

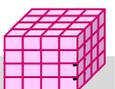
5. Performance-Techniken

- Einleitung
- Indexstrukturen
 - ein- vs. mehrdimensionale Indexstrukturen
 - eindimensionale Einbettungen, UB-Baum
 - Bitlisten-Indexstrukturen
- Datenpartitionierung
 - vertikale vs. horizontale Fragmentierung
 - Projektions-Index
 - mehrdimensionale, hierarchische Fragmentierung
- Materialisierte Sichten
 - Verwendung
 - Auswahl



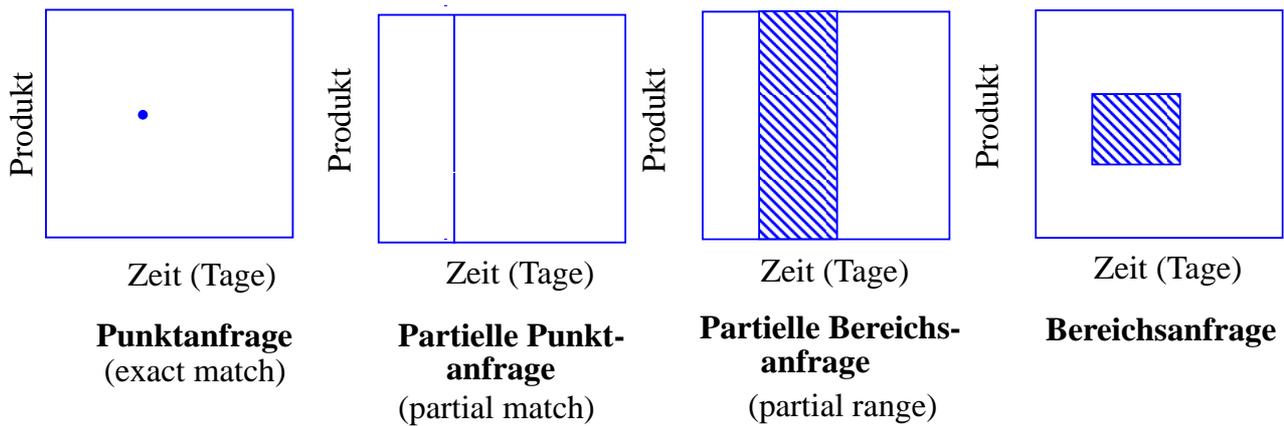
Einleitung

- hohe Leistungsanforderungen
 - sehr große Datenmengen, vor allem Faktentabelle
 - kurze Antwortzeiten für viele Benutzer
 - mehrdimensionale Auswahlbedingungen, Gruppierung, Aggregationen, Sortierung ...
 - periodische Aktualisierung mit sehr vielen Änderungen (ETL, Aktualisierung der DW-Tabellen, Hilfsstrukturen, Cubes)
- Scan-Operationen auf der Faktentabelle i.a. nicht akzeptabel
 - Bsp.: 500 GB, Verarbeitungsgeschwindigkeit 25 MB/s
- Standard-Verfahren (z.B. Hash-Join) oft zu ineffizient für Star Join
- Einsatz mehrerer Performance-Techniken unter spezieller Nutzung von DW-Charakteristika
 - Indexstrukturen (1-dimensional, mehrdimensional, Bit-Indizes)
 - materialisierte Sichten bzw. vorberechnete Aggregationen
 - parallele Anfrageverarbeitung
 - Partitionierung der Daten (Einschränkung der zu bearbeitenden Daten, Parallelverarbeitung)



Mehrdimensionale Anfragearten

■ Punkt- und Bereichsanfragen (exakt vs. partiell)

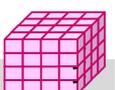


■ Aggregation, Gruppierung (Cube, Rollup), Sortierung ...



Indexstrukturen

- Optimierung selektiver Lesezugriffe: Reduzierung der für Anfrage zu lesenden Datenseiten
- Indexstrukturen enthalten redundante Verwaltungsinformation: zusätzlicher Speicherbedarf und zusätzlicher Änderungsaufwand
- Standard-Indexstruktur: B*-Baum
 - eindimensionaler Index (1 Attribut bzw. Attributkombination)
 - Primär- oder Sekundärindex
 - balanciert, gute Speicherbelegung
 - mit / ohne Clustering der Datensätze
 - geringe Höhe auch bei sehr großen Tabellen
 - Unterstützung von direktem und sortiert-sequentiellen Zugriffen



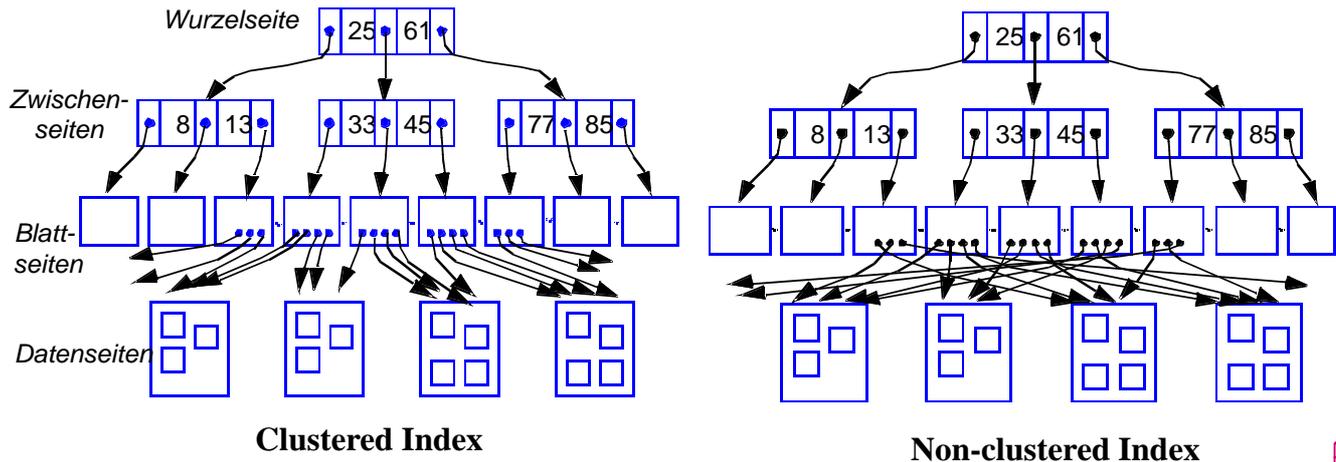
B*-Baum mit/ohne Clusterung der Sätze

■ Index mit Clusterbildung (*clustered index*)

- Clusterbildung der Sätze in den Datenseiten
- Reihenfolge der Sätze gemäß Sortierung nach Indexattribut
- sehr effiziente Bearbeitung von Bereichsanfragen
- maximal 1 Clustered Index pro Relation

