Data Warehousing und Data Mining

Logische Optimierung

Ulf Leser

Wissensmanagement in der Bioinformatik

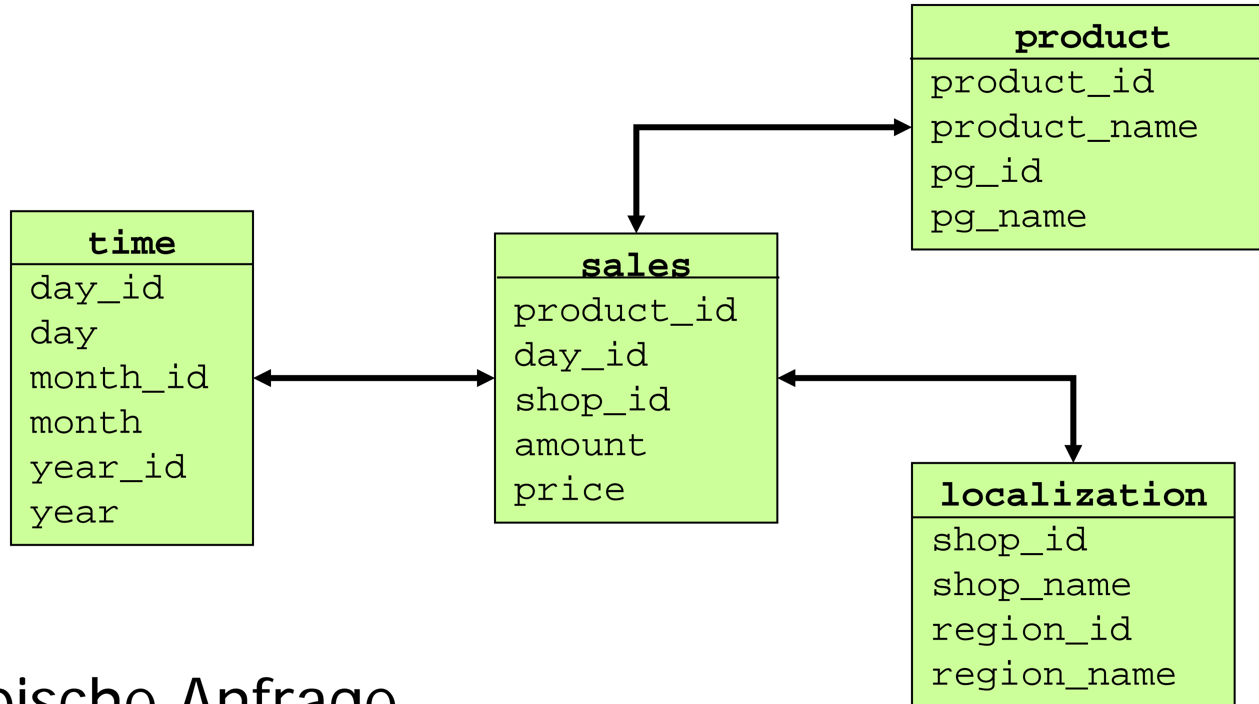
Übersicht

- Konzeptionell: Modellierung und Sprachen
  - Architektur & Prozesse
  - Multidimensionale Modellierung
    - MDDM, MER
    - ROLAP und MOLAP
    - OLAP Operationen und Sprachen
  - Extraction, Transformation, Load (ETL)
    - Differential Snapshots
    - Transformations and types of heterogeneity
    - Bulk loading
- Umsetzung: Logische und physische Ebene
  - Indexstrukturen für DWH: Bitmap, Join-Indexe
  - Multidimensionale Indexstrukturen: Grid-File, kd-Tree
  - Optimierung: Star-Join, Partitionierung
  - Implementierung von OLAP Operationen
  - Column Stores, Main Memory, Map-Reduce
- Materialisierte Sichten
  - Auswahl
  - Query Rewriting
  - Aktualisierung
- Data Mining
  - Datenaufbereitung
  - Clustering: k Means, DBScan, hierarchisch
  - Klassifikation: kNN, Naive Bayes, Decision Trees
  - Assoziationsregeln

Inhalt dieser Vorlesung

- Star-Join
  - Grundidee
  - Star-Join mit Bitmap-Indexen
  - Bloom-Filter
- Partitionierung

DWH Queries



- Typische Anfrage
  - Joins zwischen Dimensions- und Faktentabelle
  - Aggregation (Fakten) und Gruppierung (Dimensionen)
  - Bedingungen auf den Dimensionstabellen
- Thema hier: Optimale Joinreihenfolge

