

## Teil IX

# Materialisierte Sichten

# Materialisierte Sichten

- 1 Begriff und Motivation
- 2 Anfragebeantwortung mit materialisierter Sichten
- 3 Auswahl materialisierter Sichten
- 4 Aktualisierung materialisierter Sichten
- 5 Materialisierte Sichten in DBMS

# Materialisierte Sichten

- Vielzahl gleicher oder ähnlicher Anfragen auf immer denselben Relationen  $\rightsquigarrow$  **Einführung von Sichten zur Anfragevereinfachung**
- überwiegend lesender Zugriff auf weitgehend stabiler Datenbasis  $\rightsquigarrow$  **Materialisierung der Sichten ggf. sinnvoll**
  - ▶ seltene Änderungen in der Datenbasis bedeuten geringen Aufwand bei der Aktualisierung der Sichten
  - ▶ Materialisierung reduziert Berechnungsaufwand bei wiederkehrenden Anfrageteilen
  - ▶ System erkennt automatisch Anfrageteile, deren (Teil-) Ergebnisse durch materialisierte Sichten bereits zur Verfügung stehen
- Auch: **materialized views (MV)**, **summary tables**

# Materialisierte Sichten: Problembereiche

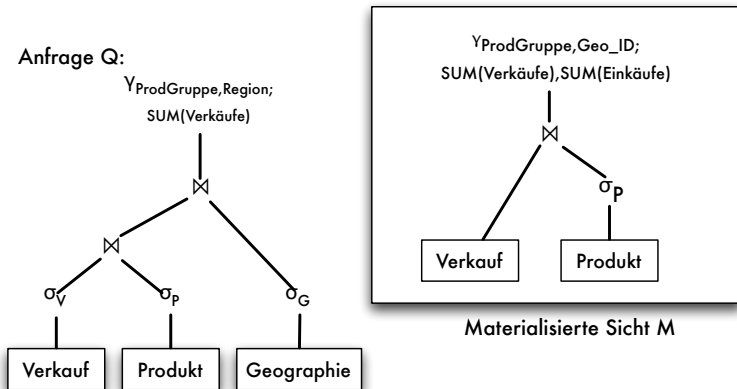
- **Auswahl materialisierter Sichten**  $\rightsquigarrow$  Abwägung zwischen folgenden Kriterien:
  - ▶ Speicherbedarf für redundant gehaltene Daten
  - ▶ zusätzlicher Verwaltungsaufwand durch Materialisierung (einschl. Analyseaufwand für Auswahl der zu materialisierenden Sichten)
  - ▶ erwartete Reduktion von Antwortzeiten
- **Wartung materialisierter Sichten**
  - ▶ Änderungen im Datenbestand erfordern Neuberechnung der materialisierten Sicht oder Update-Propagation

## Problembereiche /2

- Existenz und Konstruktion einer korrekten Anfrageersetzung
- im allgemeinen Fall bereits ein schwieriges Problem (NP-hart)
  - ▶ wenn Restriktionen in der materialisierten Sicht nicht mit der Anfrage übereinstimmen  $\rightsquigarrow$  Einfügen geeigneter Kompensationsoperationen
  - ▶ hinsichtlich der Restriktionsbedingungen Nachweis der logischen Implikation

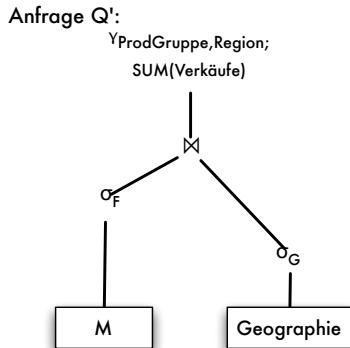
# Verwendung materialisierter Sichten

- Beispiel: Anfrage  $Q$  bei vorhandener Sicht  $M$ :



## Verwendung materialisierter Sichten /2

- Einsetzen der vorhandenen Sicht  $M$  in Anfrage  $Q$  durch Umschreiben („Rewriting“)  $\rightsquigarrow$  Anfrage  $Q'$



## Verwendung materialisierter Sichten /3

- Voraussetzung für derartige Anfrageumformulierungen unter Verwendung materialisierter Sichten:
  - ▶ umformulierte Anfrage ist äquivalent zur ursprünglichen, d.h. sie liefert dasselbe Anfrageergebnis
- 2 Problemstellungen
  - ▶ Existenz einer Anfrageersetzung (**Query Containment**)
  - ▶ eigentliche Anfrageumformung (**Query Rewriting**)

# Query Containment

- Geg.: Anfragen  $Q_1$  und  $Q_2$
- Enthaltensein

$Q_1$  ist in  $Q_2$  **enthalten** ( $Q_1 \sqsubseteq Q_2$ ), wenn für alle Datenbankinstanzen  $D$  die Ergebnismenge von  $Q_1$  eine Teilmenge des Ergebnisses von  $Q_2$  ist:  $Q_1(D) \subseteq Q_2(D)$

- Äquivalenz

$Q_1$  und  $Q_2$  sind **äquivalent**, wenn  $Q_1 \sqsubseteq Q_2$  und  $Q_2 \sqsubseteq Q_1$

## Query Containment-Problem

- unentscheidbar für beliebige relationale Kalküle bzw. Anfragen in Relationenalgebra
- für konjunktive Anfragen entscheidbar, aber NP-vollständig
- durch Beschränkung auf SPJ-Anfragen (Konstantenselektion, natürlicher Verbund, Projektion) auch in polynomialer Zeit

## Query Rewriting

- Geg.: Anfrage  $Q$ , Menge von Sichtdefinitionen  $V_1, \dots, V_n$
- Ges.: Anfrage  $Q'$ , die nur auf Sichten aus  $V_1, \dots, V_n$  zugreift (**FROM**-Klausel)  
(Zugriff auf Basisrelationen ist konzeptionell keine Einschränkung!)
- Äquivalente Umschreibung

$Q'$  ist äquivalente Umschreibung bzgl.  $V_1, \dots, V_n$ , wenn (1)  $Q'$  verweist nur auf Sichten aus  $\{V_1, \dots, V_n\}$  und (2)  $Q'$  ist äquivalent zu  $Q$

- Maximal-enthaltene Umschreibung
  - ▶ es existiert keine weitere Umschreibung  $Q_1$ , so dass  $Q' \sqsubseteq Q_1 \sqsubseteq Q$  und  $Q_1$  ist nicht äquivalent zu  $Q$

# Query Rewriting mit materialisierten Sichten

- Rewriting für SPJ-Anfragen, Chaudhuri et al. 1994
- Definition von MVs als Regel

$$L(x, y) \rightarrow V(x)$$

mit Sicht  $V$ , konjunktiver Anfrage  $L$  und Projektionsvariablen  $x$  der Sicht

- Prinzip
  - 1 Anfrage  $Q$  in kanonische Form überführen
  - 2 Ersetzungsmöglichkeiten der MVs unter Anwendung der Regeln prüfen  $\rightsquigarrow$  alternative (äquivalente) Anfragen
  - 3 Anfrageoptimierung (Plan-Enumeration) zur kostenbasierten Auswahl

## Intuitiver Ansatz

- Finde Teilausdruck in  $Q$ , der zu  $L$  einer Regel  $L(x, y) \rightarrow V(x)$  korrespondiert und ersetze diesen durch  $V$
- **aber:**

Verkauf(produkt, stadt, preis, datum), Region(stadt, einw, land)  
 $\rightarrow$  RegVerkauf(produkt, preis, land)

- **Anfrage:**

$Q(\text{produkt})$  :- Verkauf(produkt, stadt, preis, datum), datum > 1.1.2007,  
Region(stadt, einw, 'Thüringen')