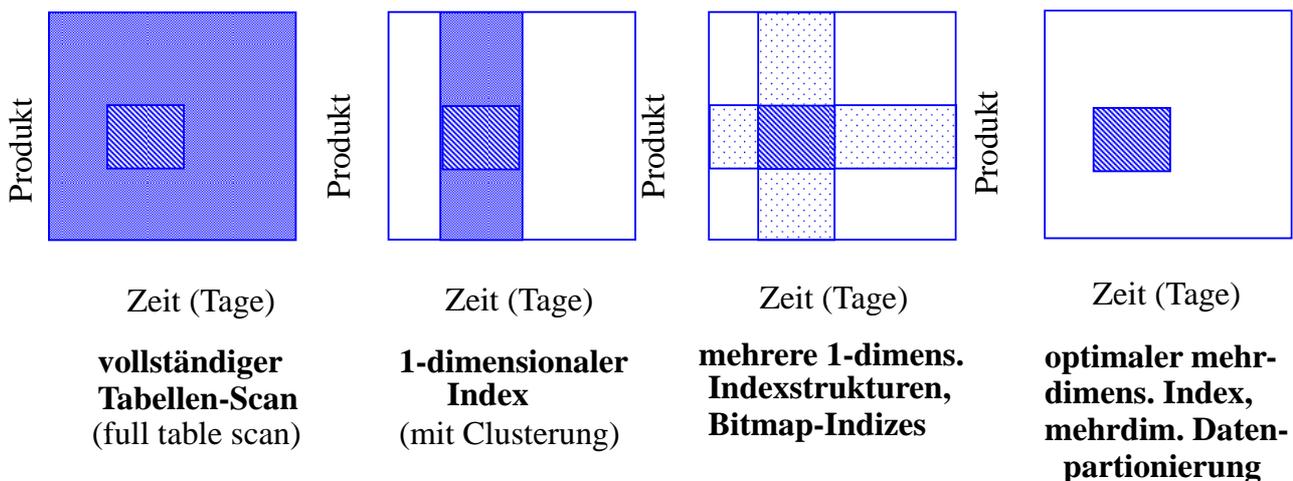
■ Non-clustered Index

- Sätze sind nicht physisch in der Reihenfolge des Indexattributs gespeichert
- gut v.a. für selektive Anfragen und Berechnung von Aggregatfunktionen



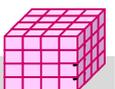
Indexunterstützung für mehrdimens. Anfragen

■ Eingrenzung des Datenraumes



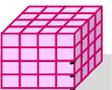
■ Bsp. mehrdimensionaler Indexstrukturen

- Grid File
- R-Baum, ...

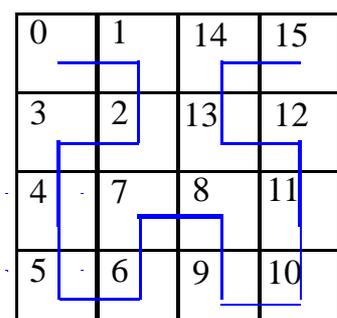
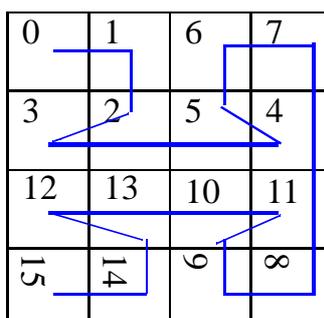
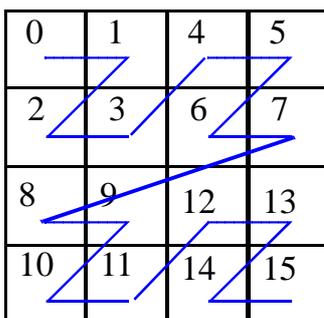
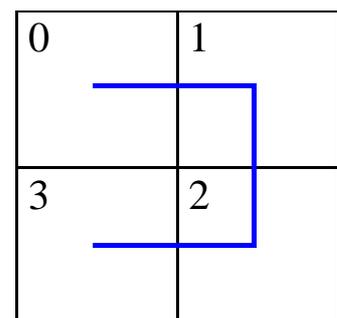
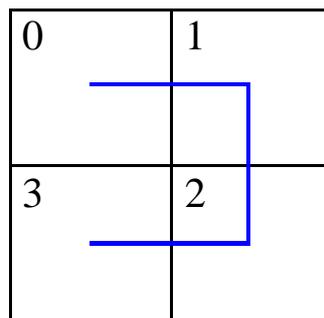
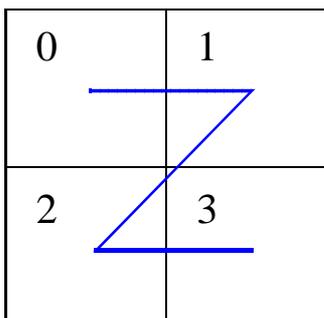


Eindimensionale Einbettung

- Transformation mehrdimensionaler Punktobjekte für eindimensionale Repräsentation, z.B. mit B*-Bäumen
- möglichst Wahrung der topologischen Struktur (Unterstützung mehrdimensionaler Bereichs- und Nachbarschaftsanfragen)
- Ansatz
 - Partitionierung des Datenraums D zunächst durch gleichförmiges Raster
 - eindeutige Nummer pro Zelle legt Position in der totalen Ordnung fest
 - Reihenfolge bestimmt eindimensionale Einbettung: *space filling curve*
- Zuordnung aller mehrdimensionalen Punktobjekte einer Zelle zu einem Bucket (Seite)
- jede Zelle kann bei Bedarf separat (und rekursiv) unter Nutzung desselben Grundmusters weiter verfeinert werden



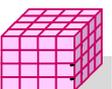
Eindimensionale Einbettungen



a) z-Ordnung

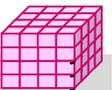
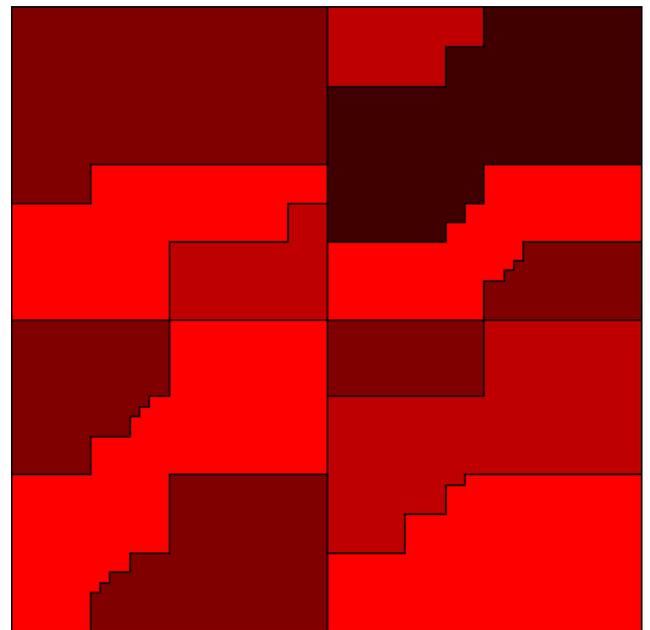
b) Gray-Code

