS(a_i)
()	0	0	0	0
A	200	0	0	8.3
B	0	0	0	0
A,B	185	200	0	1
B,C	0	0	0	0
A,C	175	0	0	1
A,B,C	115	120	0	0.1

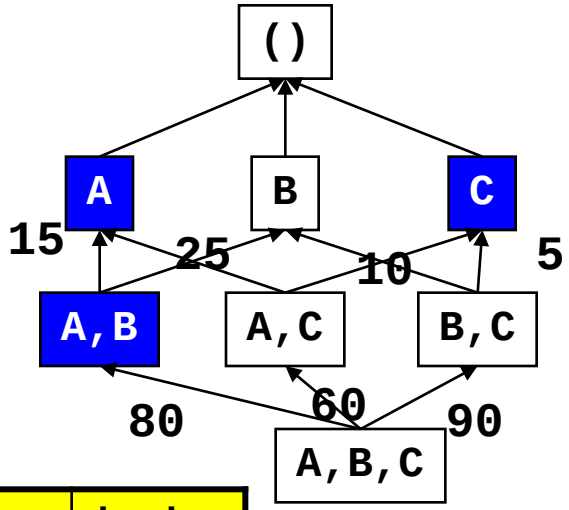
Beispiel 3



a_i	$\|a_i\|$
()	1
A	24
B	10
C	5
A,B	380
B,C	120
A,C	180
A,B,C	2200

- Kein weiterer MV passt in m
  - Und (B,C) bringt nichts
- Kosten für eine Workload
  - Ohne MV: 600
  - Wie berechnet (A und C): 200
  - Das ist hier auch optimal

Gegenbeispiel

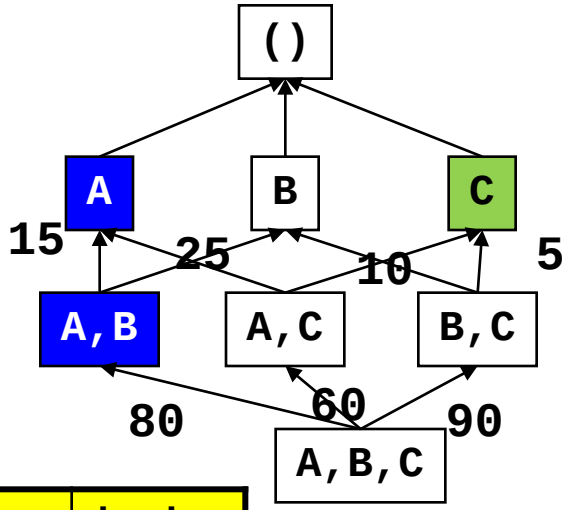


a_i	$\|a_i\|$
()	1
A	24
B	10
C	5
A,B	190
B,C	120
A,C	180
A,B,C	2200

- Annahmen
  - m=200, Anfragefrequenzen alle 1
  - Kosten zur Berechnung ohne MV: A:200, C:130, AB:350

a_i	B(A)	B(A,B)	B(C)	BS(a_i)
()	0	0	0	0
A	200	0	0	8.3
B	0	0	0	0
C	0	0	130	26
A,B	185	350	0	2.8
B,C	0	0	125	1
A,C	175	0	120	1,6
A,B,C	115	270	60	0.2

Gegenbeispiel 2

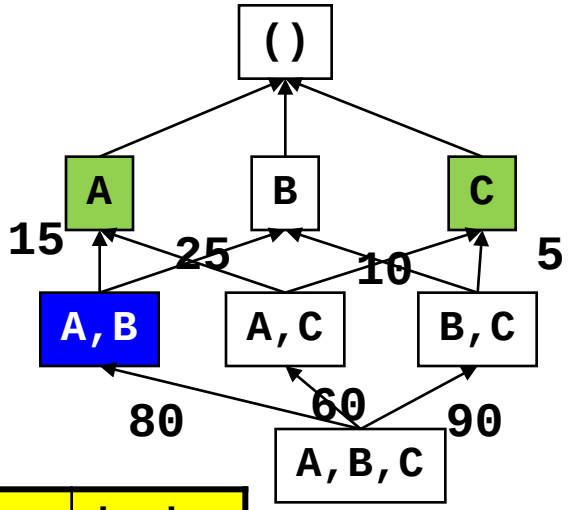


a_i	$\|a_i\|$
()	1
A	24
B	10
C	5
A,B	190
B,C	120
A,C	180
A,B,C	2200

- Annahmen
  - m=200, Anfragefrequenzen alle 1
  - Kosten zur Berechnung ohne MV: A:200, C:130, AB:350

a_i	B(A)	B(A,B)	B(C)	BS(a_i)
()	0	0	0	0
A	200	0	0	8.3
B	0	0	0	0
A,B	185	350	0	2.8
B,C	0	0	125	1
A,C	175	0	120	1,6
A,B,C	115	270	60	0.2