- **umgeschriebene Anfrage:**

$Q(\text{produkt})$  :- RegVerkauf(produkt, preis, 'Thüringen'),  
datum > 1.1.2007

## Problem syntaktischer Ersetzung

Verkauf(produkt, stadt, preis, datum), preis > 100  
→ TeureVerkäufe(produkt, stadt, preis)

$Q(\text{produkt})$  : – Verkauf(produkt, stadt, preis, datum), preis > 200,  
Region(stadt, einw, land)

- keine **syntaktische** Ersetzung möglich, jedoch

$Q(\text{produkt})$  : – TeureVerkäufe(produkt, stadt, preis), preis > 200,  
Region(stadt, einw, land)

- erfordert Auswertung der logischen Implikation

# Sichere Ersetzung

- Erweiterung der Regeln um Ungleichungs-Constraints

$$L(x, y), I(x) \rightarrow V(x)$$

mit  $I(x)$  ist Konjunktion von Constraints (inkl. Arithmetik) über Projektionsvariablen

- Bsp.:  $I(\text{produkt, stadt, preis}) \equiv \text{preis} > 100$
- Suche nach ersetzbaren Teilausdrücken erfordert Umbenennung der Variablen in der Regel
- Geg.: Regel  $r$  mit Variablen  $V_r$ , Anfrage  $Q$  mit Variablen  $V_Q$  und Konstanten  $C_Q$

gültige Umbenennung  $\sigma$  bzgl.  $Q$  ist Symbol-Mapping  $V_r \mapsto V_Q$  mit

- (a) wenn  $v \in V_r$  ist Projektionsvariable, dann  $\sigma(v) \in V_Q \cup C_Q$
- (b) wenn  $v \in V_r$  keine Projektionsvariable, dann  $\sigma(v) \in V_Q \wedge \sigma(v) \neq \sigma(v')$  mit  $v' \in V_r \setminus \{v\}$

auch **Containment Mapping**

## Sichere Ersetzung /2

- Geg. Menge  $R$  von Regeln
- zu  $Q$  existiert sicheres Auftreten von  $R$ , wenn für Regel  $r \in R$  eine gültige Umbenennung existiert, so dass umbenannte Regel folgende Form hat

$$L(x, y), I(x) \rightarrow V(x)$$

- dabei muss gelten

(1) Anfrage  $Q$  hat Form

$$Q(u) \equiv L(x, y), I'(x), G(v)$$

$x, y, u, v$  sind Variablenmengen evtl. mit gemeinsamen Variablen,  $y$  ist disjunkt zu  $x, u, v$

(2)  $I'(x) \Rightarrow I(x)$

## Sichere Ersetzung /3

- **Sichere Ersetzung** ist dann

$$Q'(u) \equiv V(x), I'(x), G(v)$$

- $I'(x) \Rightarrow I(x)$  ist nicht trivial; in  $I'(x)$  bleibt nur Teil der Constraints
- **Anfrageäquivalenz bzgl. Menge von Rewriting-Regeln**

# Anfrageplanung

- 1 Erstellung einer Abbildungstabelle mit **allen** sicheren Ersetzungen zu einer Anfrage  $[\sigma(L), \sigma(V)] \equiv [deletelist, addliteral]$

Region(stadt, einw, land), einw > 100  $\rightarrow$  Großstadt(stadt, land)  
 Verkauf(produkt, stadt, preis, datum), Region(stadt, einw, 'Thüringen')  
 $\rightarrow$  ThürVerkauf(produkt, stadt, preis, einw)

$Q(\text{produkt}) : -$  Verkauf(produkt,stadt,preis,datum),  
 Region(stadt,einw,'Thüringen'), einw > 100

Abbildungstabelle:

({ Region(stadt, 'Thüringen', einw), einw > 100 },  
 Großstadt(stadt,'Thüringen'))  
 ({ Verkauf(produkt, stadt, preis, datum), Region(stadt, einw,  
 'Thüringen') }, ThürVerkauf(produkt, stadt, preis, einw))

- 2 kostenbasierte Auswahl als Teil der Plan-Enumeration

# Erweiterung um Aggregationen: Generalized Projections

- verallgemeinerte Projektionen (Gupta, Harinarayan, Quass, 1995)
- Idee:

```
SELECT A FROM R GROUP BY A
```

kann äquivalent ersetzt werden durch

```
SELECT DISTINCT A FROM R
```

- entspricht Projektion  $\pi$  der Relationenalgebra
- muss noch für die Behandlung von Aggregationsfunktionen erweitert werden:

$$\pi_{A_1, \dots, A_n, \mathbf{AGG}(S)}(R)$$

# Generalized Projections

- damit Reduktion des Umformungsproblem auf die Relationenalgebra (mit erweiterter Projektion  $GP$ )
- ferner: Überführung aller Anfrageausdrücke (einschl. der der materialisierten Sichten) in eine Normalform

$$\sigma_h \pi \sigma_l \chi$$

mit generalisierten Projektionen  $\pi$ , Verbundoperationen  $\chi$  und konjunktiven Selektionsbedingungen  $h$  und  $l$

- Anwendung von Transformationsregeln
  - ▶ Push-down von GPs
  - ▶ Pull-up von GPs
  - ▶ Verschmelzen bzw. Aufteilen von GPs
- dabei Berücksichtigung der Duplikatsensitivität (**DISTINCT**, **MAX**, **MIN** vs. **SUM**, **COUNT**)

# Rewriting-Algorithmus

Geg.: Anfrage  $Q$ , Sicht  $V$ ; Ges.:  $Q'(V) = Q$

- (1) Push-down von  $\sigma_h$  zu  $\sigma_l$  für  $Q$  und  $V$
- (2) wenn Bedingung in  $\sigma_l(V)$  restriktiver als  $\sigma_l(Q) \rightarrow$  Abbruch
  - (a) Prüfe ob  $GP(Q)$  und  $GP(V)$  die gleichen Gruppierungskomponenten haben; falls Komponenten von  $GP(Q) \not\subseteq$  der Komponenten von  $GP(V) \rightarrow$  Abbruch  
wenn  $\sigma_h(V) = \emptyset$  dann Split von  $GP(Q)$  in  $GP_{bot}(Q)$   
(Gruppierungskomponenten von  $GP(V)$ ) und  $GP_{top}(Q)$   
(Berechnungen von  $GP(Q)$ )
  - (b) Prüfe ob Aggregate aus  $GP_{bot}(Q)$  aus Aggregaten von  $GP(V)$  ableitbar sind; anderenfalls  $\rightarrow$  Abbruch
- (3) Bedingungen in  $\sigma_l(Q)$ , die nicht aus  $\sigma_l(V)$  impliziert werden, aus  $GP_{bot}(Q)$  herausziehen; falls nicht möglich  $\rightarrow$  Abbruch
- (4) wenn Bedingungen in  $\sigma_h(V)$  restriktiver als  $\sigma_h(Q) \rightarrow$  Abbruch
- (5) Ergebnis: transformierte Anfrage  $Q$  mit Teilbaum  $GP_{bot}(Q)$  ersetzt durch  $V$

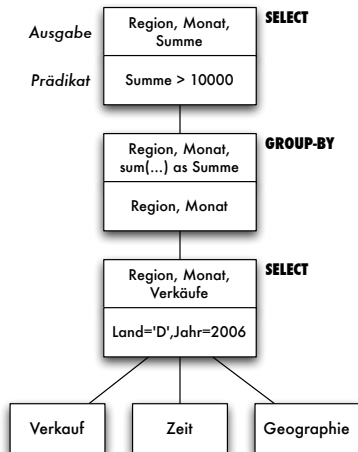
# Query Graph Model

- in DB2 verwendete Lösung
- Anfrage repräsentiert als DAG mit
  - ▶ Basistabellen als Blätter
  - ▶ Tabellenoperatoren als interne Knoten
- Knoten = Box
- Knotentypen
  - ▶ **SELECT**: Select-Project-Join-Teil einer Anfrage (inkl. **WHERE**- und **HAVING**-Bedingungen)
  - ▶ **GROUP-BY**: Gruppierung und Aggregation