c) Hilbert's Kurve



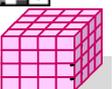
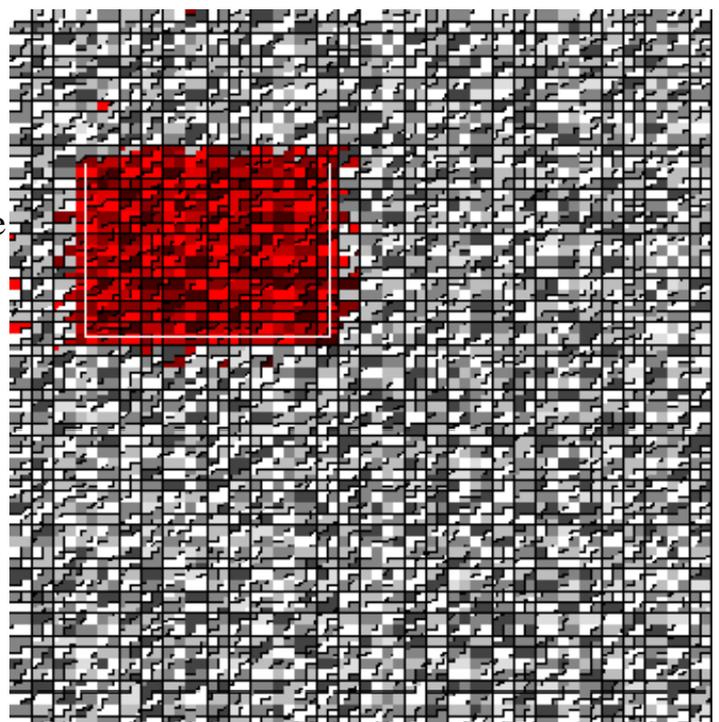
Beispiel: UB-Baum (Universal B-tree)

- Verwendung einer Z-Ordnung als raumfüllende Kurve
- geringer Berechnungsaufwand
 - jeder Punkt wird auf skalaren Wert, **Z-Wert**, abgebildet (Zellen-Nr)
 - binäre Durchnummerierung der Basisintervalle jeder Dimension
 - Z-Wert ergibt sich aus Bit-Verschrankung der Dimensionswerte
- Abbildung der Z-Werte als Schlüssel eines B*-Baum mit Clusterung
- um günstige Auslastung zu gewährleisten, werden mehrere benachbarte Zellen pro Seite zugeordnet: **Z-Region**



UB-Baum (2)

- exakte mehrdimensionale Anfragen gehen auf 1 Seite
- Bereichssuche (Begrenzung durch zwei Eckpunkte LO und RU)
 - bestimme Z-Wert (Z-Region) zu LO
 - werte Suchprädikat auf alle Sätze in Z-Region aus
 - berechne nächsten Bereich der Z-Kurve innerhalb Anfragebereich
 - wiederhole die beiden vorherigen Schritte bis Endadresse der Z-Region größer ist als RU (diesen Punkt also enthält)
- UB-Baum-Realisierung im Rahmen des relationalen DBS TransBase
- Animationen / Publikationen unter <http://mistral.in.tum.de>



Bitlisten-Indizes

■ herkömmliche Indexstrukturen ungeeignet für Suchbedingungen geringer Selektivität

- z.B. für Attribute (Dimensionen) mit nur wenigen Werten (Geschlecht, Farbe, Jahr ...)
- pro Attributwert sehr lange Verweislisten (TID-Listen) auf zugehörige Sätze
- nahezu alle Datenseiten zu lesen

■ Standard-Bitlisten-Index (**Bitmap Index**):

- Index für Attribut A umfasst eine Bitliste (Bitmap, Bitvektor) B_{A_j} für jeden der k Attributwerte $A_1 \dots A_k$
- Bitliste umfasst 1 Bit pro Satz (bei N Sätzen Länge von N Bits)
- Bitwert 1 (0) an Stelle i von B_{A_j} gibt an, dass Satz i Attributwert A_j aufweist (nicht aufweist)
- 1-dimensionaler Index, jedoch flexible Kombinierbarkeit

KID	Geschlecht	Lieblingsfarbe
122	W	Rot
123	M	Rot
124	W	Weiß
125	W	Blau
...

Kunde

Blau 0001100010001100001100000000
Rot 1100000000010010000010000001
Weiß 0010000001100000010000011110
Grün 0000011100000001100001100000 ...



Bitlisten Indizes (2)

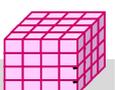
■ Vorteile

- effiziente AND-, OR-, NOT-Verknüpfung zur Auswertung mehrdimensionaler Suchausdrücke
- effiziente Unterstützung von Data-Warehouse-Anfragen (Joins)
- geringer Speicherplatzbedarf bei kleinen Wertebereichen

■ Beispielanfrage

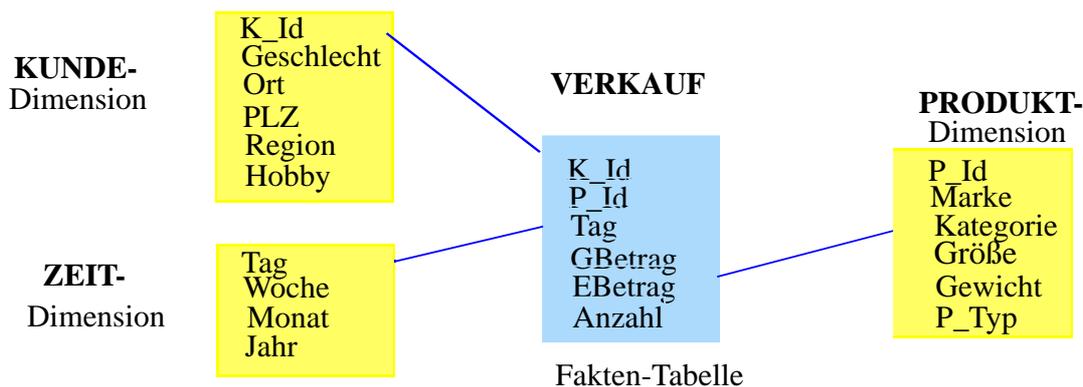
Select ... WHERE A1=C1 AND A2=C2 AND A3=C3
100 Millionen Sätze; pro Teilbedingung Selektivität 1%

■ Erweiterungen erforderlich für Bereichsanfragen, große Wertebereiche, hierarchische Dimensionen



Bitlisten-Join-Index

- Dimensionsbedingungen vor allem im Rahmen von Star Joins
- vollständige Scans auf Fakten-Tabelle zu verhindern -> Nutzung von Bitlisten-Indizes zur Bestimmung der relevanten Fakten-Tupel
- Bitlisten-Join-Index
 - Bitlisten-Index für Dimensions-Attribut auf der Fakten-Tabelle
 - Bitliste enthält Bit pro Fakten-Tupel: entspricht vorberechnetem Join
 - flexible Kombinierbarkeit für unterschiedliche Anfragen
 - Auswertung der Suchbedingungen auf Indizes ermöglicht minimale Anzahl von Datenzugriffen auf die Fakten-Tabelle



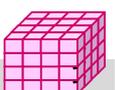
Bitlisten-Join-Index (2)

- Beispielanfrage: Video-Umsatz und Anzahl verkaufter Geräte im April mit männlichen Kunden

```
select sum (GBetrag), sum (Anzahl)
from VERKAUF v, KUNDE k, PRODUKT p, ZEIT z
where v.K_Id = k.K_Id and v.P_Id = p.P_Id and v.Tag = z.Tag
and z.Monat = "April" and p.Kategorie = "Video"
and k.Geschlecht = "M"
```

- Nutzung von 3 Bitlisten-Join-Indizes

Monat	Kategorie	Geschlecht
<i>Januar</i> 0001100010001 ...	<i>TV</i> 1001000000010 ...	<i>M</i> 1010010001001 ...
<i>Febr</i> 0000011100000 ...	<i>DVD</i> 0100000001000 ...	<i>W</i> 0111101110110 ...
<i>März</i> 1000000000010 ...	<i>Video</i> 0010100010101 ...	
<i>April</i> 0110000001100 ...	<i>Stereo</i> 0000011100000 ...	
...	...	



