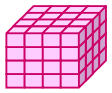
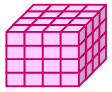
5. Performance-Techniken

- 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



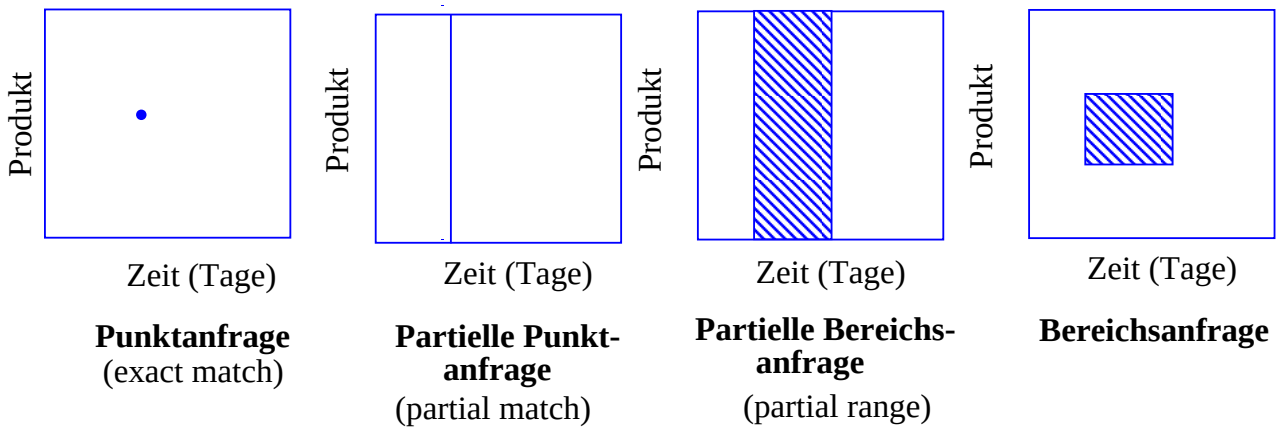
Einleitung

- hohe Leistungsanforderungen
  - sehr große Datenmengen, vor allem Faktentabelle
  - kurze Antwortzeiten für viele Benutzer
  - umfangreiche mehrdimensionale Auswahlbedingungen (Restriktionen) und Aggregationen
  - Gruppierung, 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-Join-Verfahren (z.B. Hash-Join) oft zu ineffizient für Star Join
- Einsatz mehrerer Performance-Techniken unter spezieller Nutzung von DW-Charaktistika
  - Indexstrukturen (1-dimensional, mehrdimensional, Bit-Indizes)
  - materialisierte Sichten bzw. vorberechnete Anfrageergebnisse / Aggregationen
  - parallele Anfrageverarbeitung
  - Partitionierung der Daten (Einschränkung der zu bearbeitenden Daten, Parallelverarbeitung)

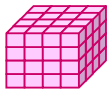


Mehrdimensionale Anfragearten

- Punkt- und Bereichsanfragen (exakt vs. partiell)

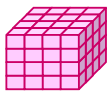


- Aggregation, Gruppierung (Cube, Rollup), Sortierung ...



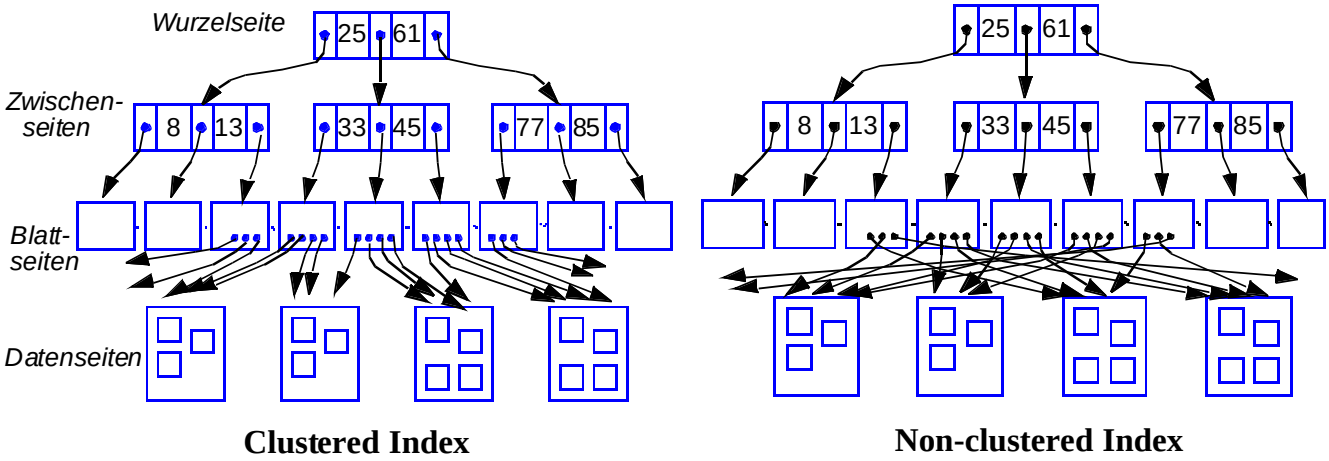
Indexstrukturen

- 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)
  - balanciert, gute Speicherbelegung
  - mit / ohne Clusterung der Datensätze
  - geringe Höhe auch bei sehr großen Tabellen
  - Unterstützung von direktem und sortiert-sequentiellen Zugriffen
  - Primär- oder Sekundärindex