# QGM: Beispiel

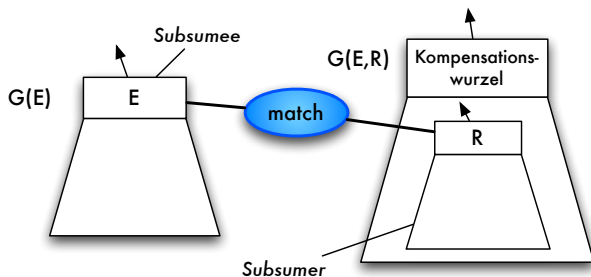
```

SELECT G.Region, Z.Monat,
        SUM(Verkaeufe)
FROM Verkauf V, Zeit Z,
        Geographie G
WHERE V.Zeit_ID = Z.Zeit_ID
        AND V.Geo_ID = G.Geo_ID
        AND G.Land = 'D'
        AND Z.Jahr = 2006
GROUP BY G.Region, Z.Monat
HAVING SUM(Verkaeufe) > 10000
  
```



# Query Matching

- Idee: Übereinstimmung zwischen Paaren von QGM-Box
  - ▶  $E$  matches  $R \Leftrightarrow$  QGM-Graph  $G(E, R)$  kann gebildet werden mit
    - ★  $G(E, R)$  beinhaltet  $G(R)$  mit Wurzel  $R$  und
    - ★  $G(E, R)$  ist äquivalent zu  $G(E)$
  - ▶  $G(E, R) - G(R)$  ist **Kompensation**: Anwendung auf  $R$  liefert gleiches Ergebnis wie  $E$



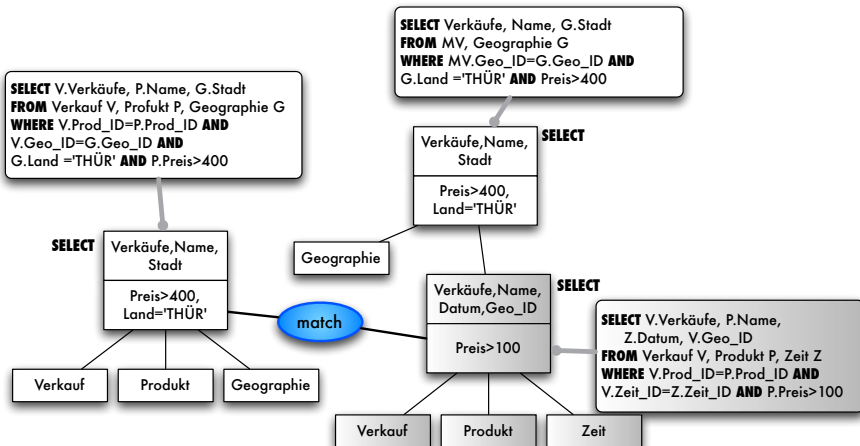
## Query Matching /2

- Betrachtung von Graphmustern zum Matching
  - ▶ Bottom-Up-Durchlauf des Query-Graphen
- grundsätzlich
  - 1 mind. eines der Kinder eines Subsumee muss mit einem Subsumer-Kind übereinstimmen
  - 2 Subsumer und Subsumee müssen vom selben Typ sein
- Muster:
  - ▶ **exact child matches**
  - ▶ **non-exact child matches**: Berücksichtigung der Kompensationen bei Nichtübereinstimmungen
  - ▶ **cube matches**

## Exact Child Matches

- **SELECT**:  $E$  (Subsumee) und  $R$  (Subsumer) sind **SELECT**-Boxen mit 1:1-Übereinstimmung der Kinder
  - ▶ jedes  $E$ -Kind stimmt mit max. einem  $R$ -Kind überein
  - ▶ keine zwei  $E$ -Kinder stimmen mit dem selben  $R$ -Kind überein (kein Self-Join!)
  - ▶ zusätzliche Verbunde sind verlustlos
  - ▶ Ausgabeattribute von  $E$  sind aus  $R$  ableitbar
  - ▶  $R$ -Prädikate, die nicht Extra-Verbundprädikate sind, sind weniger einschränkend als korrespondierende  $E$ -Prädikate, z.B.  $R: x > 10$  und  $E: x > 20$
  - ▶ Kompensation:
    - ★ Rejoin der Kinder
    - ★ **SELECT**-Box mit Rejoin-Prädikat und Anwendung der  $E$ -Prädikate, die nicht für  $R$  existieren

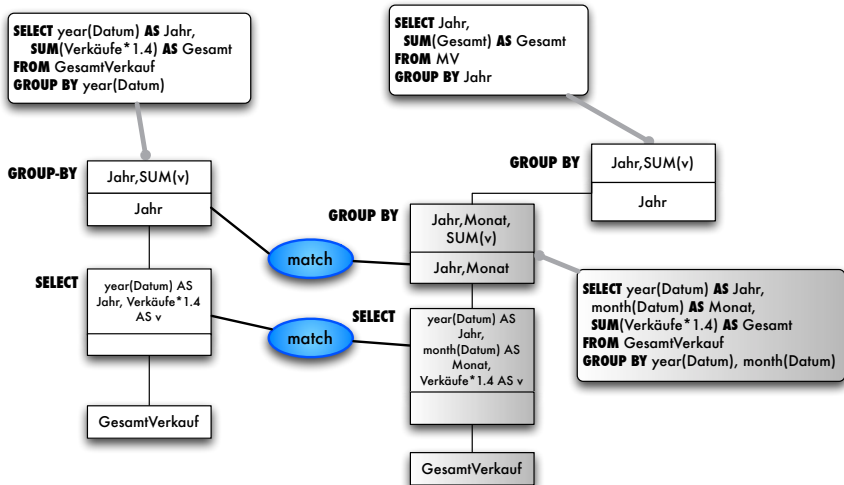
# Exact Child Matches: Beispiel



## Exact Child Matches /2

- **GROUP BY:**  $E$  (Subsumee) und  $R$  (Subsumer) sind **GROUP-BY-Boxen** und exakte Übereinstimmung ihrer Kinder
  - ▶ jedes  $E$ -Gruppierungsattribut ist äquivalent zu einem  $R$ -Gruppierungsattribut
  - ▶ wenn Gruppierungsattribute exakt übereinstimmen  $\rightarrow$   $E$ -Aggregate stimmen mit  $R$ -Aggregaten überein
  - ▶ sonst  $\rightarrow$  jedes  $E$ -Aggregat ist aus einem  $R$ -Aggregat ableitbar
  - ▶ Kompensation:
    - ★ spezielle Ableitungsregeln für Aggregatfunktionen

# Exact Child Matches: Beispiel /2



# Auswahl materialisierter Sichten

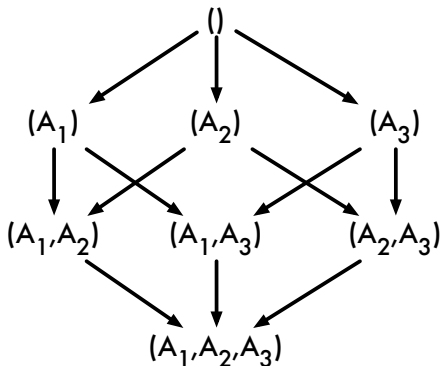
- erfordert den jeweiligen Auswertekontext für Anfragen mit Aggregationsfunktionen:
  - ▶ Eigenschaften der Aggregationsfunktionen: **Additivität** (z.B. **SUM**) bzw. Semi-Additivität (z.B. **COUNT**)
  - ▶ Berücksichtigung der Menge der Gruppierungsattribute, da diese eine Partitionierung des Datenbestands festlegen:  
**Aggregationsgitter**