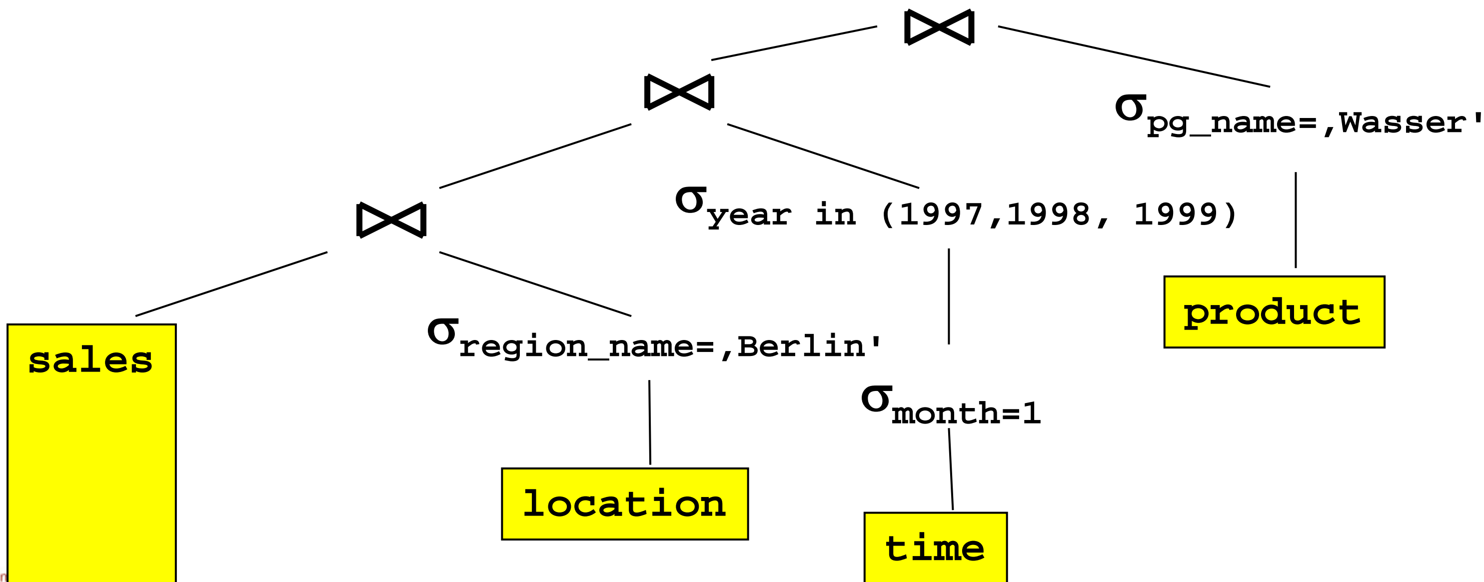
Beispiel

- Alle Verkäufe von Produkten der Produktgruppe ‚Wasser' in Berlin im Januar der Jahre 1997, 1998, 1999, gruppiert nach Jahr

```
SELECT  T.year, sum(amount*price)
FROM    sales S, product P, time T, localization L
WHERE   P.pg_name=‚Wasser' AND
        P.product_id = S.product_id AND
        T.day_id = S.day_id AND
        T.year in (1997, 1998, 1999) AND
        T.month = ‚1' AND
        L.shop_id = S.shop_id AND
        L.region_name=‚Berlin'
GROUP BY T.year
```

Anfrageplanung

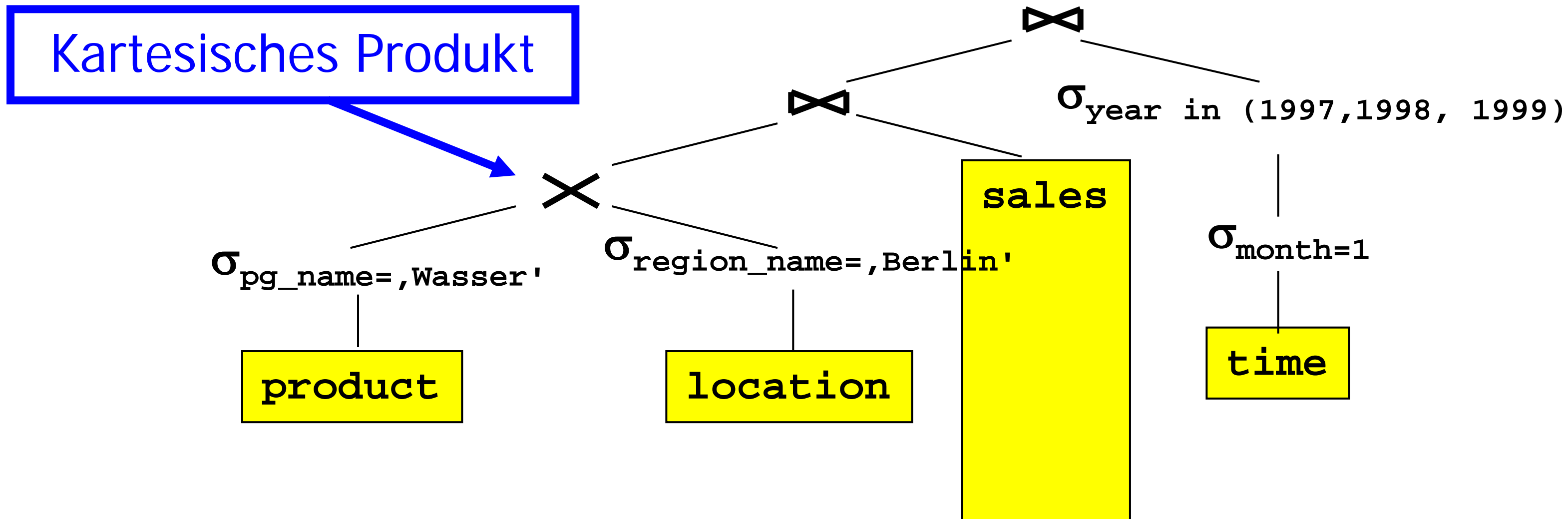
- 3 Joins über 4 Tabellen: 4! left-deep join trees
  - Andere berücksichtigen die meisten Optimierer nicht
- Aber: In der Query sind nicht alle Tabellen mit allen anderen gejoined
- Nur 2*2*3 der 4! Pläne enthalten kein kartes. Produkt
  - **sales** als erste oder zweite Relation ausgewählt



Join-Optimierung

```
SELECT   T.year, sum(amount*price)
FROM     sales S, product P, time T, localization L
WHERE    P.pg_name=‚Wasser‘ AND
         P.product_id = S.product_id AND
         T.day_id = S.day_id AND
         T.year in (1997, 1998, 1999) AND
         T.month = ‚1‘ AND
         L.shop_id = S.shop_id AND
         L.region_name=‚Berlin‘
GROUP BY T.year
```