B*-Baum mit/ohne Clusterung der Sätze

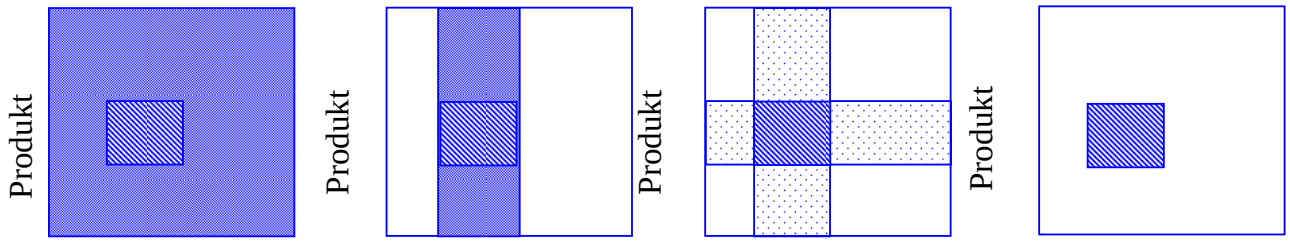
- Index mit Clusterbildung (clustered index)
  - Clusterbildung der Sätze in den Datenseiten
  - Reihenfolge der Sätze gemäß Sortierung nach Indexattribut
  - sehr effiziente Bearbeitung von Bereichsanfragen
  - maximal 1 Clustered Index pro Relation
- Non-clustered Index
  - Sätze sind nicht physisch in der Reihenfolge des Indexattributs gespeichert
  - gut v.a. für selektive Anfragen und Berechnung von Aggregatfunktionen



Clustered Index **Non-clustered Index**

Indexunterstützung für mehrdimens. Anfragen

- Eingrenzung des Datenraumes



vollständiger Tabellen-Scan (full table scan)

1-dimensionaler Index (mit Clusterung)

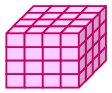
mehrere 1-dimens. Indexstrukturen, Bitmap-Indizes

optimaler mehr-dimens. Index, mehrdim. Daten-partionierung

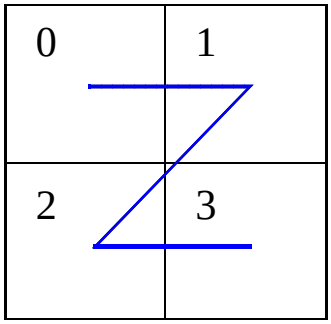
- Bsp. mehrdimensionaler Indexstrukturen
  - Grid File
  - R-Baum, ...

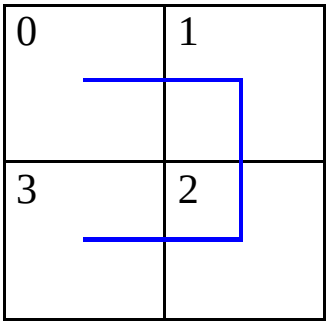
Eindimensionale Einbettung

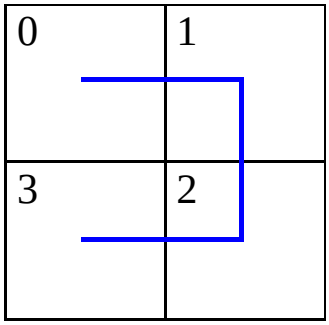
- Transformation mehrdimensionaler Punktobjekte für eindimensionale Repräsentation, z.B. mit B*-Bäumen
- möglichst Wahrung der topologischen Struktur (Unterstützung mehrdimensionaler Bereichs- und Nachbarschaftsanfragen)
- Ansatz
  - Partitionierung des Datenraums D zunächst durch gleichförmiges Raster
  - 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

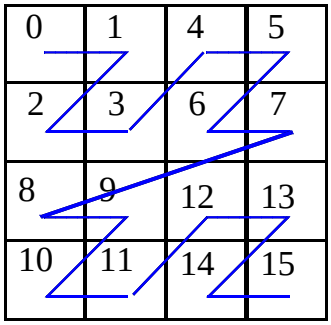


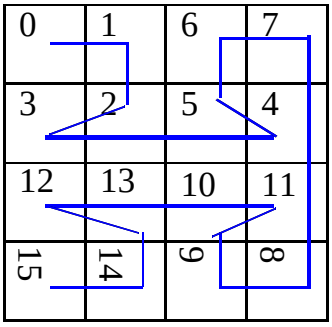
Eindimensionale Einbettungen

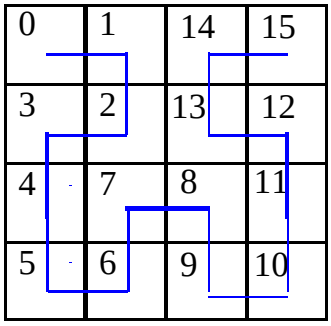








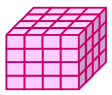




a) z-Ordnung

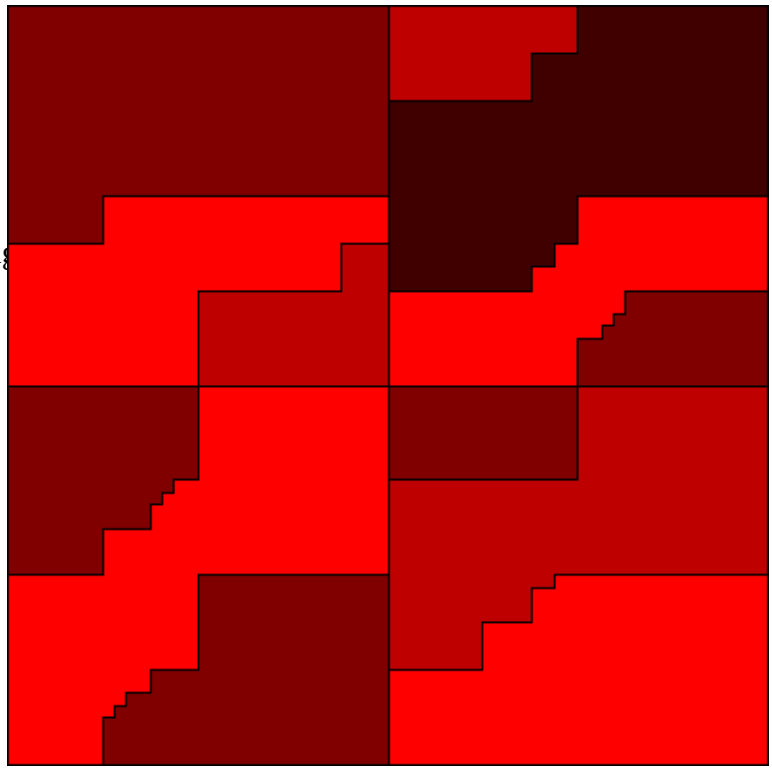
b) Gray-Code

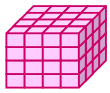
c) Hilbert‘s Kurve



Beispiel: UB-Baum (Universal B-tree)