Bereichskodierte Bitlisten-Indizes

- Standard-Bitlisten erfordern für Bereichsanfragen Auswertung vieler Bitlisten
- Bereichskodierte Bitlisten-Indizes:
 - in Bitliste zu Wert w bedeutet 1-Bit für einen Satz, dass der Attributwert *kleiner oder gleich w* ist
 - Bitliste für Maximalwert kann eingespart werden (da nur „1“ gesetzt sind)
- Für jede Bereichsanfrage max. 2 Bitlisten auszuwerten

ID	Monat	Standard-Bitlisten										Bereichskodierte Bitlisten												
		Jan	Feb	Mär	Apr	Mai	Jun	Jul	...	Dez	Jan	Feb	Mär	Apr	Mai	Jun	...	Nov	Dez					
122	Mai	0	0	0	0	1	0	0	...	0														
123	März	0	0	1	0	0	0	0	...	0														
124	Januar	1	0	0	0	0	0	0	...	0														
125	März	0	0	1	0	0	0	0	...	0														
126	Februar	0	1	0	0	0	0	0	...	0														
127	Dezember	0	0	0	0	0	0	0	...	1														
128	April	0	0	0	1	0	0	0	...	0														
...											



Bereichskodierte Bitlisten-Indizes (2)

- Bereichsanfragen: Beispiele
 - Bereich $A < x <- E$:
 - Bereich $x <= E$:
 - Bereich $x > A$:
- Punktanfrage (Gleichheitsbedingung) erfordert Lesen von 2 Bitlisten (vs. 1 Bitliste bei Standard-Bitlisten-Index)
- Speicherplatzreduzierung möglich durch verallgemeinerte *Intervall-Kodierung*



Intervallkodierte Bitlisten-Indizes

- jede Bitliste repräsentiert Wertezugehörigkeit zu bestimmtem Intervall fester Länge von der Hälfte des Gesamtwertebereichs
 - Beispiel I1 = [Jan, Jun], I2 = [Feb, Jul], I3 = [Mär, Aug], I4 = [Apr, Sep], I5 = [Mai, Okt], I6 = [Jun, Nov], I7 = [Jul, Dez]
- für jede Punkt- und Bereichsanfrage max. 2 Bitlisten zu lesen
 - z.B. Bereich März bis September:
Monat = Februar:
- etwa halbiertes Speicheraufwand

ID	Monat	Standard-Bitlisten									Intervallkodierte Bitlisten						
		Jan	Feb	Mär	Apr	Mai	Jun	Jul	...	Dez	I1	I2	I3	I4	I5	I6	I7
122	Mai	0	0	0	0	1	0	0	...	0	1	1	1	1	1	0	0
123	März	0	0	1	0	0	0	0	...	0	1	1	1	0	0	0	0
124	Januar	1	0	0	0	0	0	0	...	0							
125	März	0	0	1	0	0	0	0	...	0							
126	Februar	0	1	0	0	0	0	0	...	0							
127	Dezember	0	0	0	0	0	0	0	...	1							
128	April	0	0	0	1	0	0	0	...	0							
...						

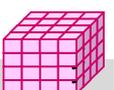


Kodierte Bitlisten-Indizes

- Speicherplatzersparnis durch logarithmische Kodierung der k möglichen Attributwerte (encoded bitmap indexing)
 - Standardverfahren: pro Satz ist nur in einer der k Bitlisten das Bit gesetzt
 - jede der k Wertemöglichkeiten wird durch $\log_2 k$ Bits kodiert \Rightarrow nur noch $\log_2 k$ Bitlisten
 - hohe Einsparungen bei großen k (z.B. Kunden, Produkte)

Blau	0001100010001 ...	Kodierung F ₁ F ₀ Blau Rot Weiß Grün		2 Bitvektoren für 4 Farben F ₁ F ₀
Rot	1100000000010 ...			
Weiß	0010000001100 ...			
Grün	0000011100000 ...			

- Auswertung von Suchausdrücken
 - höherer Aufwand bei nur 1 Bedingung ($\log_2 k$ Bitlisten statt 1 abuarbeiten)
 - bei mehreren Bedingungen wird auch Auswertungsaufwand meist reduziert



Kodierungsvarianten

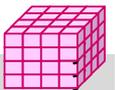
■ Mehrkomponenten-Bit-Indizes

- Zerlegung von Attributwerten in mehrere Komponenten und separate Kodierung
- Wahl der Komponenten erlaubt Kompromiss zwischen Speicheraufwand (# Bitlisten) und Zugriffsaufwand
- Bsp.: Produkt-Nr (0..999) = $x * 100 + y * 10 + z$ mit x,y,z aus 0..9

■ Hierarchische Dimensionskodierung

- Vermeidung separater Bitlisten für jede Dimensionsebene
-> hierarchische Kodierung mit 1 Bitlisten-Index pro Dimension
- Verwendung konkatenierter Schlüssel-IDs und separate Kodierung
- Beispiel: Kodierung einer Produkthierarchie mit ca. 50000 Produkten

	<i>BEREICH</i>	<i>FAMILY</i>	<i>GRUPPE</i>	<i>PNR</i>	Gesamt
#Elemente Gesamt	8	96	1536	49,152	49152
#Elemente pro Vorgänger	8	12	16	32	
#Bits für Kodierung (\log_2)	3	4	4	5	16
Beispiele-Bitmuster	bbb	ffff	gggg	ppppp	bbbffffggggppppp



Datenpartitionierung

■ Partitionierung: logische Zerlegung von Relationen

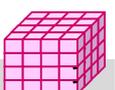
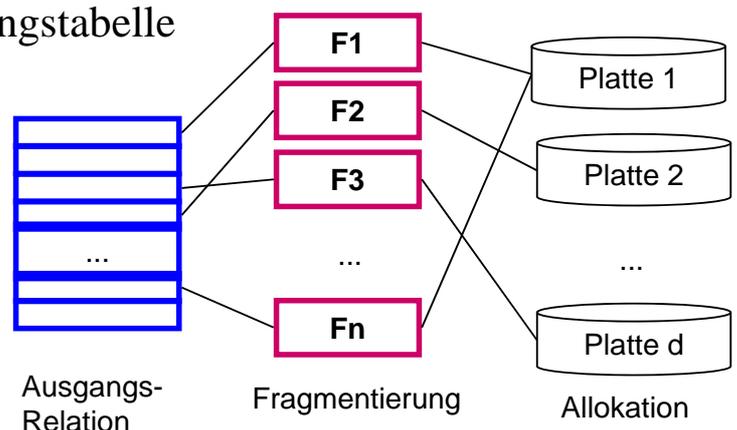
- *Fragmentierung*: Bestimmung der Verteilungseinheiten
- *Allokation*: Zordnung der Fragmente zu Plattenspeichern (Rechnerknoten)

■ Fragmentierung (Zerlegung):

- horizontal vs. vertikal
- Vollständigkeit der Zerlegung
- Rekonstruierbarkeit der Ursprungstabelle