- Typisches Vorgehen
  - Auswahl des Planes nach Größe der Zwischenergebnisse
  - Keine Beachtung von Plänen, die kartesisches Produkt enthalten



Abschätzung von Zwischenergebnissen

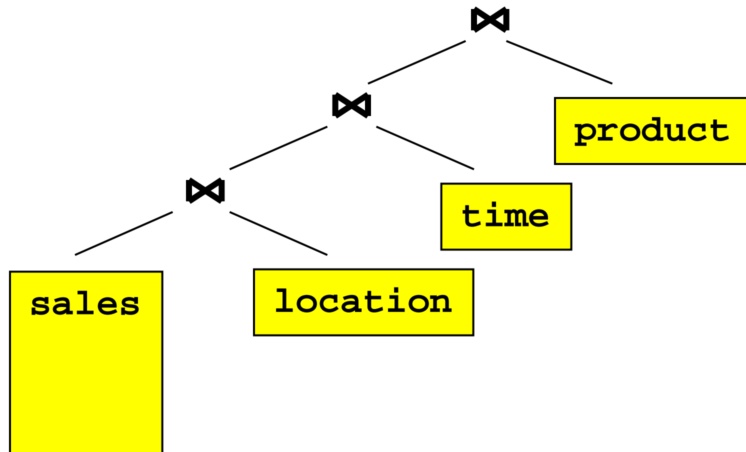
```
SELECT        T.year, sum(amount*price)
FROM   sales S, product P, time T, loc L
WHERE  P.pg_name=‚Wasser‘ AND
       P.product_id = S.product_id AND
       T.day_id = S.day_id AND
       T.year in (1997, 1998, 1999) AND
       T.month = ‚1‘ AND
       L.shop_id = S.shop_id AND
       L.region_name=‚Berlin‘
GROUP BY T.year
```

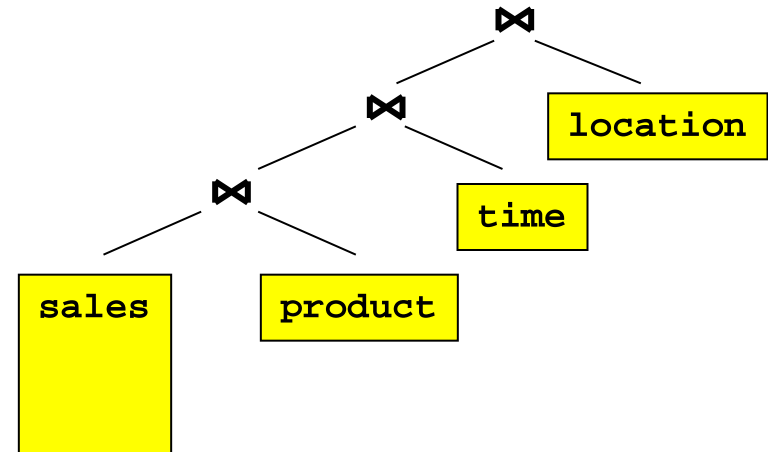
Annahmen

- m= |S| = 100.000.000
- 20 Verkaufstage pro Monat
- Daten von 10 Jahren
- 50 Produktgruppen a 20 Produkten
- 15 Regionen a 100 Shops
- Gleichverteilung aller Verkäufe

- Größe des Ergebnisses
  - Jan 97,98,99 - 60 Tage: (m / (20*12*10)) * 3 * 20
  - ‚Wasser' - 20 Produkte : (m / (20*50)) * 20
  - ‚Berlin' - 100 Shops (m / (15*100)) * 100
- Gesamt: ~3.333 Tupel
- Selektivität: 0,00003%

Reihenfolge zählt

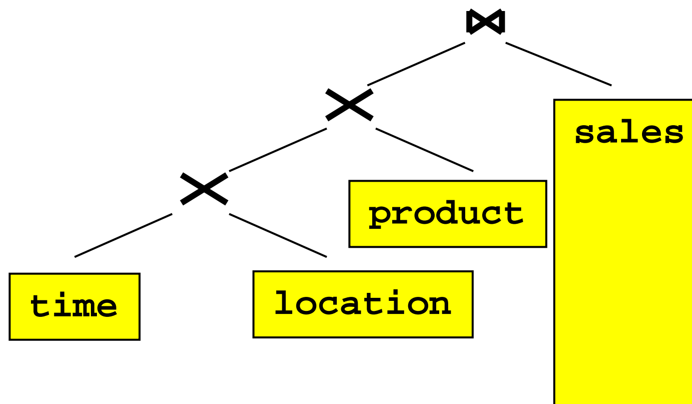




	Zwischen-ergebnis
1. Join (m / 15)	6.666.666
2. Join ($\|J_1\|$*3/120)	166.666
3. Join ($\|J_2\|$/50)	3.333

	Zwischen-ergebnis
1. Join (m / 50)	2.000.000
2. Join ($\|J_1\|$*3/120)	50.000
3. Join ($\|J_2\|$/ 15)	3.333

Plan mit kartesischen Produkten



	Zwischenergebnis
1. |time| x |location| (3*20*100)	6.000
2. … x |product| (|P_1|*20)	120.000
3. … ⋈ |sales|	3.333