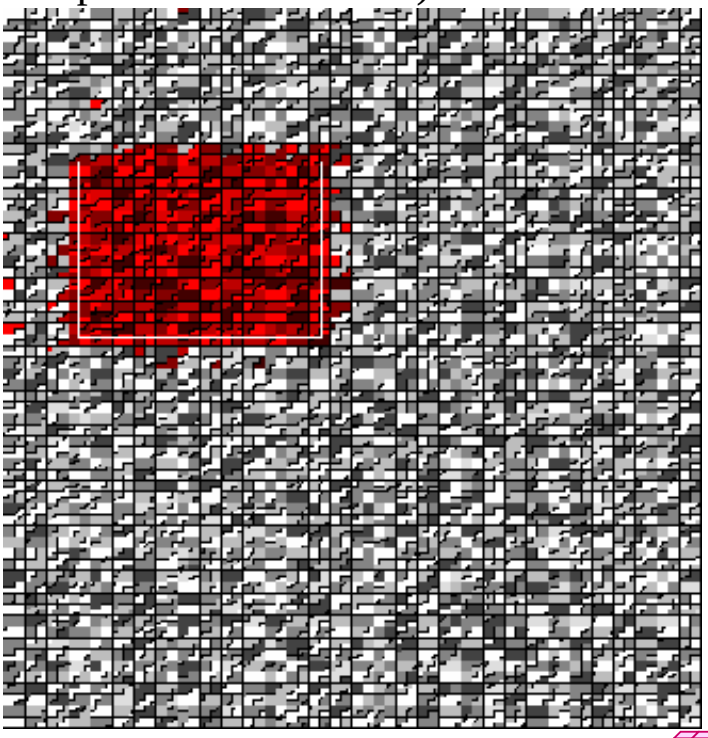
- Verwendung einer Z-Ordnung als raumfüllende Kurve
- geringer Berechnungsaufwand
  - jeder Punkt wird auf skalaren Wert, Z-Wert, abgebildet (Zellen-Nr)
  - binäre Durchnumerierung der Basisintervalle jeder Dimension
  - Z-Wert ergibt sich aus Bit-Verschränkung der Dimensionswerte
- Abbildung der Z-Werte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

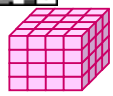




UB-Baum (2)

- exakte mehrdimensionale Anfragen gehen auf 1 Seite
- Bereichssuche (Begrenzung durch 2 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





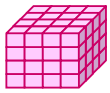
Bitlisten-Indizes

- eindimensionale Indexstrukturen
  - separate Indexierung pro Attribut / Dimension
  - flexible und effiziente Kombination im Rahmen mehrdimensionaler Anfragen
- herkömmliche Indexstrukturen ungeeignet für Suchbedingungen geringer Selektivität
  - z.B. für Attribute (Dimensionen) mit nur wenigen Werten (Geschlecht, Farbe, Region, Jahr ...)
  - pro Attributwert sehr lange Verweislisten auf zugehörige Sätze (TIDs = Tuple Identifier)
  - nahezu alle Datenseiten zu lesen
- Standard-Bitlisten-Index (Bitmap Index):
  - Index für Attribut A umfasst eine Bitliste (Bitmap, Bitvektor) B_{Aj} für jeden der k Attributwerte A1 ... Ak
  - Bitliste umfasst 1 Bit pro Satz (bei n Sätzen Länge von n Bits)
  - Bitwert 1 (0) an Stelle i von B_{Aj} gibt an, dass Satz i Attributwert Aj aufweist (nicht aufweist)

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

Kunde

Blau	0001100010001100001100000000001001000
Rot	1100000000010010000001000000110000001
Weiß	0010000001100000010000001110000000110
Grün	0000011100000001100001100010001100000...

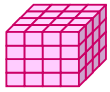


Bitlisten-Indizes (2)

- Vorteile
  - geringer Speicherplatzbedarf bei kleinen Wertebereichen
  - effiziente AND-, OR-, NOT-Verknüpfung zur Auswertung von mehrdimensionalen Suchausdrücken
  - effiziente Unterstützung von Star Joins
- Beispielanfrage

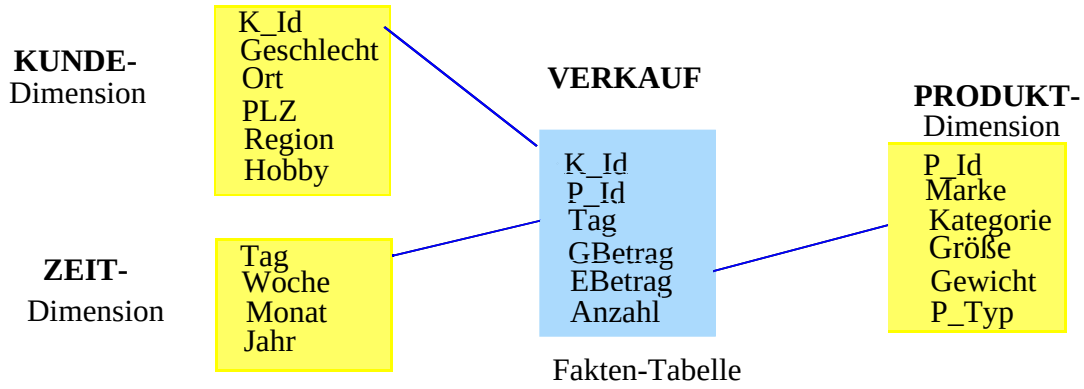
  Select ...
  WHERE A1 = C1 AND A2=C2 AND A3=C3 AND A4=C4