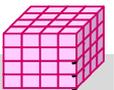
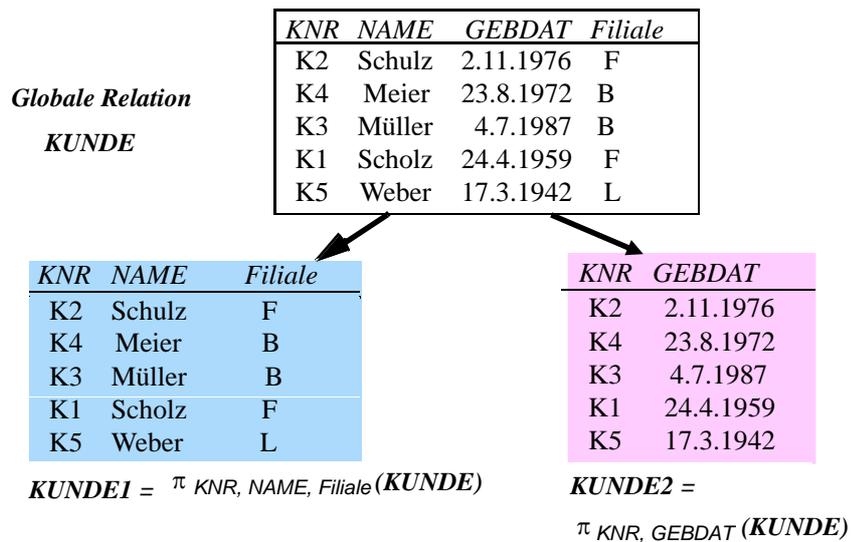
■ Ziele

- Reduzierung des Verarbeitungsumfangs
- Unterstützung von Parallelverarbeitung
- Lastbalancierung



Vertikale Fragmentierung

- Spaltenweise Aufteilung von Relationen
- Definition der Fragmentierung durch Projektion
- Vollständigkeit:
 - jedes Attribut in wenigstens 1 Fragment enthalten
- Verlustfreie Zerlegung:
 - Primärschlüssel i.a. in jedem Fragment enthalten
 - JOIN-Operation zur Rekonstruktion des gesamten Tupels
- Arbeitersparnis durch Auslagern selten benötigter Attribute in eigene Fragmente

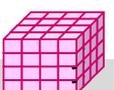


Projektions-Index

- separate Speicherung der Attributwerte ausgewählter Attribute (vertikale Partitionierung)

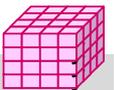
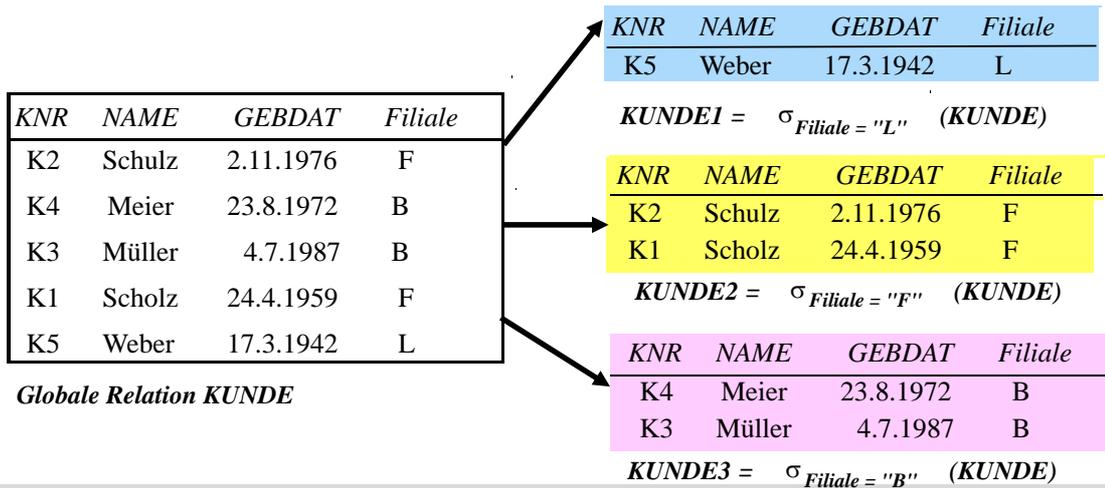
	PNR	ANR	W-ORT	GEHALT	Projektions-Index GEHALT
1	12345	K02	L	45000	45000
2	23456	K02	M	51000	51000
3	34567	K03	L	48000	48000
4	45678	K02	M	55000	55000
5	56789	K03	F	65000	65000
6	67890	K12	L	50000	50000

- starke E/A-Einsparungen verglichen mit Zugriff auf vollständige Sätze
- Bsp.: Berechnung von Durchschnittsgehalt, Umsatzsumme ...
- effektive Einsatzmöglichkeit in Kombination mit Bitlisten-Index-Auswertungen



Horizontale Fragmentierung

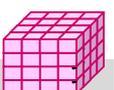
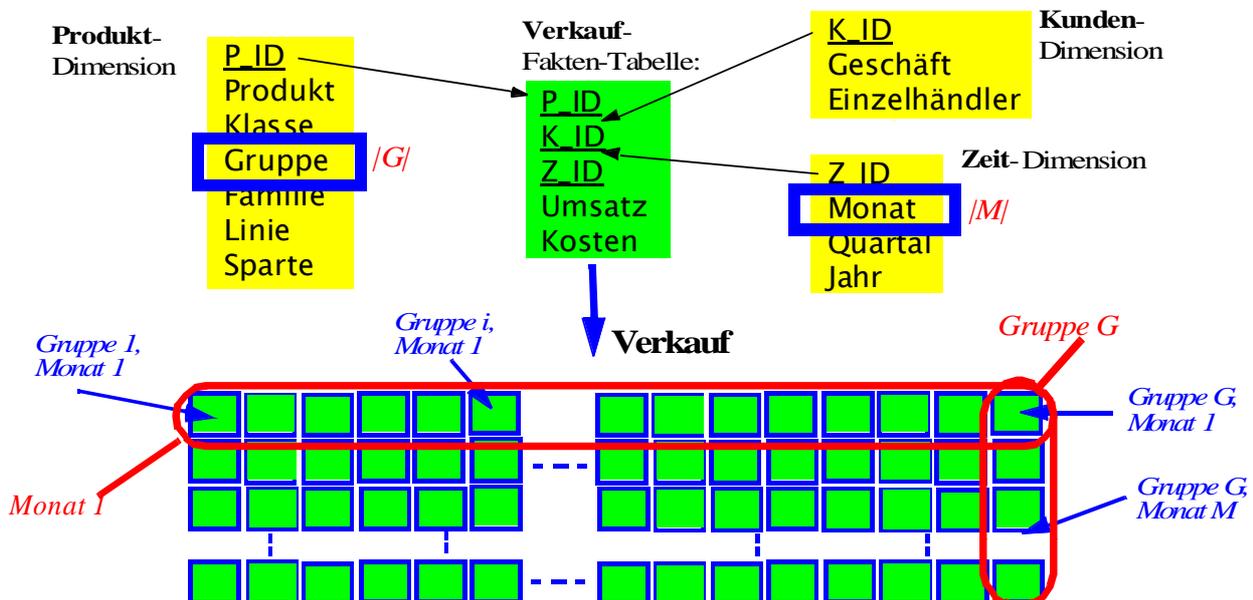
- Zeilenweise Aufteilung von Relationen
- Definition der Fragmentierung durch Selektionsprädikate P_i auf der Relation:
 - $R_i := \sigma_{P_i}(R)$ ($1 \leq i \leq n$)
 - Vollständigkeit: jedes Tupel ist einem Fragment eindeutig zugeordnet
 - Fragmente sind disjunkt: $R_i \cap R_j = \{\}$ ($i \neq j$)
 - Verlustfreiheit: Relation ist Vereinigung aller Fragmente: $R = \cup R_i$ ($1 \leq i \leq n$)
- Anfragen auf Fragmentierungsattribut werden auf Teilmenge der Daten begrenzt
- Parallelverarbeitung unterschiedlicher Fragmente