Star-Join in Oracle 7

- Kartesisches Produkt aller Dimensionstabellen
- Zugriff auf Faktentabelle über Index
  - Hohe Selektivität für Anfrage wichtig
  - Zusammengesetzter Index auf allen FKs muss vorhanden sein
    - Achtung: Problem der Attributfolge im Index versus Query
    - Bitmap-Indexe gab es noch nicht
- Fixes Ablaufschema, wenn Star-Join erkannt wurde

Star-Join ab Oracle 8i

- Benutzung komprimierter Bitmapindexe
  1. Berechnung aller FKs in Faktentabelle gemäß Dimensionsbedingungen einzeln für jede Dimension
  2. Anlegen/laden von bitmapped Join-Indexen auf allen FK Attributen der Faktentabelle
  3. Merge (AND) aller Bitmapindexe
     - Das berechnet den Inner-Join der Fakten mit allen Dimensionen
  4. Zugriff auf Faktentabelle über TID
  5. Join nur der selektierten Fakten mit Dimensionstabellen zum Zugriff auf Dimensionswerte
- Die großen Zwischenergebnisse sind jetzt nur Bitstrings
  - Passen (hoffentlich) in den Hauptspeicher

Beispiel – Schritt 1

```
SELECT  T.year, sum(amount*price)
FROM    sales S, product P, time T, localization L
WHERE   P.pg_name=‚Wasser‘ AND
        P.product_id = S.product_id AND
        T.day_id = S.day_id AND
        T.year in (1997, 1998, 1999) AND
        T.month = „1“ AND
        L.shop_id = S.shop_id AND
        L.region_name=‚Berlin‘
GROUP BY T.year
```

```
SELECT  T.year, sum(amount*price)
FROM    sales S, time T
WHERE   S.day_id IN
         (SELECT day_id FROM time
          WHERE year in (1997, 1998, 1999) AND month=‚1‘)
        AND S.shop_id IN
         (SELECT shop_id FROM localization WHERE region_name=‚Berlin‘)
        AND S.product_id IN
         (SELECT product_id FROM product WHERE pg_name=‚Wasser‘)
        AND S.day_id = T.day_id
GROUP BY T.year
```

Schritt 2 & 3

```
SELECT  T.year, sum (amount * price)
FROM    sales S, time T
WHERE   S.day_id IN
         (SELECT day_id FROM time
          WHERE year in (1997, 1998, 1999) AND month=‚1‘)
        AND S.shop_id IN
         (SELECT shop_id FROM localization WHERE region_name=‚Berlin‘)
        AND S.product_id IN
         (SELECT product_id FROM product WHERE pg_name=‚Wasser‘)
        AND S.day_id = T.day_id
GROUP BY T.year
```

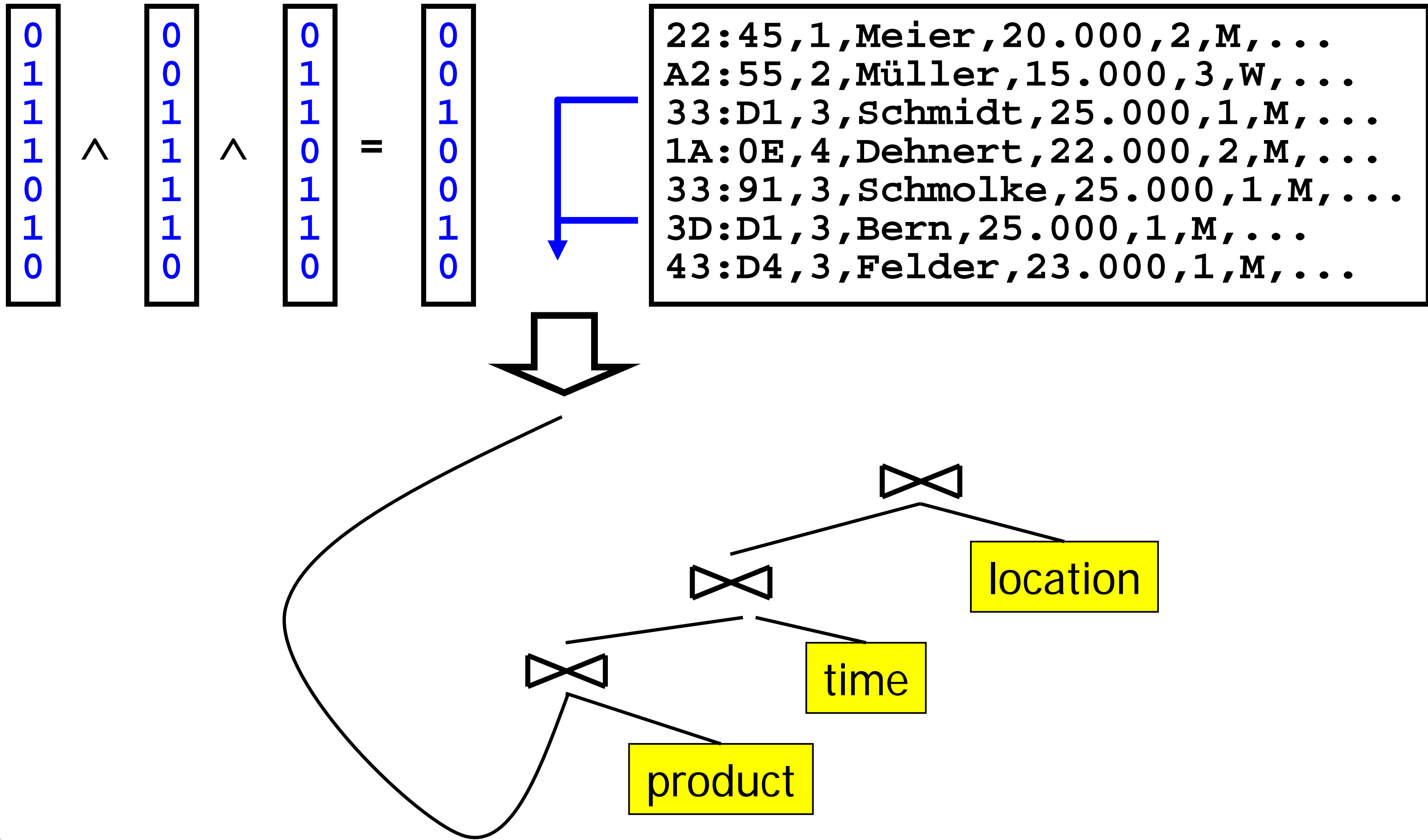
2400 Bitarrays
Davon ~60 gewählt
Mit OR verknüpfen