  10 Millionen Sätze; pro Teilbedingung Selektivität 5%
- Nachteil: hoher Änderungsaufwand
- 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 2004 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 = "Apr04" and p.Kategorie = "Video"
                    and k.Geschlecht = "M"
```

- Nutzung von 3 Bitlisten-Join-Indizes

Monat

Jan04 0001100010001 ...
Feb04 0000011100000 ...
Mär04 1000000000010 ...
Apr04 0110000001100 ...
'''

Kategorie

TV 1001000000010 ...
DVD 0100000001000 ...
Video 0010100010101 ...
Stereo 0000011100000 ...
...

Geschlecht

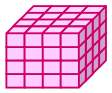
M 1010010001001 ...
W 0111101110110 ...

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)

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	
...	...	...									...

- für jede Bereichsanfrage max. 2 Bitlisten zu lesen
  - Bereich A < x <= E:
  - Bereich x <= E:
  - Bereich x > A:
- Punktanfrage (Gleichheitsbedingung) erfordert Lesen von 2 Bitlisten (vs. 1 Bitliste bei Standard-Bitlisten-Index)



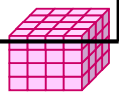
Intervallkodierte Bitlisten-Indizes

- Optimierung bereichskodierter Bitlisten-Indizes mit reduziertem Speicherplatzbedarf
- 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 halbierter Speicheraufwand

ID	Monat	Standard-Bitlisten									intervallkodierte Bitlisten						
		Jan	Feb	Mär	Ap	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	1	0	0	0	0	0	0
125	März	0	0	1	0	0	0	0	..	0	1	1	1	0	0	0	0
126	Februar	0	1	0	0	0	0	0	..	0	1	1	0	0	0	0	0
127	Dezember	0	0	0	0	0	0	0	..	1	0	0	0	0	0	0	1
128	April	0	0	0	1	0	0	0	..	0	1	1	1	1	0	0	0
...	...	...									...						

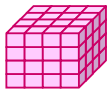


Bitlisten-Indizes: Speicherplatzbedarf

- Speicherplatzbedarf für Bitlisten vs. Listen mit Satzverweisen (TID-Listen)?

  n Sätze
  k Attributwerte
  t = Länge Satzverweis (TID) in B

- Speicherplatzersparnis durch kodierte Bitlisten (encoded bitmap indexing)
  - Standardverfahren: pro Satz ist nur in einer der k Bitlisten das Bit gesetzt
  - zugehöriger Wert läßt sich mit weniger als k Bits repräsentieren