Multi-dimensionale, hierarchische Fragmentierung (MDHF)

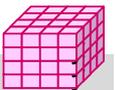
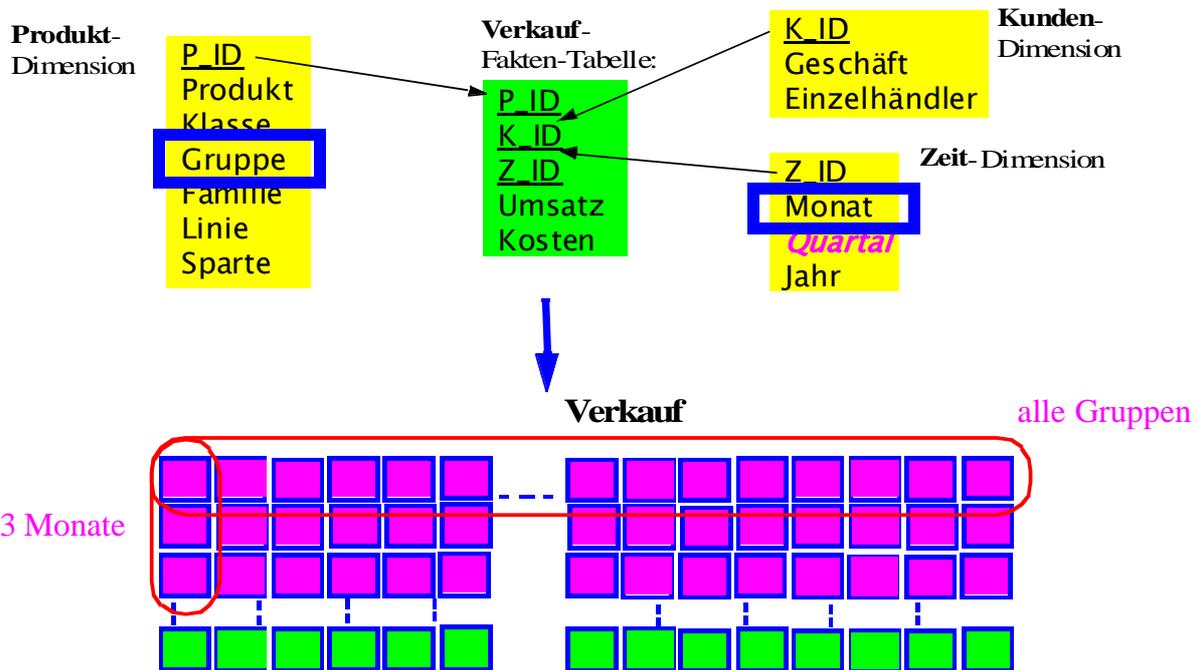
- Horizontale Bereichsfragmentierung auf *mehreren* Attributen (reihenfolge-unabhängig)
 - Auswahl höchstens eines Attributs pro Dimension als Fragmentierungsattribut(e)

Beispiel: 2-dimensionale Fragmentierung $F_{GruppeMonat}$ der Faktentabelle **Verkauf**



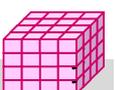
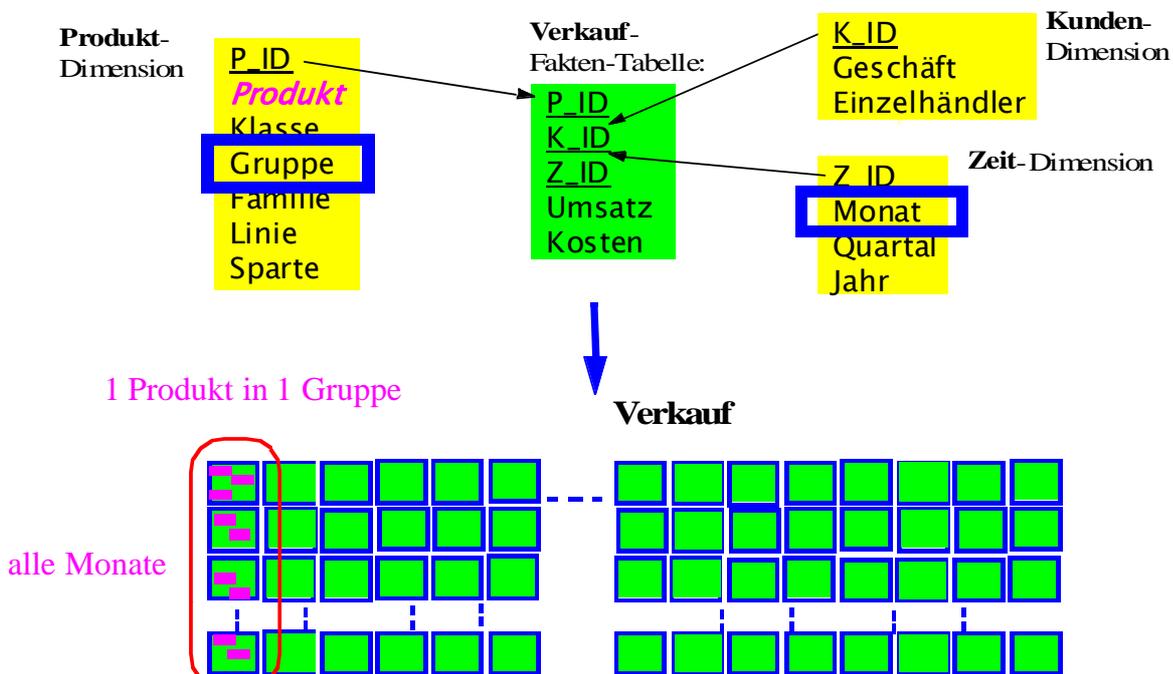
Beispiel: Sternschema-Anfrage

- Zugriff oberhalb einer Fragmentierungsebene (Anfrage auf *Quartal*)
- Clusterung der Treffer, keine Indexzugriffe notwendig