1500 Bitarrays
Davon ~100 gewählt
Mit OR verknüpfen

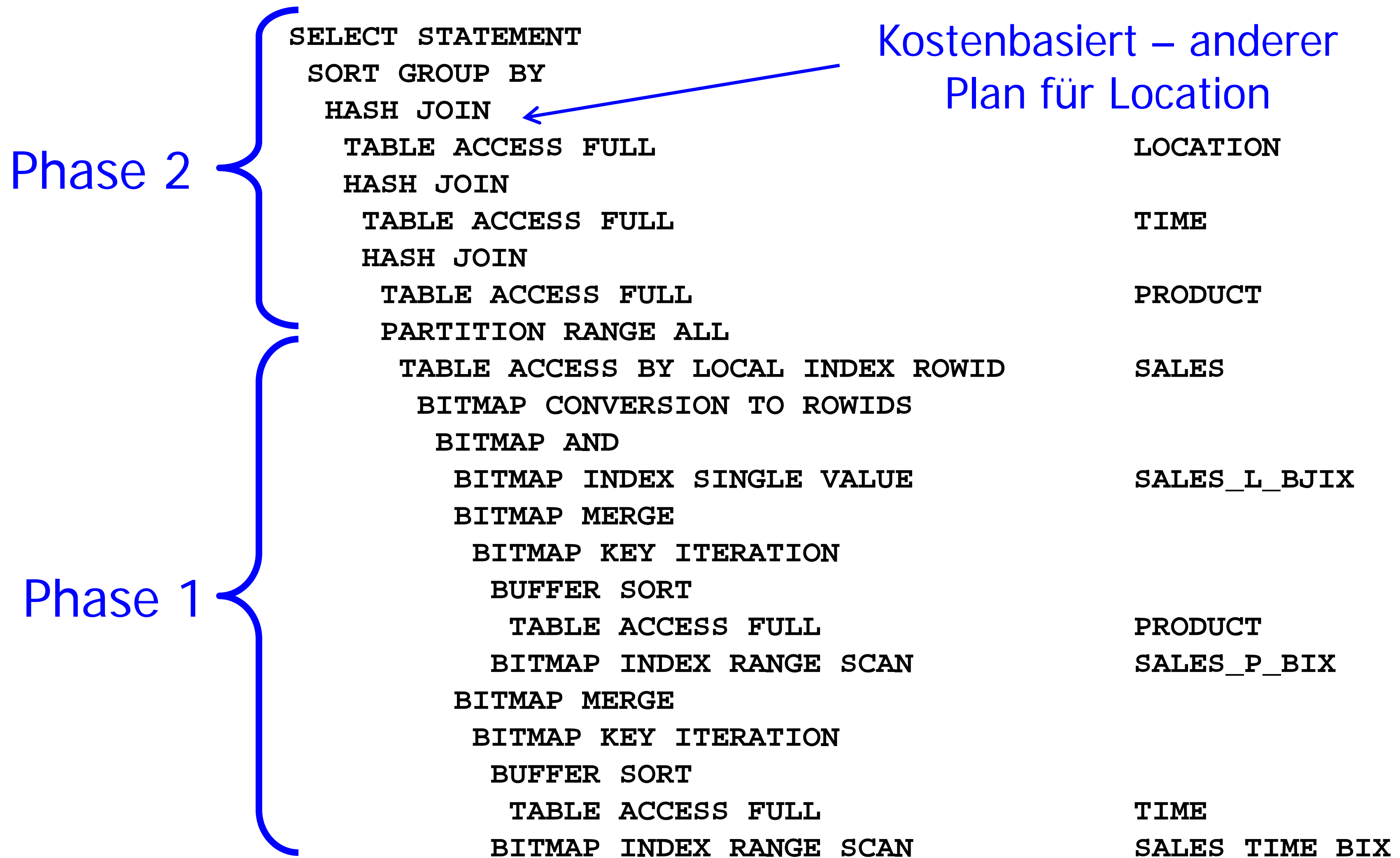
1000 Bitarrays
Davon ~20 gewählt
Mit OR verknüpfen

AND

Schritt 4 & 5



Gesamtplan



Verfeinerung: Bloom-Filter

- Vorteile gehen verloren, wenn Bitarrays nicht in den Hauptspeicher passen
- Weitere Idee: Bloom-Filter
  - Schritte 2-3 werden „unscharf“ ausgeführt
    - Ein Bit im Bitstring repräsentiert mehrere FKs
  - „Unschärfe“ wird so konstruiert, dass es nur Falsch-Positive aber keine Falsch-Negativen geben kann
    - Falsch-Positiv: Tupel wird selektiert, obwohl es nicht sollte
    - Falsch-Negativ: Tupel wird nicht selektiert, obwohl es sollte
  - Falsch-Positive werden in Schritt 4/5 herausgefiltert
    - Hier werden ja die wirklichen Werte gelesen
  - Bringt dann Vorteile, wenn FP-Wahrscheinlichkeit gering ist und durch die Unschärfe die Bitstrings in den Hauptspeicher passen
    - Trade-Off: Mehr Unschärfe – weniger Hauptspeicher – mehr FPs