  - unterschiedliche Kodierungsmöglichkeiten: logarithmisch, Mehrkomponenten-Kodierung, Berücksichtigung von Dimensionshierarchien



Kodierte Bitlisten-Indizes

- Logarithmische Kodierung
  - jede der k Wertemöglichkeiten wird durch log2 k Bits kodiert => nur noch $\log_2$ k Bitlisten
  - hohe Einsparungen bei großen k (z.B. Kunden, Produkte)

		Kodierung	F_1	F_0
Blau	0001100010001 ...	Blau		
Rot	1100000000010 ...	Rot		
Weiß	0010000001100 ...	Weiß		
Grün	0000011100000 ...	Grün		

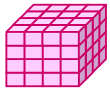
2 Bitvektoren

F_1

F_0

- Auswertung von Suchausdrücken
  - höherer Aufwand bei nur 1 Bedingung ($\log_2$k Bitlisten statt 1 abzuarbeiten)
  - bei mehreren Bedingungen kann ggf. Auswertungsaufwand reduziert werden

- 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

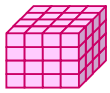


Kodierung für hierarchische Dimensionen

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

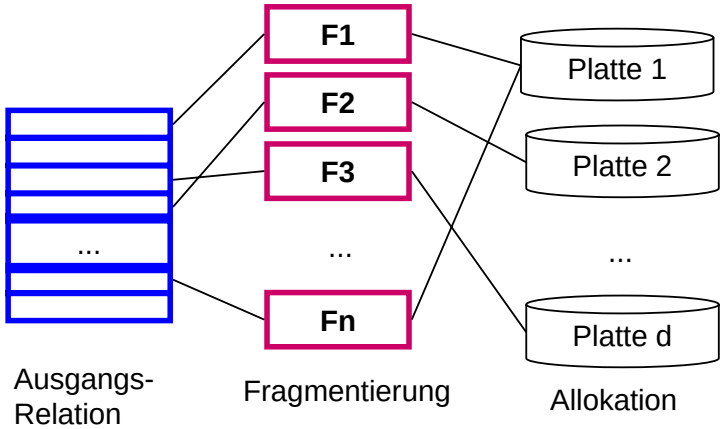
	BEREICH	*FAMILY*	*GRUPPE*	*PNR*	Gesamt
#Elemente Gesamt	8	96	1536	49,152	49152
#Elemente pro Vorgänger	8	12	16	32	
#Bits für Kodierung ($\log_2$)	3	4	4	5	16
Beispiele-Bitmuster	bbb	ffff	gggg	ppppp	bbbffffggggppppp

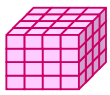
- Auswertungen für gröbere Granulate erfordern nur Zugriff auf Teilmenge der Bitlisten



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 und 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

Globale Relation

KUNDE

KNR	*NAME*	*GEBDAT*	*Filiale*
K2	Schulz	2.11.1976	F
K4	Meier	23.8.1972	B
K3	Müller	4.7.1987	B
K1	Scholz	24.4.1959	F
K5	Weber	17.3.1942	L

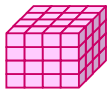
KNR	*NAME*	*Filiale*
K2	Schulz	F
K4	Meier	B
K3	Müller	B
K1	Scholz	F
K5	Weber	L

KUNDE1 = $\pi_{KNR, NAME, Filiale}$ ***(KUNDE)***

KNR	*GEBDAT*
K2	2.11.1976
K4	23.8.1972
K3	4.7.1987
K1	24.4.1959
K5	17.3.1942

KUNDE2 = $\pi_{KNR, GEBDAT}$ ***(KUNDE)***

- Verlustfreie Zerlegung:
  - Primärschlüssel i.a. in jedem Fragment enthalten
  - JOIN-Operation zur Rekonstruktion des gesamten Tupels
- Arbeitsersparnis durch Auslagern selten benötigter Attribute in eigene Fragmente



Projektions-Index

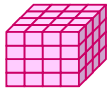