# Aggregationsgitter

- Aggregationsgitter repräsentiert Teilmengenverband über die Gruppierungsattribute ([aggregation lattice](#))
- Pfeile geben an, welche Aggregationen bzw. Gruppierungen aus welchen anderen berechnet werden können, z.B:
  - ▶ Gruppierung nach  $A_2$  kann aus der Gruppierungen nach  $(A_1, A_2)$ ,  $(A_2, A_3)$ ,  $(A_1, A_2, A_3)$  berechnet werden
- Zahl der Knoten im Aggregationsgitter wächst exponentiell mit der Zahl der Gruppierungsattribute (Dimensionen)

## Aggregationsgitter: Beispiel

- 3 Gruppierungsattribute  $A_1, A_2, A_3$  (z.B. Monat, Region, Produktgruppe)



## Aggregationsgitter /2

- jeder Knoten im Aggregationsgitter entspricht einer Möglichkeit, eine materialisierte Sicht zu bilden
  - ▶ aufgrund der exponentiellen Anzahl an potentiell materialisierbaren Sichten muss Auswahl getroffen werden (aus Speicherplatz- sowie Aufwandsgründen)
  - ▶ oder: das Aggregationsgitter muss reduziert werden
- funkt. Abhängigkeiten zwischen Gruppierungsattributen erlauben Reduktion des Aggregationsgitters
  - ▶ wenn  $A_1 \rightarrow A_2$ , repräsentieren Knoten  $(A_1)$  und  $(A_1, A_2)$  sowie  $(A_1, A_3)$  und  $(A_1, A_2, A_3)$  jeweils das gleiche
  - ▶ funktionale Abhängigkeiten bestehen z.B. zwischen den verschiedenen Ebenen einer Klassifikationshierarchie:  
Produkt  $\rightarrow$  Produktgruppe  $\rightarrow$  Produktkategorie

## Statische Auswahl material. Sichten

- Algorithmus nach Harinarayan, Rajaraman und Ullman; SIGMOD'96
- berücksichtigt das historische (aber nicht das aktuelle) Anfrageverhalten → im Gegensatz zu dynamischen Verfahren
- notwendige Vorüberlegungen zu Aufwand und Nutzen materialisierter Sichten:
  - ▶ Kosten einer Materialisierungskonfiguration
  - ▶ Nutzen einer Materialisierungskonfiguration

# Kostenfunktion

- **Kostenfunktion**  $c_q(n)$ 
  - ▶ liefert Kosten zur Berechnung der Anfrage  $q$  aus dem zu  $n$  korrespondierenden Gitterpunkt im Aggregationsgitter, falls  $q$  aus der zu  $n$  gehörenden Aggregationskombination berechenbar ist
  - ▶ andernfalls  $c_q(n) = \infty$
- Anfragekostenfunktion  $c_q(n)$  ist **monoton**, wenn für zwei beliebige Knoten  $n_i, n_j$  aus einem Aggregationsgitter gilt:

$$|n_i| < |n_j| \Rightarrow c_q(n_i) < c_q(n_j)$$

(mit  $|n|$  Größe des Gitterpunkts)

## Kostenfunktion /2

- Monotonie besagt:
  - ▶ Kosten zur Beantwortung einer Anfrage bzgl. einer Aggregationskombination steigen mit der Größe der zu verarbeitenden Einheiten dieser Kombination an
  - ▶ Berechnungskosten einer Anfrage basierend auf einem Knoten  $n_i$  sind kleiner als die zu  $n_j$ , falls  $n_j$  aus  $n_i$  abgeleitet werden kann

# Kosten einer Materialisierungskonfiguration

- Menge der materialisierten Gitterpunkte  $M = \{n_1, \dots, n_k\}$ ; Menge von Anfragen  $Q = \{q_1, \dots, q_m\}$  (mit  $f(q_i)$  Häufigkeit von  $q_i$ )
- Aufwand zur Beantwortung einer Anfrage  $q$  bzgl.  $M$ :

$$c_q(M) = \min_{n \in M} c_q(n)$$

- Gesamtanfragekosten für  $Q$  basierend auf  $M$ :

$$C(Q, M) = \sum_{q \in Q} f(q) \cdot c_q(M)$$

- Gesamtaktualisierungskosten bei lokalen Aktualisierungskosten  $u(n)$  und Aktualisierungsrate  $h(n)$  für einen materialisierten Gitterpunkt  $n$ :

$$U(M) = \sum_{n \in M} h(n) \cdot u(n)$$

- Gesamtkosten einer Gruppierungskombination zur Auswertung einer Menge von Anfragen:

$$C_{\text{gesamt}}(Q, M) = C(Q, M) + U(M)$$

# Nutzen einer Materialisierungskonfiguration

- Sei
  - ▶  $M$  die Menge der bereits materialisierten Gitterknoten und
  - ▶  $Q$  eine Menge von Anfragen
- Nutzen (benefit) eines zusätzlich materialisierten Gitterknotens  $n$

$$B_Q(n, M) = \begin{cases} C_Q(M) - C_Q(M \cup \{n\}) & \text{falls } C_Q(M \cup \{n\}) < C_Q(M) \\ 0 & \text{sonst} \end{cases}$$

# Statisches Auswahlverfahren

- Auswahlproblem ist NP-vollständig!
- Verfahren von Harinarayan, Rajaraman und Ullman basiert auf dem Greedy-Prinzip
- Idee:
  - ▶ für ein vorgegebenes Aggregationsgitter wird iterativ immer der Knoten bestimmt, dessen Materialisierung den maximalen Nutzen bezgl. der Menge der schon zur Materialisierung ausgewählten Knoten bringt
  - ▶ dabei ist zu berücksichtigen, dass der zur Materialisierung zur Verfügung stehende Speicherplatz beschränkt ist

## Auswahl-Algorithmus

- Eingabe: Menge aller Gitterknoten  $N$ ; erwarteter Speicheraufwand  $|n|$  bei Materialisierung von Knoten  $n \in N$ ; maximaler Speichermehraufwand  $S$
- Ausgabe: Menge der zu materialisierenden Knoten  $M$

**begin**

$M = \{n_{Detaildaten}\}$  // Detaildaten sind bereits „materialisiert“

$s = 0$  // noch kein zusätzlicher Speicheraufwand

**while** ( $s < S$ ) **and**  $M \neq N$

// berechne Gitterpunkt mit max. Nutzen

$n$  ist der Gitterpunkt mit  $n \in (N \setminus M)$  :

$$B_Q(n, M) = \max_{n_i \in (N \setminus M)} (B_Q(n_i, M))$$

$M = M \cup \{n\}$

$s = s + |n|$  // zusätzlich benötigter Speicher zur Materialisierung

**end**

**return**  $M$

**end**

# Analyse des Verfahrens

- Komplexität  $O(n^3)$  mit  $n$  Anzahl der Gitterknoten
- Aber Verbesserungen möglich, z.B.:
  - ▶ bei Existenz funktionaler Abhängigkeiten zwischen Gruppierungsattributen reduziert sich die Zahl zu betrachtender Knoten
  - ▶ Menge der zu betrachtenden Anfragen erlaubt schrittweise Konstruktion des Aggregationsgitters, so dass nur Knoten betrachtet werden, die von Anfragen benötigt werden
  - ▶ Knoten, die eine bestimmte Reduktionsrate (bezogen auf das Datenvolumen) gegenüber detaillierteren Knoten nicht erreichen, werden a priori von der Materialisierung ausgeschlossen und im Algorithmus nicht betrachtet

