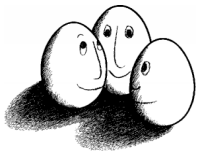
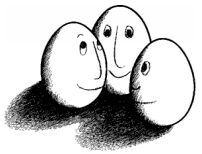


Data Cube

1. Einführung
2. Aggregation in SQL, GROUP BY
3. Probleme mit GROUP BY
4. Der Cube-Operator
5. Implementierung des Data Cube
6. Zusammenfassung und Ausblick

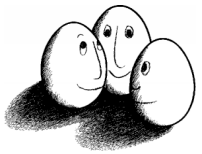
Dank an Hanna Köpcke!



On-line Analytical Processing (OLAP)

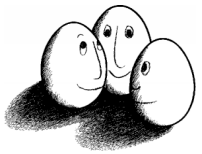
Ziel: Auffinden interessanter Muster in großen Datenmengen

- Formulierung einer Anfrage
- Extraktion der Daten
- Visualisierung der Ergebnisse
- Analyse der Ergebnisse und Formulierung einer neuen Anfrage



Beispiel: Autoverkäufe

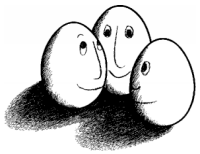
Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Ford	1990	rot	64
Ford	1990	weiß	62
Ford	1990	blau	63
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39



Aggregation in SQL

- **Aggregatfunktionen**:
  COUNT(), SUM(), MIN(), MAX(), AVG()

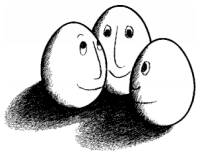
  Beispiel: SELECT **AVG**(Anzahl)
  FROM Autoverkäufe
- Aggregation nur über verschiedene Werte
  Beispiel: SELECT COUNT(**DISTINCT** Modell)
  FROM Autoverkäufe
- Aggregatfunktionen liefern einen einzelnen Wert
- Aggregation über mehrere Attribute mit **GROUP BY**



GROUP BY

SELECT Modell, Jahr, **SUM**(Anzahl)
FROM Autoverkäufe
GROUP BY Modell, Jahr

- Die Tabelle wird gemäß den Kombinationen der ausgewählten Attributmenge in Gruppen unterteilt.
- Jede Gruppe wird über eine Funktion aggregiert.
- Das Resultat ist eine Tabelle mit aggregierten Werten,
- indiziert durch die ausgewählte Attributmenge.

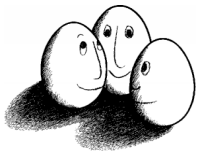


Beispiel: GROUP BY

Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Ford	1990	rot	64
Ford	1990	weiß	62
Ford	1990	blau	63
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39

SELECT Modell, Jahr, **SUM**(Anzahl)
FROM Autoverkäufe
GROUP BY Modell, Jahr

Modell	Jahr	Anzahl
Opel	1990	154
Opel	1991	198
Opel	1992	156
Ford	1990	189
Ford	1991	116
Ford	1992	128



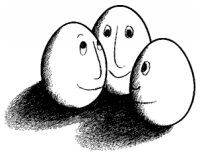
Roll Up

Gleiche Anfrage in unterschiedlichen Detailierungsgraden

- Verminderung des Detailierungsgrades = **Roll Up**
- Erhöhung des Detailierungsgrades = **Drill Down**

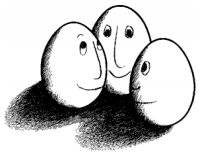
Beispiel: Autoverkäufe

- Roll Up über drei Ebenen
- Daten werden nach Modell, dann nach Jahr, dann nach Farbe aggregiert
- die Verkaufszahlen werden zuerst für jedes Modell aus jedem Jahr in jeder Farbe aufgelistet, dann werden alle Verkaufszahlen des gleichen Modells und Jahres aufsummiert und daraus die Verkaufszahlen der Modelle berechnet



GROUP BY: Roll Up

Modell	Jahr	Farbe	Anzahl nach Modell, Jahr, Farbe	Anzahl nach Modell, Jahr	Anzahl nach Modell
Opel	1990	rot	5		
		weiß	87		
		blau	62		
				154	
	1991	rot	54		
		weiß	95		
		blau	49		
				198	
	1992	rot	31		
		weiß	54		
		blau	71		
				156	
					508