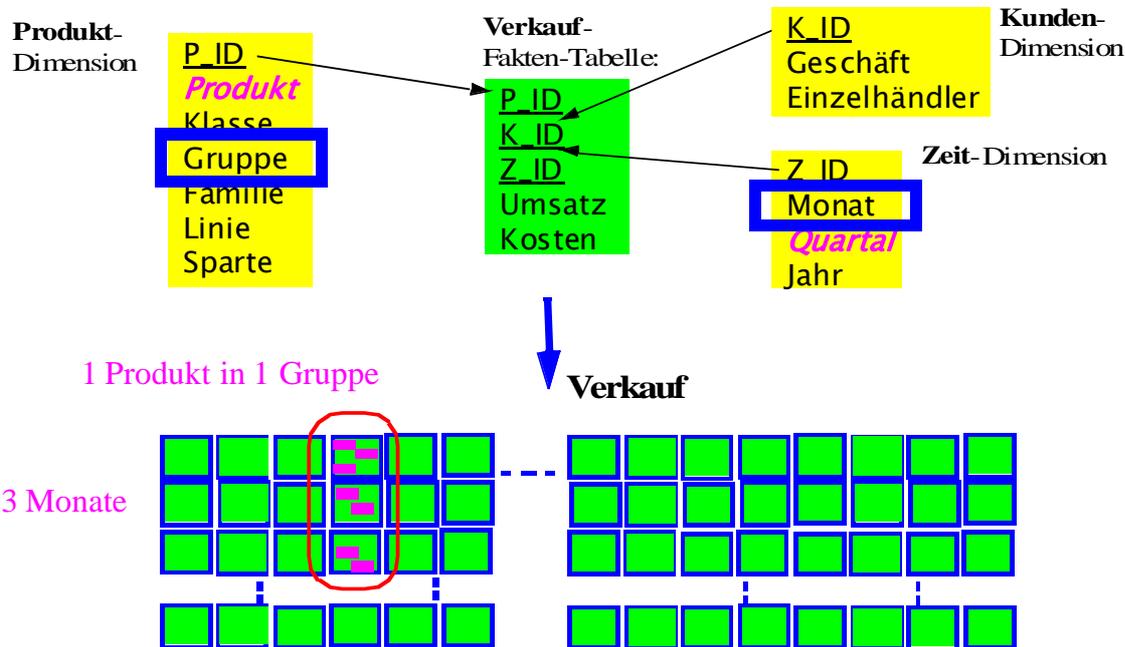
Sternschema-Anfrage unter MDHF (2)

- Zugriff unterhalb einer Fragmentierungsebene (Anfrage auf *Produkt*)
- Einschränkung der Fragmente, ggf. Index-Zugriff zum Auffinden des Produktes im Fragment



Sternschema-Anfrage unter MDHF (3)

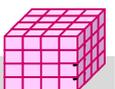
- Zugriff oberhalb und unterhalb der Fragmentierungsebene (Anfrage auf *Quartal* und *Produkt*)
- nur 3 Fragmente, ggf. Index-Zugriff zum Auffinden des Produktes im Fragment



Eigenschaften von MDHF*

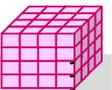
- Reduktion des E/A-Aufwandes für viele Warehouse-Anfragen
 - durch Bezug auf *mehrere* Dimensionen
 - schon bei mindestens 1 referenzierter Fragmentierungsdimension
 - Anfrageattribute müssen nicht mit Fragmentierungsattributen übereinstimmen
- Einsparung von Bitlisten-Indizes
 - Materialisierung von Indizes nur für Ebenen *unterhalb* der Fragmentierungsebene(n)
- Unterstützung von Parallelität bei fragment-orientierter Verarbeitung
- Analytische Optimierung / Tuning
 - analytische Formeln für #Fragmente, #Bitlisten-Zugriffe, I/O-Umfang, Antwortzeiten etc. für gegebenen Query-Mix, DB-Schema und Allokation
 - Bestimmung der Top-Fragmentierungen bezüglich Antwortzeit und I/O-Umfang
 - Tool **WARLOCK** (*Warehouse Allocation To Disk*):

* Stöhr, T., Märtens, H., Rahm, E.: *Multi-Dimensional Database Allocation for Parallel Data Warehouses* Proc. VLDB, 2000, Stöhr, T., Rahm, E.: *Warlock: A Data Allocation Tool for Parallel Warehouses*. Proc. VLDB, 2001 (software demo)



Materialisierte Sichten

- Unterstützung in kommerziellen DBS: Oracle, DB2, SQL Server
 - materialized views, summary tables, ...
- explizite Speicherung von Anfrageergebnissen, z.B. Aggregationen, zur Beschleunigung von Anfragen
- sehr effektive Optimierung für Data Warehousing
 - häufig ähnliche Anfragen (Star Queries)
 - Lokalität bei Auswertungen
 - relativ stabiler Datenbestand
- Realisierungs-Aspekte
 - Verwendung von materialisierten Sichten für Anfragen (Query-Umformulierung, query rewrite)
 - Auswahl der zu materialisierenden Sichten: statisch vs. dynamisch (Caching von Anfrageergebnissen)
 - Aktualisierung materialisierter Sichten



Verwendung materialisierter Sichten

- Einsatz von materialisierter Sichten transparent für den Benutzer
 - DBS muss während Anfrageoptimierung relevante materialisierte Sichten automatisch erkennen und verwenden können (Anfrageumstrukturierung)
 - Umgeformte Anfrage muss äquivalent zur ursprünglichen sein (dasselbe Ergebnis liefern)

■ Beispiel

Anfrage Q

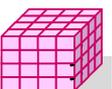
```
select sum (v.GBetrag)
from VERKAUF v, PRODUKT p, ZEIT z
where v.Tag = z.Tag and v.P_Id = p.P_Id
and z.Monat = "Jun04" and
p.Kategorie = "Video"
```

modifizierte Anfrage Q'

```
select
```

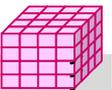
mat. Sicht M1

```
create materialized view M1 (K, M, S, A) AS
select P.Kategorie, Z.Monat,
       SUM (GBetrag), SUM (Anzahl)
from VERKAUF v, PRODUKT p, ZEIT z
where v.Tag = z.Tag and v.P_Id = p.P_Id
group by cube (p.Kategorie, z.Monat)
```



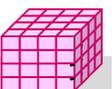
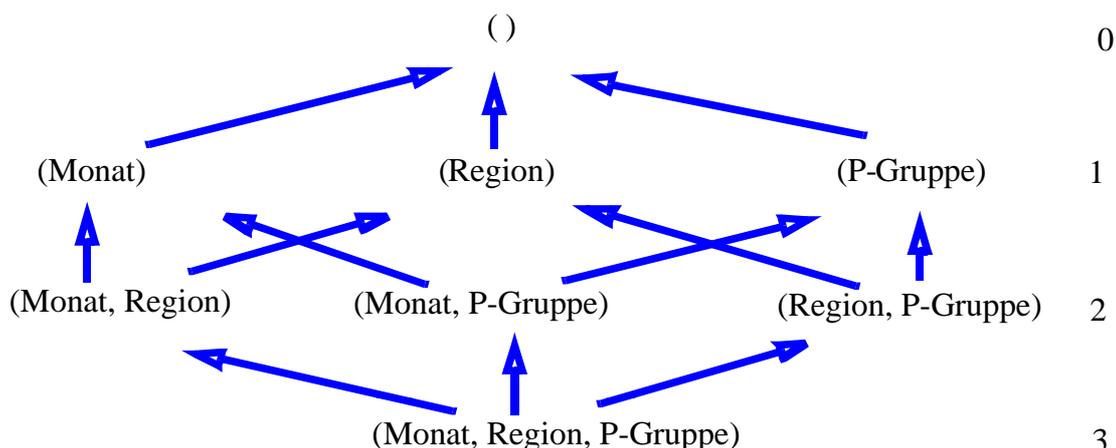
Auswahl von materialisierter Sichten