Ablauf

- Aufbau zweier Bitmaps für Faktentabelle S
  - Erzeuge zwei maximal große Bitmaps B_1 und B_2
  - Wähle eine Hashfunktion $h(S) \rightarrow B_1$ (bzw. B_2)
  - Weil $|B_1|=|B_2|<|S|$: Mehrere TID mappen auf ein Bit
- Initialisierung
  - Auswahl der Dimension D mit höchster Selektivität
  - Für alle selektierten Tuple t aus D
    - Scan Faktentabelle nach FK-Wert aus t
    - $B_1[h(TID)]=1$
- Iteriere über alle restlichen D_i
  - Berechne $B_2[]$ für D_i wie bei D
  - $B_1[] = B_1[]$ AND $B_2[]$

Ergebnis

- Wenn $|B_1|=|B_2|=|S|$ wäre
  - Nur Tupel TID mit $B_1[h(TID)]=1$ sind relevant
- Wir haben aber
  - $B_1[z]=0$: Alle Tupel mit $h(TID)=z$ garantiert nicht relevant
    - Selbst Kombinationen aller Tupel mit $h(TID)=z$ schaffen keine 1 in allen notwendigen Dimensionen
  - $B_1[z]=1$: Alle Tupel mit $h(TID)=z$ eventuell relevant
    - Das sind potentiell Falsch-Positive
- Erneute Auswertung aller Bedingungen nach Joins mit Dimensionstabellen
  - Filterschritt für Falsch-Positive
  - Menge der zu untersuchenden Fakten aber idR schon sehr klein

Eigenschaften

- Implementiert in DB2
- Kernidee ist Reduktion der Faktentabelle vor Ausführen der Joins mit Dimensionstabellen
  - Faktentabelle muss aber mehrmals gescanned werden
  - Aber immer nur Zugriff auf (wenige) FK's
- Dynamischer Aufbau von Bitmap-Indexen
  - Keine Vorabfestlegung des verwendbaren Speichers, sondern optimale Ausnutzung des verfügbaren Speichers
- Voraussetzungen für Gewinn
  - Hohe Selektivität in Dimensionsbedingungen
  - Aufbau hinreichend großer Bitmaps möglich
  - Wahrscheinlichkeit für FPs steigt überproportional mit Verhältnis |S| / |B|

Inhalt dieser Vorlesung

- Star-Join
- Partitionierung
  - Partitionierungsarten
  - Management von Partitionen
  - Spezielle Themen

Partitionierung

- Aufteilung der Daten einer Tabelle in Untereinheiten
  - Physische Partitionierung (Allokation): Für den Benutzer transparent
    - Nach Definition der Partitionen
  - Logische Partitionierung: Für den Benutzer nicht transparent
    - Explizites Anlegen mehrerer Tabellen
    - Anfragen müssen entsprechend formuliert werden
- Ursprünglich für verteilte Datenbanken entwickelt
  - Verteilung der Partitionen auf verschiedenen Knoten
  - Vereinfachte Synchronisation & Lastverteilung
  - Wichtig ist dann Lokalität Anfrage – Partition
    - Ein Knoten soll eine Anfrage möglichst alleine ausführen können
    - Im parallelen Fall will man lieber alle Threads gleichzeitig arbeiten lassen

Vertikale Partitionierung

id	price	amount	code	form	...
1	50	4	011010	CASH	
2	60	2	110101	VISA	
3	20	1	001101	VISA	

id	price	amount
1	50	4
2	60	2
3	20	1

id	code	form	...
1	011010	CASH	
2	110101	VISA	
3	001101	VISA	

Bewertung

- „Zerstört“ semantische Einheiten – die Tupel
- Zusammenfassen erfordert (teure) Joins
- Geeignete Technik zur „Auslagerung“ von ...
  - Selten benutzten Attributen
  - Attributen, die häufiger als andere verändert werden
    - Und deshalb zu Reorganisation / Row Splits führen
  - BLOBs, LONGS, etc.
- Herstellung von Transparenz?
  - Definition eines Views – Join muss aber berechnet werden

Horizontale Partitionierung

id	price	amount	code	form	...
1	50	4	011010	CASH	
2	60	2	110101	VISA	
3	20	1	001101	VISA	
4	30	4	110101	CASH	

1	50	4	011010	CASH
2	60	2	110101	VISA

3	20	1	001101	VISA
4	30	4	110101	CASH

1	50	4	011010	CASH
4	30	4	110101	CASH

2	60	2	110101	VISA
3	20	1	001101	VISA

Prinzip

```
CREATE TABLE sales_range
       (salesman_id  NUMBER(5),
        salesman_name VARCHAR2(30),
        sales_amount  NUMBER(10),
        sales_date    DATE)
PARTITION BY RANGE(sales_date)
( PARTITION t21 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
  PARTITION t22 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
  …);
```

- Partitionen sind eigene Datenbankobjekte (Tabellen)
- Einmal angelegt, ist Existenz für Benutzer transparent
- Die letzte Partition kann als Grenze `MAXVALUE` haben
  - „Specifying a value other than MAXVALUE for the highest partition bound imposes an implicit integrity constraint on the table. „

Arten von Partitionierung (Oracle)