## Analyse des Verfahrens /2

- Untere Schranke für Güte der schlechtesten Lösung: 63% gegenüber der optimalen Lösung (Angabe der Autoren; bezogen auf ursprünglichen Vorschlag)
- Ursprüngliches Verfahren berücksichtigt kein explizites Anfrageprofil; Annahme, dass jeder Knoten gleich häufig angefragt wird
- Verfahren berücksichtigt Restriktionsbedingungen, die in den Anfragen auftreten, in keiner Weise

## Auswahl von Partitionen materialisierter Sichten

- Nicht alle Daten eines Aggregationsknoten werden gleich häufig angefragt, z.B.
  - ▶ Aktuelle Daten werden zumeist häufiger angefragt, etwa Verkäufe im letzten Monat
  - ▶ Wenn der Datenbestand z.B. 24 Monate umfasst und 80% der Anfragen nur den letzten Monat betreffen, würde die vollständige Materialisierung unverhältnismäßig viel Speicher benötigen
  - ▶ Bei Partitionierung nach Monaten und nur Materialisierung der Daten des letzten Monats würden mit 4 – 5% des Speicheraufwands 80% der Anfragen unterstützt

## Auswahl von Partitionen /2

- Zur Bestimmung, welche Partitionen materialisiert werden sollen, kann der zuvor behandelte Algorithmus erweitert werden
- Je nach Anfrageprofil können für jeden Aggregationsknoten unterschiedliche Partitionierungen (sowohl horizontal als auch vertikal) mit unterschiedlicher Auswahl zu materialisierender Partitionen bestimmt werden
- Spezialfall: vollständige Materialisierung einzelner Knoten

# Dynamische Auswahl materialisierter Sichten

- Statische Auswahl nimmt an, dass ein festes, vorher bekanntes Anfrageprofil existiert
  - ▶ basiert damit in der Regel auf dem Anfrageverhalten der Vergangenheit
  - ▶ interaktive Komponente von OLAP-Anwendungen wird nicht berücksichtigt
    - ★ oft werden in einem iterativen Prozess Anfragen immer weiter konkretisiert, um bestimmte Zusammenhänge näher zu untersuchen
    - ★ Anfrageverhalten kann dadurch spontanen Änderungen unterliegen
- Änderungen der Daten führen zu einem schnellen Veralten materialisierter Sichten

# Semantisches Caching

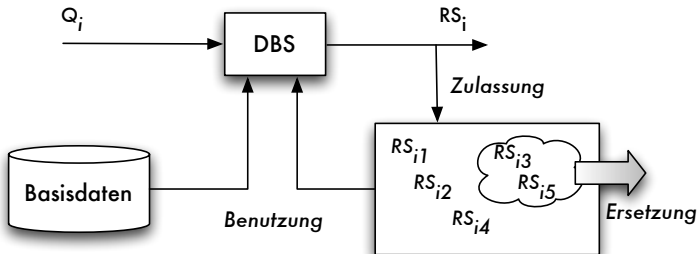
- Pufferung (Zwischenspeichern) von Anfrageergebnissen
  - ▶ im Hauptspeicher oder
  - ▶ auf einer schnellen Platte
- Beschränkung des dafür verfügbaren Speichers bedingt Verdrängung
  - ▶ Entscheidung über Verdrängung kann datenabhängig getroffen werden
  - ▶ Wissen über das Anwendungsgebiet (hier: OLAP und dafür typisches Anfrageverhalten) kann ausgenutzt werden

# Verdrängung materialisierter Sichten

- Mögliche Entscheidungskriterien:
  - ▶ Zeit des letzten Zugriffs
  - ▶ Referenzierungshäufigkeit
  - ▶ Größe der materialisierten Sicht
  - ▶ Kosten einer Neuberechnung oder Aktualisierung der materialisierten Sicht
  - ▶ Anzahl der Anfragen, die in der Vergangenheit mit dieser Sicht hätten beantwortet werden können (oder wurden)
  - ▶ Anzahl der Anfragen, die prinzipiell mit dieser Sicht beantwortet werden könnten
- Gewichtete Kombination dieser Kriterien sinnvoll!

## Beispiel: Watchman

- DW-Cache Manager [Scheuermann, Shim, Vingralek; VLDB'96]



## Watchman: Ablauf

- 1 Ergebnis  $RS_i$  zu Anfrage  $Q_i$  an Nutzer und Cache
- 2 Aufnahme in Cache anhand LNC-A-Verfahren („least normalized cost – admission“) prüfen
- 3 Falls zur Materialisierung zugelassen und nicht genug Speicher: zu ersetzende Ergebnisse ermitteln mithilfe von LNC-R („replacement“)

## Watchman: Kennzahl

- Kennzahl für Zulassung/Ersetzung: Gewinn (profit) einer Materialisierung

$$P(RS_i) = \frac{\lambda \cdot C(Q_i)}{|RS_i|}$$

- Mit
  - ▶  $\lambda$ : durchschnittliche Referenzierungsrate von  $Q_i$
  - ▶  $C(Q_i)$ : Kosten von  $Q_i$  ohne Materialisierungen

# Watchman: LNC-A

Eingabe:  $RS_i$  // Ergebnis der Anfrage  $Q_i$

**begin**

**if** (Referenzinfo  $R_i$  noch nicht verfügbar)

    init\_ref\_info ( $R_i$ )

**end**

  update\_ref\_info ( $R_i$ )

  // Zulassung, falls Gewinn von  $RS_i$  größer als

  // Gewinn der zu verdrängenden Ergebnisse

$C := \text{LNC-R}(|RS_i|)$

**if** ( $P(RS_i) > P(C)$ )

    remove\_from\_cache ( $C$ )

    add\_to\_cache ( $RS_i$ )

**end**

**end**

# Watchman: LNC-R

Eingabe:  $S$  // freizugebender Speicherplatz

Ausgabe:  $C$  // zu ersetzende Ergebnisse

**begin**

$C := \{\}$

**for**  $i = 1$  **to**  $n$

// Ergebnisse  $RS_j$  pro Gruppe mit gleicher

// Ref.häufigkeit nach Gewinn  $P(RS_j)$  aufsteigend

// sortieren

$rs\_list = \text{sort\_rs\_by\_profit}(i)$

**foreach**  $RS_j$  **in**  $rs\_list$

$C := C + \{RS_j\}$

**if**  $(|C| > S)$  **return**  $C$

**end**

**end**

**return**  $C$

**end**

# Aktualisierung materialisierter Sichten

- Effiziente Aktualisierungsverfahren benötigt, die teure Rematerialisierung (Neuberechnung) vermeiden  $\rightsquigarrow$  **inkrementelle Aktualisierung**
- **Konsistenzproblem**, da materialisierte Sichten im allgemeinen nicht unabhängig voneinander sind
  - ▶ redundante Speicherung von Daten (oder daraus abgeleitete Aggregationen) in mehreren Sichten
  - ▶ Aktualisierung einer Sicht führt zur Inkonsistenz
  - ▶ Aktualisierung mehrerer Sichten benötigt ggf. viel Zeit, so dass zwischendurch ein inkonsistenter Zustand gesehen werden kann

# Vollständige Aktualisierung

- Ansatz: gemeinsame Aktualisierung mehrerer Sichten (multi-query optimization)
- Idee:
  - 1 Ableitung eines gemeinsamen Vorgängers (Basisanfrage) mit minimaler Extension
  - 2 Berechnung beider Sichten auf Basis des Vorgängers