- separate Speicherung der Attributwerte ausgewählter Attribute (vertikale Partitionierung)

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

Projektions-Index
GEHALT

45000
51000
48000
55000
65000
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 Pi auf der Relation:
  $R_i := \sigma_{Pi}\ (R)\ \ (1 \le i \le\ n)$
  - Vollständigkeit: jedes Tupel ist einem Fragment eindeutig zugeordnet
  - Fragmente sind disjunkt: $R_i \cap R_j = \{\}\quad (i \neq j))$
  - Verlustfreiheit: Relation ist Vereinigung aller Fragmente: $R = \cup R_i\ \ (1 \le i \le\ n)$
- Anfragen auf Fragmentierungsattribut werden auf Teilmenge der Daten begrenzt
- Parallelverarbeitung unterschiedlicher Fragmente

KNR	*NAME*	*GEBDAT*	*Filiale*
K2	Schulz	2.11.1976	F
K4	Meier	23.8.1972	B
K3	Müller	4.7.1987	B
K1	Scholz	24.4.1959	F
K5	Weber	17.3.1942	L

Globale Relation KUNDE

KNR	*NAME*	*GEBDAT*	*Filiale*
K5	Weber	17.3.1942	L

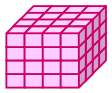
KUNDE1 = $\sigma_{Filiale = "L"}$ ***(KUNDE)***

KNR	*NAME*	*GEBDAT*	*Filiale*
K2	Schulz	2.11.1976	F
K1	Scholz	24.4.1959	F

KUNDE2 = $\sigma_{Filiale = "F"}$ ***(KUNDE)***

KNR	*NAME*	*GEBDAT*	*Filiale*
K4	Meier	23.8.1972	B
K3	Müller	4.7.1987	B

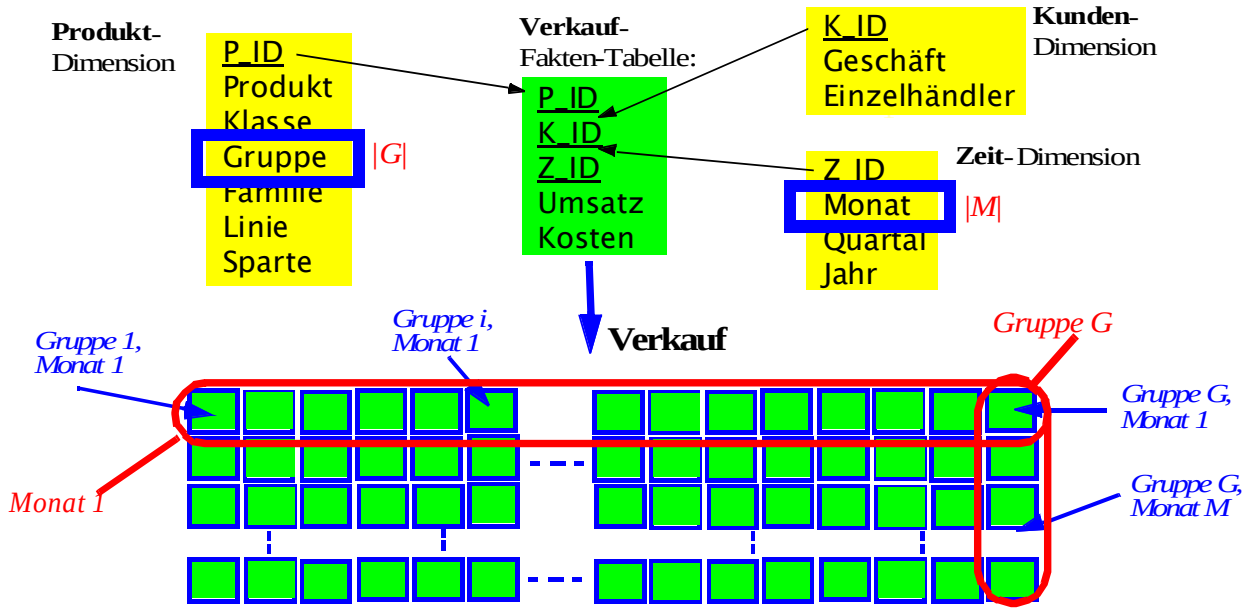
KUNDE3 = $\sigma_{Filiale = "B"}$ ***(KUNDE)***

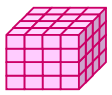


Multi-dimensionale, hierarchische Fragmentierung (MDHF)

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

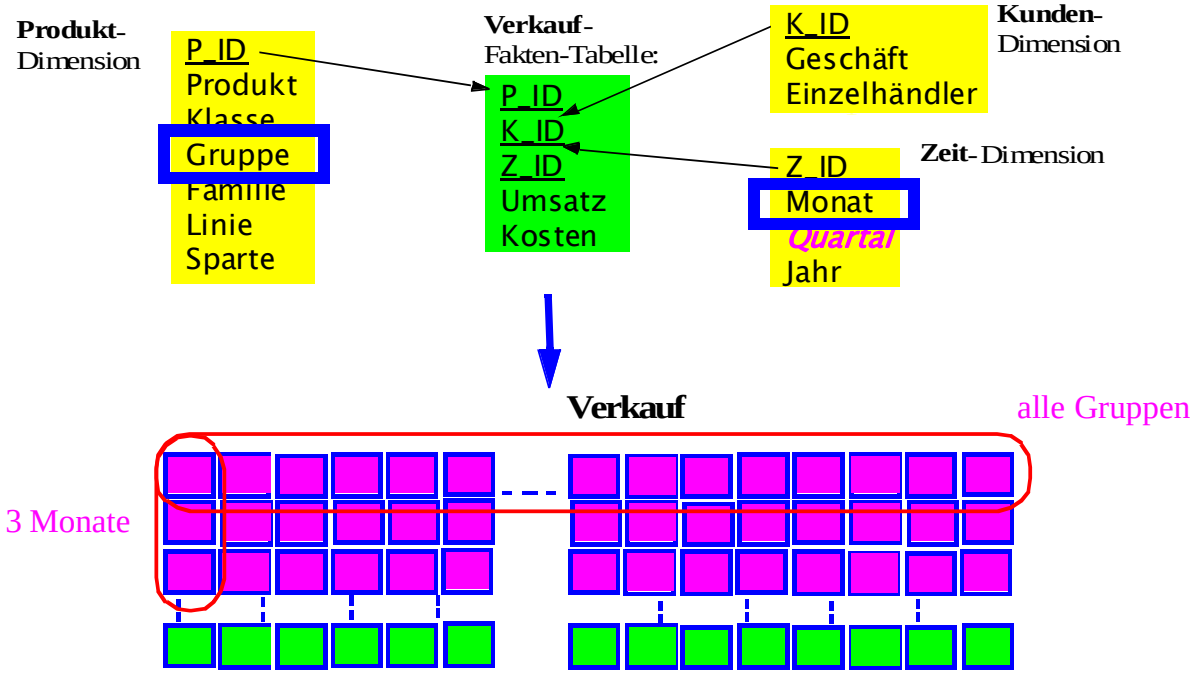
Beispiel: 2-dimensionale Fragmentierung $F_{GruppeMonat}$ der Faktentabelle **Verkauf**



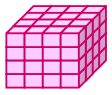


Beispiel: Sternschema-Anfrage

- Zugriff oberhalb einer Fragmentierungsebene (Anfrage auf ***Quartal***)

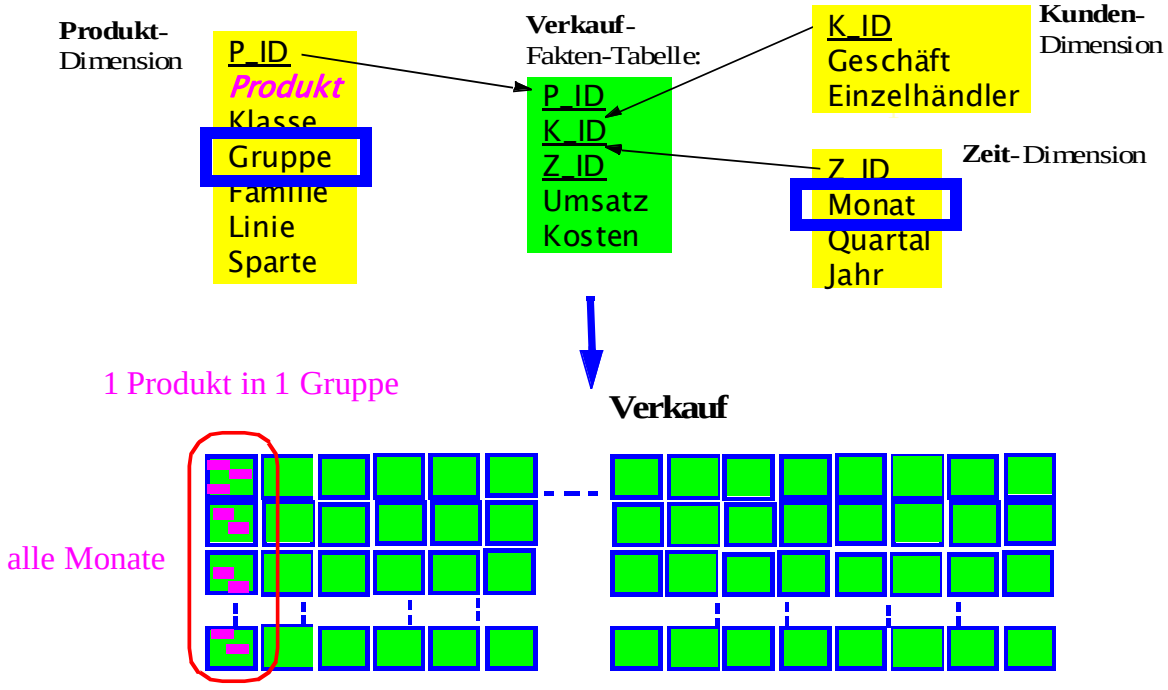


- Clusterung der Treffer, keine Indexzugriffe notwendig

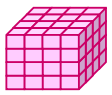


Sternschema-Anfrage unter MDHF (2)

- Zugriff unterhalb einer Fragmentierungsebene (Anfrage auf ***Produkt***)

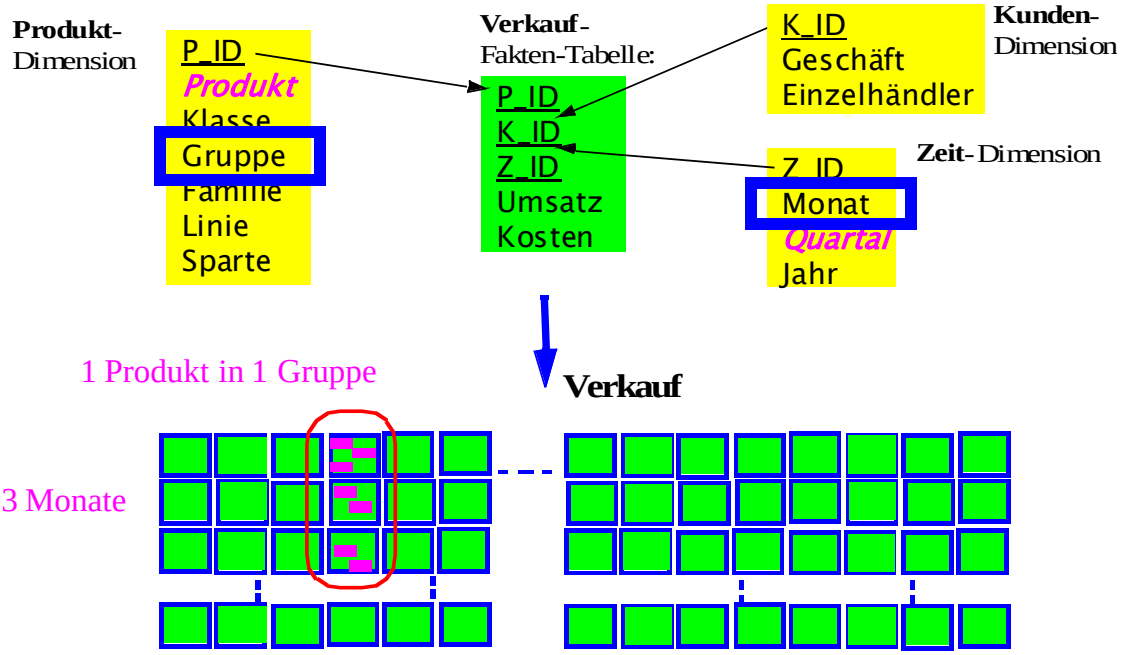


- Einschränkung der Fragmente, ggf. Index-Zugriff zum Auffinden des Produktes im Fragment



Sternschema-Anfrage unter MDHF (3)