- Bereichspartitionierung
  - Explizite Angabe der Partitionsbereiche
  - Typisch: Zeiträume (Archivierung historischer Daten)
  - Balancierung muss der Administrator verantworten
  - Einfaches Management, Partition Pruning etc.
- Hash–Partitionierung
  - Angabe einer Hashfunktion über Attributwerten eines Tupel
  - Balancierung wird (hoffentlich) durch Hashfunktion erreicht
  - Führt idR zu hoher Parallelität in Anfragebearbeitung
- Achtung: Ungleiche Partitionsgrößen: Evt. sogar Performanzverschlechterung
  - Da Parallelisierung ausgehebelt

Horizontale Partitionierung

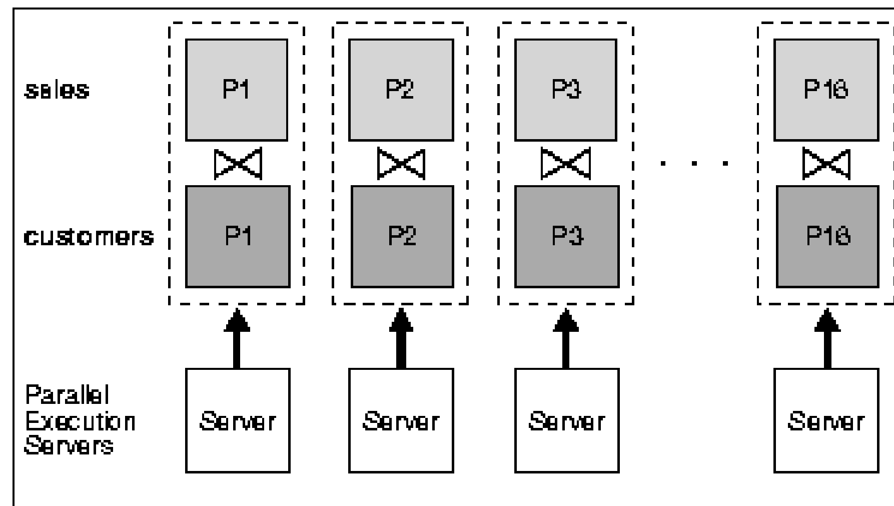
- Wichtige Erweiterung der meisten kommerziellen RDBMS zur Parallelisierung von Anfragen
- Admin legt Partitionen an, SQL-Entwickler (Benutzer) muss sie nicht kennen
- Hauptvorteile
  - Parallele Verarbeitung
  - Datenmanagement – Partitionen als eigenständige Datenbankobjekte
  - Scans können Partitionen auslassen (Partition pruning)

Parallelität

- Interquery
  - Verteilen von Anfragen auf Prozessoren / Knoten
  - Keine Beschleunigung einzelner Queries
  - Behandlung auf Session-Ebene (Dedicated Server / MTS)
- Intraquery
  - Aufbrechen der Query in parallel ausgeführte Teilanfragen
  - Typischerweise erfolgt die Aufteilung durch die Daten: Query wird mehrmals parallel auf disjunkten Datenbereichen ausgeführt
    - „Datenparallel“: Geht nicht immer (z.B. ORDER-BY)
  - Welche Datenbereiche? Das bestimmt die Partitionierung (Teile einer) Anfrage laufen mehrmals parallel auf unterschiedlichen Partitionen

Partitionierung und Joins

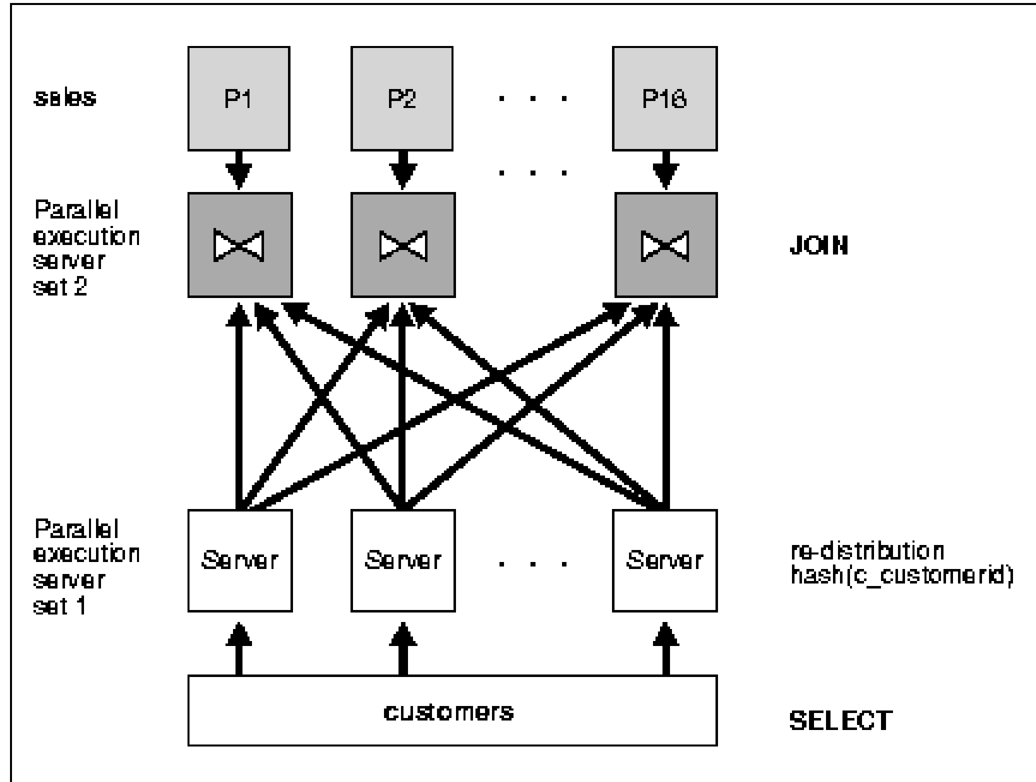
- Join mit zwei auf dem Join-Attribut partitionierten Tabellen