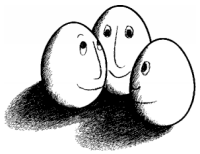


Probleme mit GROUP BY: Roll Up

- Tabelle ist nicht relational, da man wegen der leeren Felder (Null-Werte) keinen Schlüssel festlegen kann.
- Die Zahl der Spalten wächst mit der Zahl der aggregierten Attribute
- Um das exponentielle Anwachsen der Spaltenanzahl zu vermeiden, wird der ALL-Wert eingeführt.
- Der ALL-Wert repräsentiert die Menge, über die die Aggregation berechnet wird.

Beispiel:

Ein ALL in der Spalte Farbe bedeutet, dass in der Anzahl dieser Zeile die Verkaufszahlen der roten, weißen und blauen Autos zusammengefasst sind.

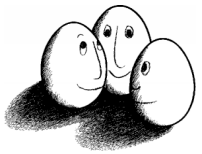


GROUP BY: Roll Up mit ALL

Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1990	ALL	154
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1991	ALL	198
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Opel	1992	ALL	156
Opel	ALL	ALL	506

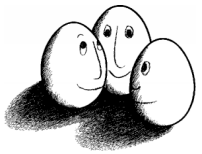
Erzeugung der Tabelle mit SQL:

```
SELECT Modell, ALL, ALL, SUM(Anzahl)
 FROM Autoverkäufe
 WHERE Modell = 'Opel'
 GROUP BY Modell
UNION
 SELECT Modell, Jahr, ALL, SUM(Anzahl)
 FROM Autoverkäufe
 WHERE Modell = 'Opel'
 GROUP BY Modell, Jahr
UNION
 SELECT Modell, Jahr, Farbe,SUM(Anzahl)
 FROM Autoverkäufe
 WHERE Modell = 'Opel'
 GROUP BY Modell, Jahr, Farbe
```



Probleme mit GROUP BY: Roll Up

- Beispiel war ein einfaches dreidimensionales Roll Up
- Eine Aggregation über n Dimensionen erfordert n-1 Unions
- Roll Up ist asymmetrisch:
  Verkäufe sind nach Jahr, aber nicht nach Farbe aggregiert



Kreuztabellen

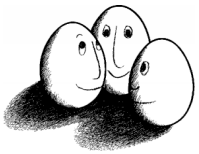
Symmetrische Darstellung mehrdimensionaler Daten und Aggregationen

Opel	1990	1991	1992	Total (ALL)
rot	5	54	31	90
weiß	87	95	54	236
blau	62	49	71	182
Total (ALL)	154	198	156	508

Diese Kreuztabelle ist eine zweidimensionale Aggregation

Nimmt man noch andere Automodelle hinzu, kommt für jedes Modell eine weitere Ebene hinzu