- Optimierungs-Tradeoff: Nutzen für Anfragen vs. erhöhten Speicherungs- und Aktualisierungskosten
- statische Bestimmung durch DBA oder Tool:
 - keine Berücksichtigung aktueller Anfragen
 - keine Änderung bis zur nächsten Warehouse-Aktualisierung
- dynamische Auswahl: Caching von Anfrageergebnissen (semantisches Caching)
 - Nutzung von Lokalität bei Ad-Hoc-Anfragen
 - günstig bei interaktiven Anfragen, die aufeinander aufbauen (z.B. Rollup)
- Komplexe Verdrängungsentscheidung für variabel große Ergebnismengen unter Berücksichtigung von
 - Zeit des letzten Zugriffs, Referenzierungshäufigkeit
 - Größe der materialisierten Sicht
 - Kosten, die Neuberechnung verursachen würde
 - Anzahl der Anfragen, die mit Sicht bedient wurden



Statische Auswahl materialisierter Sichten

- Auswahl vorzuberechnender Aggregationen des Aggregationsgitters
 - *Aggregationsgitter*: azyklischer Abhängigkeitsgraph, der anzeigt, für welche Kombinationen aus Gruppierungsattributen sich Aggregierungsfunktionen (SUM, COUNT, MAX, ...) direkt oder indirekt aus anderen ableiten lassen
- vollständige Materialisierung aller Kombinationen i.a. nicht möglich
 - #Gruppierungskombinationen wächst exponentiell mit Anzahl von Gruppierungsattributen n
 - möglichst optimale Teilmenge zu bestimmen



Statische Auswahlheuristik*

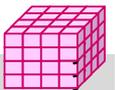
■ Annahmen

- Gleiche Nutzungswahrscheinlichkeit pro Cuboid (Kombination von Dimensionsattributen)
- Aufwand sei proportional zu Anzahl zu berechnender Sätze/Aggregate

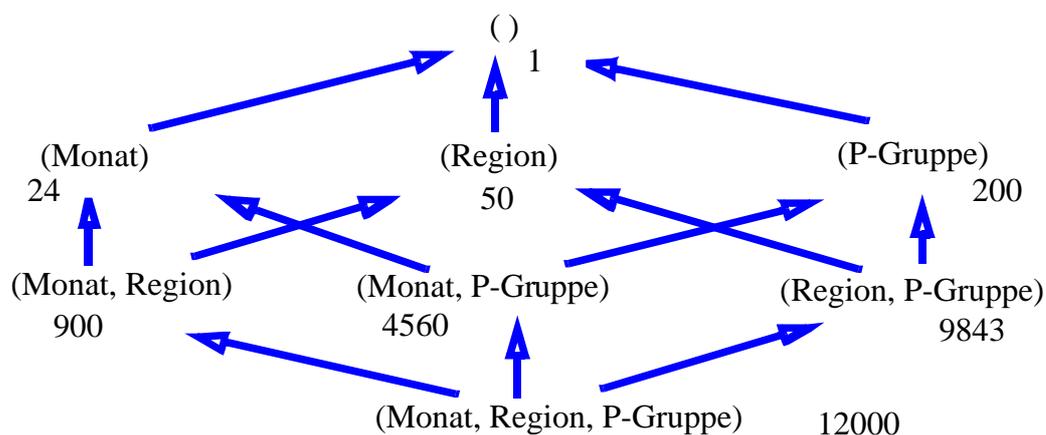
■ Heuristik für vorgegebenes Limit für Speicheraufwand

- pro Kombination von Dimensionsattributen wird Summe der Einsparungen berechnet, die sie für andere nicht materialisierte Kombinationen bewirkt
- in jedem Schritt wird die Kombination ausgewählt, die die größte Summe an Einsparungen zulässt, solange der maximal zugelassene Speicheraufwand nicht überschritten ist

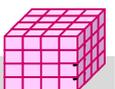
*Harinarayan/Rajaraman/Ullman, *Implementing Data Cubes Efficiently*. Proc. Sigmod 1996



Statische Auswahlheuristik: Beispiel



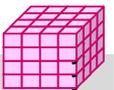
- Limit sei 75% Zusatzaufwand (bezogen auf Kardinalität der Detaildaten)
 - Vollauswertung erfordert $12.000 + 15.578 = 27.578$ Sätze (+ 130%) -> Beschränkung
- Schritt 1: maximale Einsparung für Monat/Region
 - nur 900 statt 12.000 Werte auszuwerten
 - Nutzung für 4 Knoten (Einsparung $4 * 11.100$): (Monat, Region), (Monat), (Region), ()
- Schritt 2:



Statische Auswahl materialisierter Sichten (4)

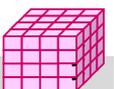
■ Berücksichtigung weiterer Faktoren

- Eliminierung aller Knoten, für die Verdichtung bestimmten Schwellwert (z.B. 50%) nicht erreicht
- funktionale Abhängigkeiten zwischen Attributen (z.B. innerhalb Dimensionshierarchie) erlauben Eliminierung von Kombinationen, z.B.:
Gruppierung (Artikel, Produktgruppe) liefert identische Werte wie Gruppierung (Artikel)
- Berücksichtigung nur solcher Knoten, welche vorgegebene Anfragen unterstützen



Zusammenfassung

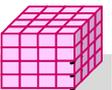
- Mehrdimensionale vs. 1-dimensionale Indexstrukturen
- UB-Baum: Abbildung mehrdimensionaler Wertekombinationen auf eindimensionale Reihenfolge (Z-Kurve)
- Bit-Indizes
 - effizient kombinierbar für mehrdimensionale Auswertungen und Star-Joins
 - Bereichs- und Intervallkodierung für Bereichsanfragen
 - Kodierung für höhere Kardinalität: logarithmisch, Mehr-Komponenten-Kodierung, hierarchische Kodierung
- Partitionierung
 - vertikale oder horizontale Zerlegung von Relationen zur Reduzierung des Arbeitsaufwandes und Unterstützung von Parallelverarbeitung
 - Projektions-Index Spezialfall vertikaler Partitionierung
 - mehrdimens. horizontale Fragmentierung: ähnliche Vorteile wie mehrdim. Indexstrukturen



Zusammenfassung (2)

■ Materialisierte Sichten

- große Performance-Vorteile durch Vorberechnung von Anfragen/Aggregationen
- transparente Umformulierung von Anfragen unter Verwendung der materialisierten Sichten
- dynamische vs. statische Auswahl an materialisierten Sichten



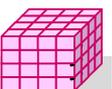
Übungsfragen

■ Indexstrukturen

- Welche Anfragen lassen sich durch UB-Bäume effizienter als über konventionelle B*-Bäumen mit Clusterung beantworten?
- Welche Anfragearten können durch Standard-B*-Bäume effektiver als mit Bitlisten-Indizes unterstützt werden?

■ Bitlisten-Indizes

- Bestimmen Sie für die Beispiele bereichskodierter bzw. intervallkodierter Bitlisten-Indizes auf Folien 5-15 bzw. 5-17 wie folgende Anfrageprädikate evaluiert werden können:
 - Monat – “Januar“
 - Monat IN („Mai“, “Juni“, “Juli“)



Übungsfragen (2)

■ Auswahl materialisierter Sichten

Welche drei Materialisierungen bringen für die im Aggregationsgitter angegebenen Kardinalitäten den größten Einspareffekt gemäß dem Verfahren von Harinarayan et al. ?

Welcher Zusatzaufwand wird durch sie gegenüber der Speicherung von (A,B,C) eingeführt?