# Einschränkungen

- Anpassung der Ergebnisattribute
  - ▶ Basisanfrage muss alle Attribute liefern, die in Sichten referenziert werden (Projektion, Selektion, Gruppierung, Aggregation etc.)
- Anpassung der referenzierten Basisrelationen
  - ▶ Basisanfrage muss gleiche Menge von Basisrelationen beinhalten (Ausnahme: verlustlose 1:n-Joins, Rückwärtsverbund)
- Anpassung der Prädikate
  - ▶ Gemeinsame globale Prädikate in der Basisanfrage auswerten
- Anpassung der Gruppierungskombinationen
  - ▶ aus Gruppierung der Basisanfrage müssen alle Kombinationen der Sichten ableitbar sein

## Beispiel [Lehner02]

```
CREATE MATERIALIZED VIEW matview1 AS  
SELECT p_brand, o_shipprio,  
        SUM(l_quantity), COUNT(*)  
FROM lineitem, orders, part  
WHERE l_orderkey = o_orderkey AND  
        l_partkey = p_partkey AND  
        year(l_shipdate) = 2002 AND p_type = 'spc'  
GROUP BY CUBE(p_brand, o_shipprio)
```

```
CREATE MATERIALIZED VIEW matview2 AS  
SELECT p_container, o_shipprio,  
        AVG(l_quantity), COUNT(*)  
FROM lineitem, orders, part  
WHERE l_orderkey = o_orderkey AND  
        l_partkey = p_partkey AND  
        year(l_shipdate) = 2002  
GROUP BY p_container, o_shipprio
```

## Beispiel: Basisanfrage

```
SELECT p_brand, p_container, p_type /* lokales Prädikat */,  
        o_shipprio,  
        SUM(l_quantity) AS $1, COUNT(*) AS $2  
FROM lineitem, orders, part  
WHERE l_orderkey = o_orderkey  
        AND l_partkey = p_partkey  
        /* globales Prädikat */  
        AND year(l_shipdate) = 2002  
GROUP BY GROUPING SETS((p_container, o_shipprio),  
                          /* lokales Prädikat */  
                          (p_type, cube(p_brand, o_shipprio)))
```

## Beispiel: Aktualisierung für Sichten

```
INSERT INTO matview1 (  
  SELECT p_brand, o_shipprio, SUM($1), SUM($2)  
  FROM $basisanfrage  
  WHERE p_type = 'spc'  
  GROUP BY CUBE(p_brand, o_shipprio))
```

```
INSERT INTO matview2 (  
  SELECT p_container, o_shipprio,  
    SUM($1)/SUM($2) /* AVG(l_quantity) */, SUM($2)  
  FROM $basisanfrage  
  GROUP BY p_container, o_shipprio)
```

## Ebenen gemeinsamer Vorauswertung

- Bildung gemeinsamer Gruppierungskombinationen
- Materialisierung von Verbundoperationen
- Materialisierung von Selektionsoperationen
- Gemeinsame Nutzung auf physischer Ebene (gemeinsame Datenblöcke etc.)

# Inkrementelle Aktualisierung

- Wesentliche Idee:
  - ▶ Feststellen, welche Detaildaten sich geändert haben
  - ▶ nur diese Änderungen in den materialisierten Sichten nachvollziehen
- Im einfachsten Fall (ohne Aggregation):
  - ▶ Basisrelation  $R$ , Sicht  $V = f(R)$
  - ▶  $\Delta^+R$  neu eingefügte Tupel,  $\Delta^-R$ : gelöschte Tupel (Updates werden als Löschung und anschließende Einfügung betrachtet)

$$\begin{aligned}V_{neu} &= f((R - \Delta^-R) \cup \Delta^+R) \\ &= (f(R) - f(\Delta^-R)) \cup f(\Delta^+R) \\ &= (V - \Delta^-V) \cup \Delta^+V\end{aligned}$$

## Beispiel

- **Basisrelation:**  $\text{link}(S, D)$   
 $\text{link}(a, b) \rightarrow$  Verbindung zw. Knoten  $a$  und  $b$
- **Sicht:**

$$\text{hop}(X, Y) = \pi_{X,Y}(\text{link}(X, V) \bowtie_{V=W} \text{link}(W, Y))$$

- Menge der eingefügten Tupel in  $\text{link}$ :  $\Delta(\text{link})$
- Korrespondierende Menge für  $\text{hop}$ :  $\Delta(\text{hop})$

$$\begin{aligned} \Delta(\text{hop}) = & \pi_{X,Y}((\Delta(\text{link})(X, V) \bowtie_{V=W} \text{link}(W, Y)) \cup \\ & (\text{link}(X, V) \bowtie_{V=W} \Delta(\text{link})(W, Y)) \cup \\ & (\Delta(\text{link})(X, V) \bowtie_{V=W} \Delta(\text{link})(W, Y))) \end{aligned}$$

# Klassifikation: Einflussfaktoren auf Effizienz

## ● Zur Verfügung stehende Information

- ▶ entscheidet über die Anwendbarkeit eines Algorithmus
- ▶ Definition der Sicht sowie ihre Ausprägung müssen zur Verfügung stehen
- ▶ Weitere Informationen? Integritätsbedingungen, Hilfssichten, ...

## ● **Verwendete Anfragekonstrukte** zur Definition der Sicht

- ▶ Selektion und Projektion im allgemeinen unproblematisch
- ▶ Viele Aktualisierungsalgorithmen unterstützen auch Joins
- ▶ Aggregationsfunktionen verlangen bereits aufwendigere Algorithmen

# Klassifikation: Einflussfaktoren auf Effizienz /2

## ● **Unterstützte Modifikationstypen**

- ▶ Im einfachsten Fall: Einfügen neuer Tupel, Löschen von Tupeln
- ▶ Updates: Behandeln als Löschen und anschließendes Neueinfügen führt zu Informationsverlust und kann teure Zugriffe auf die Basisrelation erfordern

## ● **Granularität der Aktualisierung**

- ▶ Aktualisierung einzelner Sichten oder (im Extremfall) des ganzen Data Warehouse
- ▶ gemeinsame Aktualisierung verschiedener materialisierter Sichten kann für Konsistenz notwendig sein

# Klassifikation: Einflussfaktoren auf Effizienz /3

## ● Zeitpunkt der Aktualisierung

- ▶ Sofortige (synchrone) Aktualisierung
  - ★ Sichten sind immer aktuell
  - ★ dafür werden die Modifikationstransaktionen behindert
- ▶ Verzögerte Aktualisierung
  - ★ Entkoppelung der Aktualisierung von Modifikationstransaktion; bei Zugriff auf Sicht wird diese aktualisiert
  - ★ Sicht beim Lesen immer aktuell
  - ★ Lesende Transaktion trägt die Aktualisierungskosten; u.U. müssen viele Modifikationen nachgezogen werden, wenn auf die Sicht lange nicht zugegriffen wurde
- ▶ Snapshot-Aktualisierung
  - ★ asynchron zur Modifikation und zum Lesezugriff nach anwendungsspezifischen Gesichtspunkten

# Strategie vs. Zeitpunkt

<b>Aktualisierungsstrategie</b>			
complete	–	–	✓
incremental	✓	✓	–
	immediate	on commit	deferred
	<b>Aktualisierungszeitpunkt</b>		

# Beispiele /1

- Relation:

`part(part_no, part_cost, contract)`

- Sicht:

`expensive_parts(part_no) =  $\pi_{\text{part\_no}}(\sigma_{\text{part\_cost} > 1000}(\text{part}))$`

- Einfügen: `part( $p_1$ , 500,  $c_1$ )`
  - ▶ keine Änderung der Sicht notwendig

## Beispiele /2

- Einfügen `part(p2, 5000, c2)`

- (1) nur mat. Sicht verfügbar

- ★  $p_2$  in der alten Sicht vorhanden → keine Änderung
    - ★ sonst → Einfügen