- Zugriff oberhalb und unterhalb des Fragmentierungsebene (Anfrage auf ***Quartal*** und ***Produkt***)



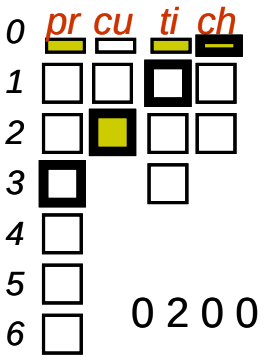
- nur 3 Fragmente, ggf. Index-Zugriff zum Auffinden des Produktes im Fragment

Eigenschaften von MDHF*

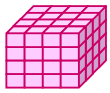
- Reduktion des E/A-Aufwandes für viele Warehouse-Anfragen
  - durch Bezug auf *mehrere* Dimensionen
  - schon bei mindestens 1 referenzierter Fragmentierungs*dimension*
  - Anfrageattribute müssen nicht mit Fragmentierungsattributen übereinstimmen
- Einsparung von Bitlisten-Indizes
  - Materialisierung von Indizes nur für Ebenen *unterhalb* der Fragmentierungsebene(n)
  - Zugriff nur auf zu Trefferfragmenten assoziierten Indexfragmenten
- 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 (*War*ehouse Al*loc*ation To Disk):

* Stöhr, T., Märtens, H., Rahm, E.: *Multi-Dimensional Database Allocation for Parallel Data Warehouses* Proc. 26th Intl. Conf. On Very Large Databases (VLDB), 2000
Stöhr, T., Rahm, E.: *Warlock*: A Data Allocation Tool for Parallel Warehouses. Proc. 27th VLDB Conf., Sep. 2001 (software demo)

Beispiel: Fragmentierungskandidaten für Q_{Store}

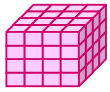


	IO-Aufwand				kombinierte IO-Metrik			
Rang	Fragmentierung	F_{FT}	IOA_Q (in s)	Rang	Fragmentierung	F_{FT}	$IORT_Q$ (in s)	IOA_Q (in s)
1	0 2 0 0	1	8.96	1	3 2 1 0	240	0.135	10.8
2	0 2 1 0	2	9.0	2	1 2 1 1	240	0.135	10.8
3	1 2 0 0	8	9.0	3	4 2 0 0	480	0.146	13.9
4	0 2 2 0	8	9.0	4	2 2 3 0	576	0.175	16.7
5	1 2 1 0	16	9.36	5	0 2 2 1	360	0.180	10.8
...				...				
14	3 2 1 0	240	10.8	34	0 2 0 0	1	8.96	8.96



Materialisierte Sichten

- materialisierte Sichten: materialized views, vorberechnete Aggregationen, 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
- Unterstützung in kommerziellen DBS: Oracle, DB2, MS SQL Server



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

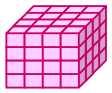
```
select sum (v.GBetrag)
from VERKAUF v, PRODUKT p, ZEIT z
where v.Tag = z.Tag and v.P_Id = p.P_Id
and z.Monat = "Jun04" and
p.Kategorie = "Video"
```

modifizierte Anfrage Q'