Gegenbeispiel 3



a_i	$\|a_i\|$
()	1
A	24
B	10
C	5
A,B	190
B,C	120
A,C	180
A,B,C	2200

- Kein weiterer guter MV passt in m
  - Es ist noch 171 Platz
  - (B,C) bringt nichts
- Kosten für eine Workload
  - Ohne MV: 680
  - Wie berechnet (A und C): 350
  - Besser wäre (A,B)
    - Platz: 190 (von 200)
    - Kosten der Workload: 330

Verbesserung [SDN98]

- Komplexität des Verfahrens?
  - O(|M|*|A|*|Q|)
  - Wenn man 20 Views für 50 Queries bei 5 Dimensionen a 4 Stufen annimmt: $20*50*2^{20}=1E9$
- Schnellere Methode: Pick-by-size
  - MV nach Größe sortieren (O(|A|*log(|A|)))
  - Greedy MV so lange auswählen, bis m erschöpft ist
  - Liefert unter bestimmten Voraussetzungen gleiche Güte
    - Wenn alle Knoten ihre Tupel relativ gleichmäßig auf alle Nachkommen verteilten
    - Ist z.B. bei statistischer Unabhängigkeit aller Klassifikationsknoten gegeben

Einbeziehung von Aktualisierungskosten

- Die bisherige Kostenfunktion ist monoton

$$B(\{u,v\},Q,M) \leq B(u,Q,M) + B(v,Q,M)$$

- Das garantiert, dass man iterativ vorgehen kann
  - Gilt Monotonie nicht, kann (A,B) zu teuer sein, aber (A,B,C) plötzlich wieder billig
- Ohne Monotonie entfällt die (63-X)% Garantie
- Berücksichtigt man aber Aktualisierungskosten als Kostenmaß, gilt die Monotonie nicht mehr
- Keine ähnlich effektiven Heuristiken bekannt
  - Also: Suchheuristiken verwenden

Literatur

- [Leh03]: 6.1, 6.2 (Konsistenz), 7.4 (Aktualisierung), 7.3 (Auswahl)
- Bello, R. G., Dia, K., Downing, A., Feenen Jr, J., Norcott, W. D., Sun, H., Witkowski, A. and Ziauddin, M. (1998). "Materialized Views in ORACLE". 24th Conference on Very Large Database Systems, New York. pp 659-664.
- Goldstein, J. and Larson, P.-A. (2001). "Optimizing Queries Using Materialized Views: A Practical, Scalable Solution". ACM SIGMOD Int. Conference on Management of Data 1998, Seattle, Washington.
- Zaharioudakis, M., Cochrane, R., Lapis, G., Pirahesh, H. and Urata, M. (2000). "Answering Complex SQL Queries Using Automatic Summary Tables". SIGMOD Conference, Dallas, Texas. pp 105-116.
- Gupta, A. and Mumick, I. S. (1995). "Maintenance of Materialized Views: Problems, Techniques and Applications." *IEEE Quarterly Bulletin on Data Engineering* **18**(2): 3-18.
- Shukla, A., Deshpande, P. and Naughton, J. F. (1998). "Materialized View Selection for Multidimensional Datasets". 24th Conference on Very Large Database Systems, New York. pp 488-499.

Selbsttest

- Wie funktioniert inkrementelle MV Aktualisierung prinzipiell?
- Welche Schwierigkeiten gibt es dabei?
- Wie kann man einen Join-MV ohne Benutzung der Originaltabellen in alter Version aktualisieren?
- Warum sind Deletes schwierig bei der MV Aktualisierung?
- In welcher Reihenfolge sollte man voneinander abhängige MVs aktualisieren?
- Definieren Sie das MV-Auswahl Problem. Wie schwer ist es? Wie kann man das zeigen?
- Wie kann man die Größe eines MVs vor seiner Materialisierung abschätzen?