- (2) Basisrelation verfügbar

- ★ Überprüfen, ob Basisrelation Tupel mit `part_cost ≥ 5000` enthält  
→ keine Änderung
    - ★ sonst → Einfügen

- (3) `part_no` als PK bekannt

- ★ Tupel kann noch nicht in der Sicht sein → Einfügen

## Beispiele /3

- **Relation:** `supp(supp_no, part_no, price)`
- **Sicht:**

$$\text{supp\_parts}(\text{part\_no}) = \pi_{\text{part\_no}}(\text{supp} \bowtie_{\text{part\_no}} \text{part})$$

- **Einfügen:**  $(p_2, 5000, c_2)$ 
  - ▶ nur mat. Sicht verfügbar:
    - ★ `supp_parts` enthält bereits  $p_2$  → keine Änderung
    - ★ sonst → kann nicht entschieden werden

# Zusatzinformationen für effiziente inkrementelle Aktualisierung

- Ziel: Vermeidung von Zugriffen auf die Basisrelationen → (partielle) autonome Aktualisierbarkeit
- Verwendung von Zusatzinformationen
  - ▶ Schemainformation: vor allem Primärschlüssel- und Fremdschlüsseleigenschaften
  - ▶ Zähler: Anzahl der Tupel in der Basisrelation, aus denen ein Tupel in der Sicht abgeleitet wurde (→ Counting-Algorithmen)
  - ▶ Hilfssichten: Materialisierung weiterer Information, die eine (partielle) autonome Aktualisierbarkeit ermöglicht

# Verfahren zur Aktualisierung

- Vollständige Information verfügbar
  - ▶ Counting-Algorithmus
  - ▶ Outer-Join-Sichten
  - ▶ DRed-Algorithmus
- Partielle Information verfügbar
  - ▶ „selbst-aktualisierbare“ Sichten: MV + Schlüsselbedingungen
  - ▶ „Chronicle Views“: MV + einige Basisrelationen
- Keine Information verfügbar
  - ▶ „Irrelevante“ Updates

# Counting-Algorithmus

- Idee: Anzahl der Ableitungen aller Tupel in der Sicht speichern
- Beispiel:

```
CREATE VIEW hop(s, d) AS  
  SELECT DISTINCT l1.s, l2.d  
  FROM link l1, link l2  
  WHERE l1.d = l2.s
```

- $\text{link} = \{(a, b), (b, c), (b, e), (a, d), (d, c)\}$
- $\text{hop} = \underbrace{\{(a, c)\}}_2, \underbrace{\{(a, e)\}}_1$

## Counting-Algorithmus /2

- Löschen von  $\text{link}(a, b) \rightsquigarrow \text{hop} = \{(a, c)\}$
- Schlussfolgerung: jeweils eine Ableitung von  $\text{hop}(a, c)$  und  $\text{hop}(a, e)$  löschen !
- Prinzip des Algorithmus
  - ▶ zu jedem Tupel  $t$  der Sicht  $\text{count}(t)$  speichern
  - ▶ aus Änderungen der Basisrelationen Sichtänderungen ableiten  $\Delta V$
  - ▶ Einfügen:  $\text{count}++$ , Löschen:  $\text{count}--$
  - ▶ Änderungen  $\Delta V$  in die Sicht einbringen und  $\text{count}$  aktualisieren
  - ▶ Tupel mit  $\text{count} = 0$  werden gelöscht

## Outer-Join-Sichten

- Outer Join wichtig für Datenintegration
- Sicht:

```
CREATE VIEW V AS  
  SELECT  $X_1, \dots, X_n$   
  FROM  $R$  FULL OUTER JOIN  $S$  ON  $g(Y_1, \dots, Y_m)$ 
```

- ▶  $g(Y_1, \dots, Y_m)$ : Konjunktion von Prädikaten
- ▶ Änderungen an  $R$ :  $\Delta(R)$  (Einfügen:  $\Delta^+(R)$ ; Löschen:  $\Delta^-(R)$ )

## Outer-Join-Sichten

- Algorithmus: Berechnung von  $\Delta(V)$  durch zwei Anfragen

```
SELECT  $X_1, \dots, X_n$  -- Anfrage (a)  
FROM  $\Delta(R)$  LEFT OUTER JOIN  $S$  ON  $g(Y_1, \dots, Y_m)$ 
```

```
SELECT  $X_1, \dots, X_n$  -- Anfrage (b)  
FROM  $R^v$  RIGHT OUTER JOIN  $\Delta(S)$  ON  $g(Y_1, \dots, Y_m)$ 
```

- Relation  $R^v$ :  $R$  nach Änderungen

## Outer-Join-Sichten

- Anfrage (a) berechnet Effekt der Änderungen an  $R$  auf Sicht  $V$
- dito (b) für  $S$
- Beispiel:  $r^+$  (Einfügen in  $R$ )
  - ▶ wenn kein Verbund mit Tupel  $s \rightarrow r^+ . \text{NULL}$  einfügen
  - ▶ sonst  $\rightarrow r^+ . s$  einfügen
- aber:
  - ▶  $r^+$ : Verbund mit  $s \rightarrow$  eventuell  $\text{NULL} . s$  aus  $V$  löschen, wenn bereits in  $V$
  - ▶  $r^-$ : eventuell Einfügen von  $\text{NULL} . s$  notwendig

# Selbst-aktualisierbare Sichten

- **Self-maintenable Views**: Aktualisierung nur auf Basis der materialisierten Sicht und Schlüsselbedingungen
- Beispiel:

$$\text{supp\_parts}(\text{part\_no}) = \pi_{\text{part\_no}}(\text{supp} \bowtie_{\text{part\_no}} \text{part})$$

`part_no` ist PK in `part`

- Löschen eines Tupels
  - ▶ wenn in der Sicht `part_no` vorhanden → dort löschen
  - ▶ selbst-aktualisierbar bzgl. **DELETE** in `part`
- Löschen von `supp(s1, p2, 100)`
  - ▶ `p2` kann nicht gelöscht werden, da z.B. `supp(s2, p2, 200)` existieren kann
  - ▶ nicht selbst-aktualisierbar bzgl. **DELETE** in `supp`
- generell: Sicht ist nicht selbst-aktualisierbar bzgl. **INSERT** in `supp` oder `part`

## Selbst-aktualisierbare Sichten /2

- Allgemein für Select-Project-Join (SPJ)-Sichten
  - ▶ SPJ-Sicht mit Join über zwei oder mehr verschiedene Relationen ist nicht selbst-aktualisierbar bzgl. **INSERT**
  - ▶ SPJ-Sicht ist selbst-aktualisierbar bzgl. **DELETE** in  $R$ , wenn das Schlüsselattribut  $a$  der Relation  $R$  entweder in der Sicht enthalten ist oder in einer Bedingung  $a = \text{const}$  der Sichtdefinition erscheint
- Sicht mit left oder full outer join der Relationen  $R$  und  $S$ 
  - ▶ ist selbst-aktualisierbar bzgl. aller Arten von Änderungen in  $S$ , wenn alle Attribute von  $R$ , die in einem Prädikat vorkommen auch in der Projektion der Sicht auftauchen

## Sichtaktualisierung in DW

- Basisrelationen stammen meist aus externen Quellen
- Quellen besitzen keine Informationen über Sichten
- Aktualisierung nur über Ereignismeldungen der Quellen an das DW möglich
- DW muss Quellen in geeigneter Weise abfragen