```
select
```

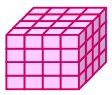
mat. Sicht M1

```
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)
```



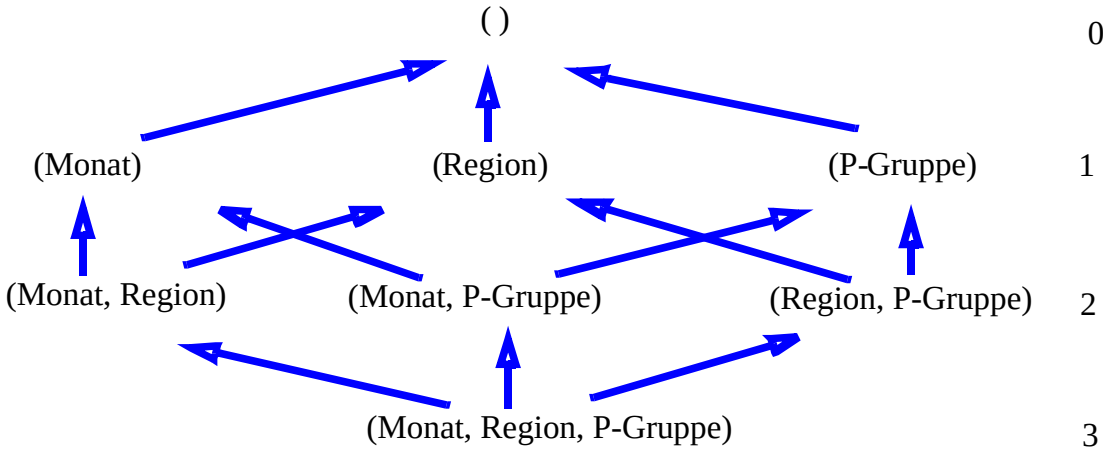
Auswahl materialisierter Sichten

- Optimierungs-Tradeoff: Nutzen für Anfragen vs. erhöhten Speicherungs- und Aktualisierungskosten
- statische Bestimmung durch DBA oder Tool:
  - keine Änderung bis zur nächsten Warehouse-Aktualisierung
  - höchstens Anfragen aus Vergangenheit werden berücksichtigt
- dynamische Auswahl: Caching von Anfrageergebnissen (semantisches Caching)
  - Nutzung von Lokalität bei Ad-Hoc-Anfragen
  - günstig bei interaktiven Anfragen, die aufeinander aufbauen (z.B. Rollup)
  - dynamische Pufferverwaltung und Interaktion mit Anfrageoptimierung
- 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

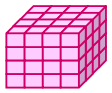


Statische Auswahl materialisierter Sichten

- Konzentration auf vorzuberechnende 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
- Beispiel

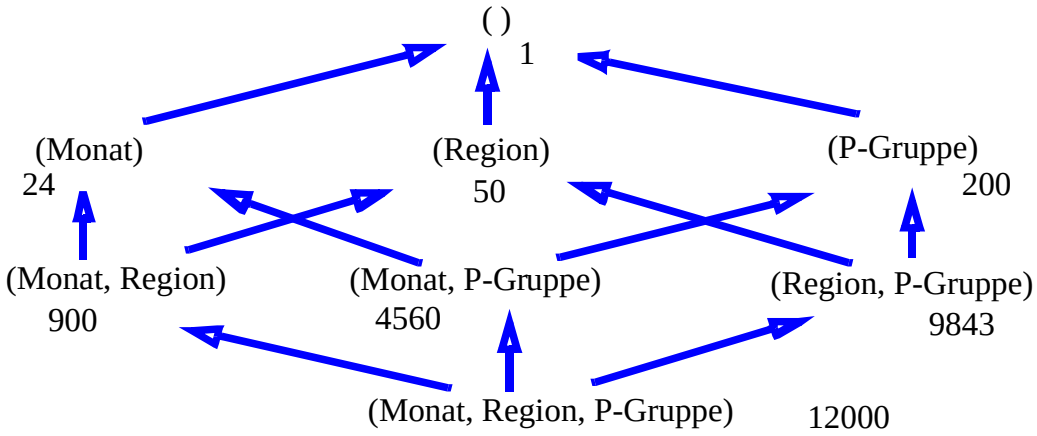


- vollständige Materialisierung aller Kombinationen i.a. nicht möglich
  - #Gruppierungskombinationen wächst exponentiell mit Anzahl von Gruppierungsattributen n
  - möglichst optimale Teilmenge zu bestimmen



Statische Auswahl materialisierter Sichten (2)

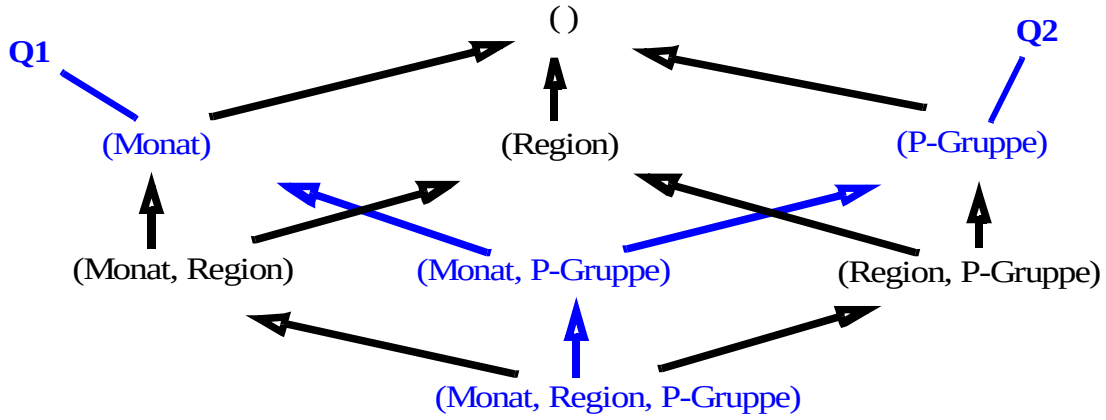
- Heuristik (Harinarayan/Rajaraman/Ullman, Proc. Sigmod 1996)
  - pro Kombination wird Summe der Einsparungen berechnet, die sie für andere nicht materialisierte Kombinationen bewirkt (Aufwand sei proportional zu Anzahl zu berechnender Sätze/Aggregate)
  - in jedem Schritt wird die Kombination ausgewählt, die die größte Summe an Einsparungen zuläßt, solange der maximal zugelassene Speicheraufwand nicht überschritten ist

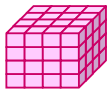


- Beispiel mit Kardinalitäten (Limit sei 75% Zusatzaufwand)
  - Vollauswertung erfordert 12.000 + 15.578 = 27.578 Sätze (+ 130%) -> Beschränkung erforderlich
  - Schritt 1: maximale Einsparung für Monat/Region: nur 900 statt 12.000 Werte auszuwerten für 4 Knoten (Einsparung 4*11.100): (Monat, Region), (Monat), (Region), ()
  - Schritt 2:

Statische Auswahl materialisierter Sichten (3)

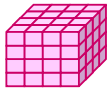
- Berücksichtigung weiterer Faktoren
  - funktionale Abhängigkeiten zwischen Attributen (z.B. innerhalb Dimensionshierarchie) erlauben Eliminierung von Kombinationen
    Gruppierung (Artikel, Produktgruppe) liefert identische Werte wie Gruppierung (Artikel)
  - Eliminierung aller Knoten, für die Verdichtung bestimmten Schwellwert (z.B. 50%) nicht erreicht
- Berücksichtigung nur solcher Knoten, welche vorgegebene Anfragen unterstützen
  - je Paar von Anfragen Berücksichtigung des größten gemeinsamen Vorgänger-Knoten
  - Bsp.: Q1 - Aggregation auf Monat; Q2 - Aggregation für Produktgruppe





Zusammenfassung

- ein- vs. mehrdimensionale Indexstrukturen
- UB-Baum: Abbildung mehrdimensionaler Wertekombinationen auf eindimensionale Reihenfolge (Z-Kurve)
- Bit-Indizes
  - günstig v.a. für Attribute geringer Kardinalität
  - Kodierung für höhere Kardinalität: logarithmisch, Mehr-Komponenten-Kodierung, hierarchische Kodierung
  - Bereichs- und Intervallkodierung
  - effizient nutzbar u.a. für (Star) Joins (Bitlisten-Join-Index); effiziente Mengenoperation
- 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
  - Nutzung hierarchischer Dimensionen
- 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. 5-16 wie folgende Anfrageprädikate evaluiert werden können:
  - Monat="Januar"
  - Monat IN („Mai","Juni","Juli")
- 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?