- Effektive Parallelisierung möglich
- Erhebliche Beschleunigung

Partitionierung und Joins

- Join bei nur einer partitionierten Tabelle
  - Dynamische Partitionierung „at runtime“



Datenmanagement

- MERGE – Verschmelzen zweier Partitionen
- ADD – Hinzufügen einer Partition (abh. von P-Art)
- DROP/TRUNCATE – Löschen einer Partition
  - Nur sinnvoll bei Range-Partitionierung
  - Viel schneller als „**`DELETE FROM sales WHERE ...`**“
  - Möglich: Eigenständige Archivierung
  - DWH als „Sliding Window“ über dem Archiv
- Verteilung der Partitionen auf verschiedene Platten
  - Parallelität beim IO Zugriff

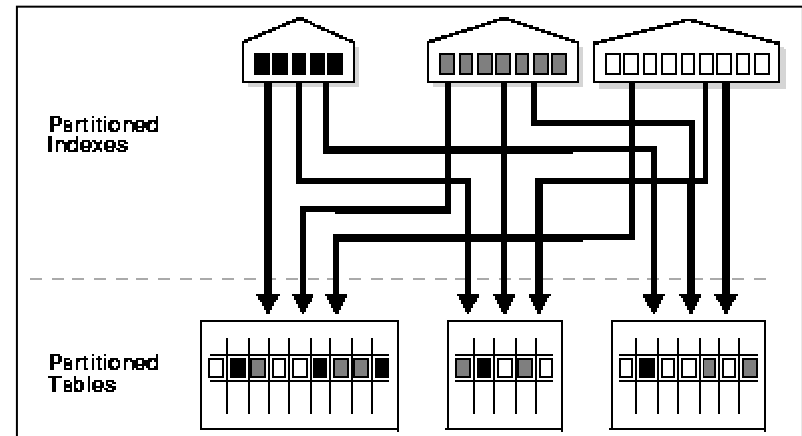
Exchange

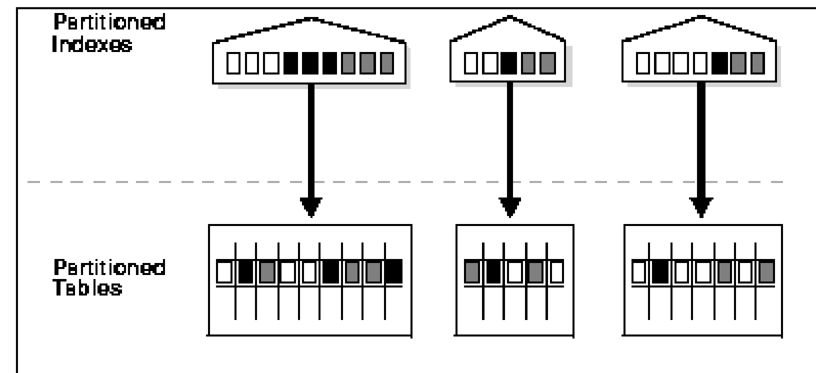
- EXCHANGE: Wandelt Tabelle in Partition (einer anderen Tabelle) um oder umgekehrt
  - Nur sinnvoll bei Range-Partitionierung
  - Sehr nützlich für ETL
  - Beispiel: Vorverarbeitung der Daten eines Tages in einer eigenen Tabelle
  - Dann hinzufügen zu **`sales`** mit einem Befehl
  - Keine Reorganisation, kein Indexneubau, …

Partitionierung und Indexe

- Partitionierte Indexierung partitionierter Tabellen

- „Globale“ Indexe
  - Index unabhängig von Tabelle partitioniert
  - Eigene DDL Kommandos
- „Lokale“ Indexe
  - Index partitioniert wie Tabelle
  - Bildung eines lokalen Index (Indexpartition) pro Partition
  - Manipulation automatisch mit Tabellenpartition (DROP, ADD, TRUNCATE, ...)





Partition Pruning

- Anfragen mit Bedingung auf Partitionierungsattribut
- Punktanfragen
  - Direkte Auswahl relevanter Partition
  - Partition ähnlich einer Ebene in B*-Baum
  - Keine Performanceverbesserung
- Bereichsanfragen
  - Direkte Auswahl relevanter Partitionen
  - Daten liegen partitioniert vor (nicht nur TIDs)
  - Kann erhebliche Performanceverbesserung bringen
- Unterstützte Prädikate
  - Pruning mit Bereichspartitionierung nur bei =, <, >, IN
  - Pruning mit Hashpartitionierung nur bei =, IN

Literatur

- [Leh03]: Kapitel 8.5.2, 8.5.3, 7.1.2
- [BG02]: Kapitel 7.3, 7.4
- Oracle Dokumentation: „Data Warehousing Guide"

Selbsttest

- Was ist ein Star-Join? Was ist anders als bei „normalen“ Joins?
- Erklären Sie den Star-Join in Oracle 8i folgende.
- Was ist ein Bloom-Filter? Warum soll sich der bei Star-Joins lohnen?
- Welche Arten von Partitionierung gibt es? Welche davon dient der Parallelisierung?
- Welche Vorteile hat Hast-Partitionierung gegenüber Range-Partitionierung?
- Wie funktioniert ein paralleler Join über zwei nicht auf dem Join Attribute partitionierten Tabellen?