Man erhält eine dreidimensionale Aggregation



Der CUBE-Operator

n-dimensionale Generalisierung der bisher genannten Konzepte

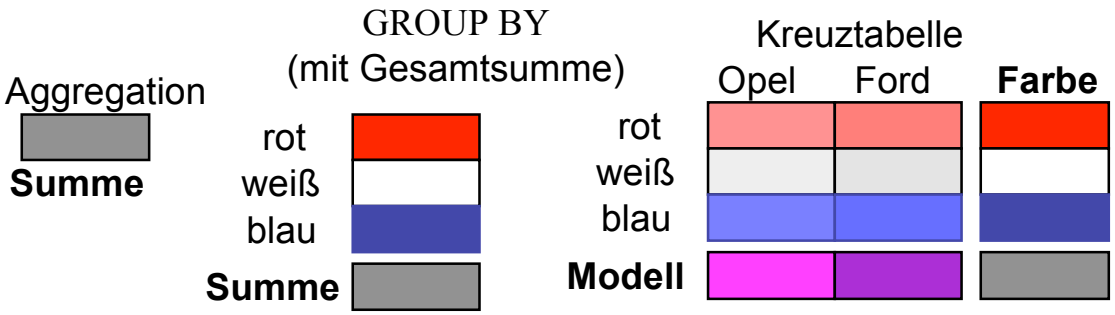
Der 0D Data Cube ist ein Punkt

Der 1D Data Cube ist eine Linie mit einem Punkt

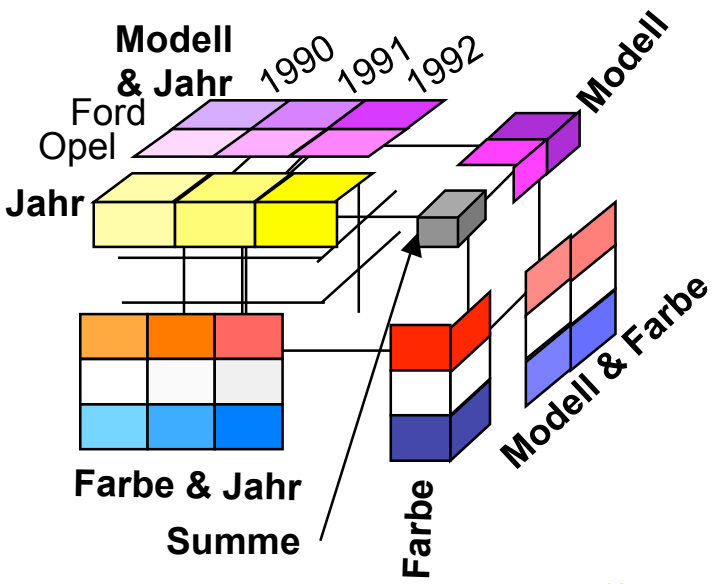
Der 2D Data Cube ist eine Kreuztabelle

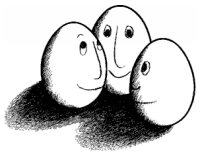
Der 3D Data Cube ist ein Würfel mit drei sich überschneidenden Kreuztabellen

(Gray, Chaudhuri, Bosworth, Layman 1997)



Data Cube mit allen Aggregationen

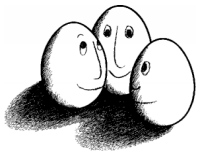




Der CUBE-Operator

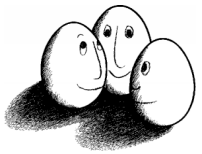
- Beispiel: **SELECT** Modell, Jahr, Farbe, **SUM**(Anzahl)
  FROM Autoverkäufe
  GROUP BY CUBE Modell, Jahr, Farbe

- Der Cube-Operator erzeugt eine Tabelle, die sämtliche Aggregationen enthält
- Es werden GROUP BYs für alle möglichen Kombinationen der Attribute berechnet
- Die Erzeugung der Tabelle erfordert die Generierung der Potenzmenge der zu aggregierenden Spalten.
- Bei n Attributen werden 2^n GROUP BYs berechnet
- Sei C_1, C_2, ..., C_n die Kardinalität der n Attribute, dann ist die Kardinalität der resultierenden Data Cube-Relation $\prod(C_i + 1)$



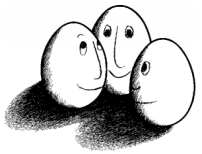
Data Cube des Beispiels

Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Ford	1990	rot	64
Ford	1990	weiß	62
Ford	1990	blau	63
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39



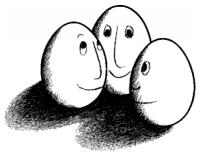
Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1990	ALL	154
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1991	ALL	198
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Opel	1992	ALL	156
Opel	ALL	rot	90
Opel	ALL	weiß	236
Opel	ALL	blau	182
Opel	ALL	ALL	508
Ford	1990	rot	64
Ford	1990	weiß	72
Ford	1990	blau	63
Ford	1990	ALL	189
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1991	ALL	116

Modell	Jahr	Farbe	Anzahl
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39
Ford	1992	ALL	128
Ford	ALL	rot	143
Ford	ALL	weiß	133
Ford	ALL	blau	157
Ford	ALL	ALL	433
ALL	1990	rot	69
ALL	1990	weiß	149
ALL	1990	blau	125
ALL	1990	ALL	343
ALL	1991	rot	106
ALL	1991	weiß	104
ALL	1991	blau	104
ALL	1991	ALL	314
ALL	1992	rot	58
ALL	1992	weiß	116
ALL	1992	blau	110
ALL	1992	ALL	284
ALL	ALL	rot	233
ALL	ALL	weiß	369
ALL	ALL	blau	339
ALL	ALL	ALL	941



Implementationsalternativen

- Physische Materialisierung des gesamten Data Cube:
  - beste Antwortzeit
  - hoher Speicherplatzbedarf
- Keine Materialisierung:
  - jede Zelle wird nur bei Bedarf aus den Rohdaten berechnet
  - kein zusätzlicher Speicherplatz
  - schlechte Antwortzeit
- Materialisierung von Teilen des Data Cube:
  - Werte vieler Zellen sind aus Inhalt anderer Zellen berechenbar
  - diese Zellen nennt man „abhängige“ Zellen
  - Zellen, die einen All-Wert enthalten, sind abhängig
  - Problem: Welche Zellen des Data Cube materialisieren?
  - Zellen des Data Cube entsprechen SQL Anfragen (Sichten)



Abhängigkeit von Sichten

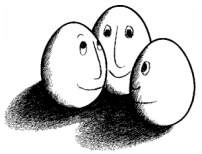
Die Abhängigkeitsrelation $\leq$ zwischen zwei Anfragen Q_1 und Q_2

$Q_1 \leq Q_2$ gdw. Q_1 kann beantwortet werden, indem die Ergebnisse von Q_2 verwendet werden. Q_1 ist abhängig von Q_2

- Anfragen bilden einen Verband unter folgenden Voraussetzungen:
  1. $\leq$ ist eine Halbordnung und
  2. es gibt ein maximales Element (eine oberste Sicht)

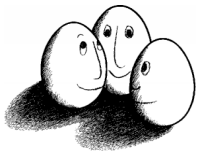
- Der Verband wird durch eine Menge von Anfragen (Sichten) L und der Abhängigkeitsrelation $\leq$ definiert und mit $\langle L, \leq \rangle$ bezeichnet

- Ein Verband wird dargestellt durch einen Graphen, in dem die Anfragen die Knoten sind und $\leq$ die Kanten.



Auswahl von Sichten zur Materialisierung

- *Optimierungsproblem*, das unter folgenden Bedingungen gelöst werden soll:
  - Die durchschnittliche Zeit für die Auswertung der Anfragen soll minimiert werden.
  - Man beschränkt sich auf eine feste Anzahl von Sichten, die materialisiert werden sollen, unabhängig von deren Platzbedarf
- Das Optimierungsproblem ist NP-vollständig.
- Heuristiken für Approximationslösungen: Greedy-Algorithmus
- Der Greedy-Algorithmus verhält sich nie zu schlecht: Man kann zeigen, dass die Güte mindestens 63% beträgt (Harinayaran, Rajaraman, Ullman 1996).



Der Greedy Algorithmus

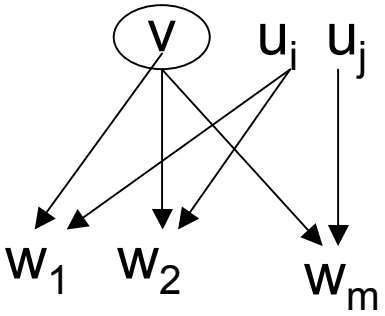
- Gegeben ein Verband mit Speicherkosten C(v) für jede Sicht v
- Annahme: Speicherkosten = Anzahl der Reihen in der Sicht
- Beschränkung auf k materialisierte Sichten
- Nach Auswahl einer Menge S von Sichten wird der Nutzen der Sicht v relativ zu S mit B(v, S) bezeichnet und wie folgt definiert:

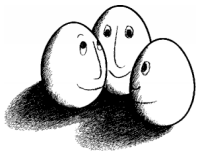
  1. Für jede Sicht $w \leq v$ wird B_w berechnet:

     (a) Sei u die Sicht mit den geringsten Kosten in S, so dass $w \leq u$

     (b) $B_w = \begin{cases} C(u) - C(v), & \text{falls } C(v) < C(u) \\ 0 & \text{ansonsten} \end{cases}$

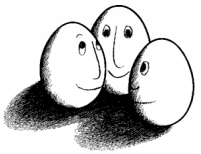
  2. $B(v, S) = \sum_{w \leq v} B_w$



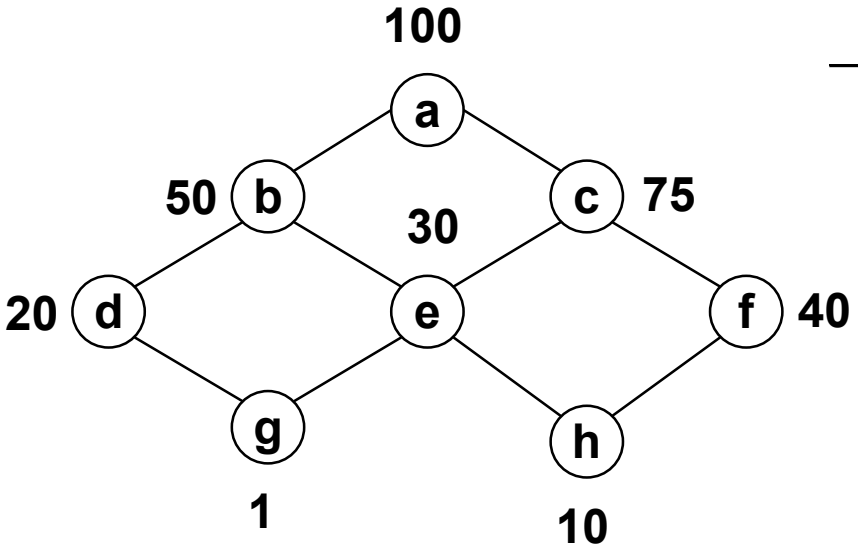


Der Greedy Algorithmus

```
1  S = {oberste Sicht}
2  for i = 1 to k do begin
3        Wähle die Sicht v∉S, so dass B(v, S) maximal ist;
4        S = S ∪ {v}
5        end;
6  return S;
```

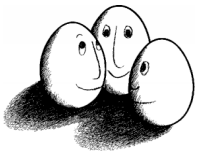


Beispiel

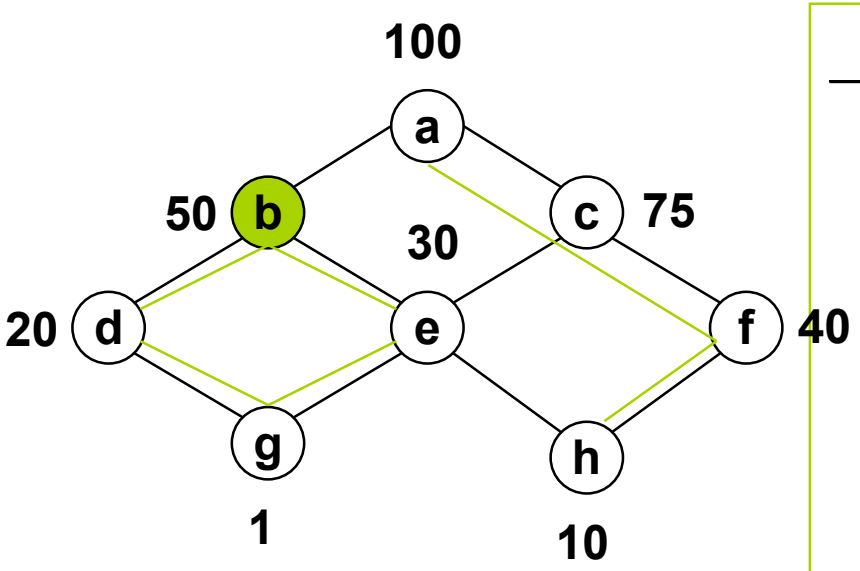


	Erste Wahl	Zweite Wahl	Dritte Wahl
b	50 x 5 = 250		
c	25 x 5 = 125	25 x 2 = 50	25 x 1 = 25
d	80 x 2 = 160	30 x 2 = 60	30 x 2 = 60
e	70 x 3 = 210	20 x 3 = 60	20 + 20 + 10 = 50
f	60 x 2 = 120	60 + 10 = 70	
g	99 x 1 = 99	49 x 1 = 49	49 x 1 = 49
h	90 x 1 = 90	40 x 1 = 40	30 x 1 = 30

S:{a}, S:{a,b}, S:{a,b,f}, S:{a,b,d,f}
Greedy Auswahl: b,d,f werden zusätzlich materialisiert



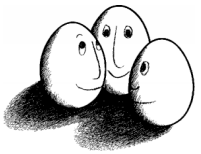
Beispiel



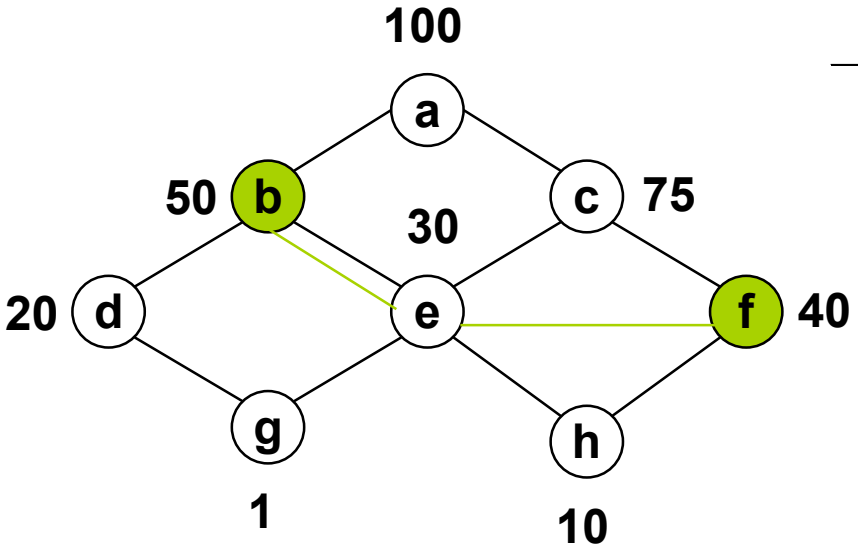
	Erste Wahl	Zweite Wahl	Dritte Wahl
b	50 x 5 = 250		
c	25 x 5 = 125	25 x 2 = 50	25 x 1 = 25
d	80 x 2 = 160	30 x 2 = 60	30 x 2 = 60
e	70 x 3 = 210	20 x 3 = 60	20 + 20 + 10 = 50
f	60 x 2 = 120	60 + 10 = 70	
g	99 x 1 = 99	49 x 1 = 49	49 x 1 = 49
h	90 x 1 = 90	40 x 1 = 40	30 x 1 = 30

S:{a}, S:{a,b}, S:{a,b,f}, S:{a,b,d,f}

Greedy Auswahl: b,d,f werden zusätzlich materialisiert

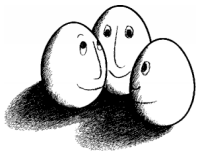


Beispiel



	Erste Wahl	Zweite Wahl	Dritte Wahl
b	50 x 5 = 250		
c	25 x 5 = 125	25 x 2 = 50	25 x 1 = 25
d	80 x 2 = 160	30 x 2 = 60	30 x 2 = 60
e	70 x 3 = 210	20 x 3 = 60	20 + 20 + 10 = 50
f	60 x 2 = 120	60 + 10 = 70	
g	99 x 1 = 99	49 x 1 = 49	49 x 1 = 49
h	90 x 1 = 90	40 x 1 = 40	30 x 1 = 30

S:{a}, S:{a,b}, S:{a,b,f}, S:{a,b,d,f}
Greedy Auswahl: b,d,f werden zusätzlich materialisiert



Was wissen Sie jetzt?

- Möglichkeiten und Grenzen der Aggregation in SQL
- Einführung von Data Cubes zur Unterstützung von Aggregationen über n Dimensionen
- Greedy-Algorithmus zur Auswahl einer festen Anzahl von Sichten, die materialisiert werden