## Aktualisierung: Beispiel

- $r_1(W, X), r_2(X, Y), r_1 = \{(1, 2)\}, r_2 = \{(2, 4)\}$   
 $MV = \pi_W(r_1 \bowtie r_2), MV = \{(1)\}$
- ① Update  $U_1 = \mathbf{insert}(r_2, \{(2, 3)\}) \rightarrow$  Benachrichtigung des DW (Event)
- ② DW: Empfang von  $U_1 \rightarrow$  Algorithmus zur inkrementellen Sichtänderung sendet Anfrage

$$Q_1 = \pi_W(r_1 \bowtie \{(2, 3)\})$$

- ③ Quelle von  $r_1$  empfängt  $Q_1$  und liefert Ergebnis  $A_1 = \{(1)\}$
- ④ DW empfängt  $A_1$ , fügt  $\{(1)\}$  in  $MV$  ein und erhält  $\{(1), (1)\}$

## Aktualisierung: Anomalien

- Annahme:  $r_2 = \{\}$ ,  $MV = \{\}$   
 $U_1 = \mathbf{insert}(r_2, \{(2, 3)\})$ ,  $U_2 = \mathbf{insert}(r_1, \{(4, 2)\})$
- ① Quelle:  $U_1 = \mathbf{insert}(r_2, \{(2, 3)\}) \rightarrow U_1$  an DW
- ② DW empfängt  $U_1$  und sendet  $Q_1$
- ③ Quelle:  $U_2 = \mathbf{insert}(r_1, \{(4, 2)\}) \rightarrow U_2$  an DW
- ④ DW empfängt  $U_2$  und sendet  $Q_2 = \pi_W(\{(4, 2)\} \bowtie r_2)$
- ⑤ Quelle empfängt  $Q_1$  und berechnet auf  $r_1 = \{(1, 2), (4, 2)\}$  das Ergebnis  $A_1 = \{(1), (4)\}$
- ⑥ DW empfängt  $A_1$  und aktualisiert  $MV \cup A_1 = \{(1), (4)\}$
- ⑦ Quelle empfängt  $Q_2$  und berechnet auf  $r_1$  und  $r_2$  das Ergebnis  $A_2 = \{(4)\}$
- ⑧ DW empfängt  $A_2$  und aktualisiert  $MV \cup A_2 = \{(1), (4), (4)\}$   
 $\Rightarrow$  korrekt wäre  $\{(1), (4)\}$

# Lösungsansätze

- vollständige Neuberechnung der Sicht
  - ▶ Zeit- und Ressourcenintensiv
- Kopie aller Basisrelationen im Warehouse und „lokale“ Auswertung der Anfragen
  - ▶ zusätzlicher Speicherbedarf
  - ▶ Aktualisierung der Basisrelationen notwendig (Replikation)
- Hinzufügen von Kompensationsanfragen zu Quellenanfragen für Ausgleich konkurrierender Änderungen

# Eager-Compensating-Algorithmus

- Idee: Kompensationsanfragen für Anfragen, die auf einem „inkorrekten“ Zustand ausgeführt werden
- Beispiel:  $U_2 = \mathbf{insert}(r_1, \{(4, 2)\})$ 
  - ▶ mit  $U_2$  kann geschlussfolgert werden, dass  $Q_1$  auf einen „inkorrekten“ Zustand (Quelle  $\neq$  DW) ausgeführt wird:  $Q_1$  sieht  $\{(4, 2)\}$ , sonst würde  $A_1$  vor  $U_2$  eintreffen
  - ▶ Kompensationsanfrage

$$Q_2 = \pi_W(\{(4, 2)\} \bowtie r_2) - \pi_W(\{(4, 2)\} \bowtie \{(2, 3)\})$$

# Konsistenz während der Aktualisierung

- Problem: **Abhängigkeiten zwischen Sichten**
- Aktualisierung einer Sicht führt dann zur Inkonsistenz zwischen Sichten
- erfordert gemeinsame Aktualisierung, so dass keine Anwendung inkonsistenten Zustand sieht
  - ▶ Sperren aller Sichten während der Aktualisierung: bei großer Anzahl zu aktualisierender Sichten führt dies zu inakzeptabel langen Wartezeiten
  - ▶ Alternative: Vorberechnung aller Aktualisierungen und abschließende Übernahme der Änderungen in die Sichten (**Propagate and Refresh**)

# Konsistenz in Data Warehouses

- Anwenderdefinierte Aktualitätsforderungen als
  - ▶ Zeitlicher Abstand
    - ★ Zeitspanne, um die die Sicht älter sein darf als die Basisrelation
  - ▶ Wertemäßiger Abstand
    - ★ Maximale absolute oder relative Abweichung (z.B. bei Aggregationen) zwischen Sicht und Basisdaten
  - ▶ Versionsbezogener Abstand
    - ★ Zahl der Versionen die nach dem Aktualitätsstand der Sicht generiert wurden

# Anfragekonsistenz

- Ergebnis einer Anfrage an das DW hätte auch durch eine Anfrage an die Datenquellen berechnet werden können
- dies verlangt Konsistenz zwischen den Sichten, d.h. eine Anfrage an das DW darf nie durch Sichten mit unterschiedlichem Aktualitätsstand beantwortet werden
- Beachte: Dies bedeutet nicht notwendigerweise, dass alle zur Beantwortung verwendeten Sichten aktuell sein müssen

# Sitzungskonsistenz

- **Sitzung**: Folge von Anfragen, insbesondere Folgen von Drill-Down- und Roll-Up-Operationen
- Ergebnisse solcher nacheinander gestellten Anfragen sollten untereinander konsistent sein
- wenn alle Anfragen innerhalb einer Sitzung durch Daten (aus Basisrelationen und Sichten) gleicher Aktualität beantwortet werden, ist Sitzungskonsistenz gewährleistet
- Sitzungskonsistenz ist damit mehr als Anfragekonsistenz!

# Aktualisierungsgranulate

- Welche Objekte des Data Warehouse können unabhängig voneinander aktualisiert werden?
  - ▶ Aktualisierung des ganzen Data Warehouses
    - ★ Konzeptionell die einfachste Lösung
    - ★ zur Vermeidung von Inkonsistenzen während der Aktualisierung wird das gesamte Data Warehouse gesperrt (keine Nebenläufigkeit; ggf. lange Wartezeit)
  - ▶ Aktualisierung einzelner Sichten
    - ★ Flexible Lösung
    - ★ Abhängigkeiten zwischen den Sichten müssen berücksichtigt werden; dies kann Aktualisierung anderer Sichten erzwingen
  - ▶ Gemeinsame Aktualisierung mehrerer Sichten
    - ★ Zusammenfassen von Sichten zu einem Aktualisierungsgranulat als praktikabler Kompromiss (→ Viewgroups)

# Materialisierte Sichten in Oracle

- Syntax

```
CREATE MATERIALIZED VIEW matview  
BUILD IMMEDIATE  
REFRESH COMPLETE  
ENABLE QUERY REWRITE  
AS SELECT ...FROM ...WHERE ...GROUP BY ...
```

- „Füllen“ der Sicht

- ▶ **BUILD IMMEDIATE** (sofort)
- ▶ **BUILD DEFERRED** (explizit zum späteren Zeitpunkt)

- Aktualisierungszeitpunkte:

- ▶ **REFRESH ON COMMIT** (Änderung der Basisrelation),
- ▶ **REFRESH ON DEMAND** (explizite Aktualisierung, z.B. über `dbms_mview.refresh`)

# Materialisierte Sichten in Oracle /2

- Aktualisierungsstrategie
  - ▶ **COMPLETE**: vollständig
  - ▶ **FAST**: Deltas über View-Log
  - ▶ **NEVER**: keine Aktualisierung
  - ▶ **FORCE**: wenn möglich **FAST**, sonst **COMPLETE**
- View-Logs: Log-Tabellen mit Änderungsoperationen (über Trigger auf Basisrelationen)

```
CREATE MATERIALIZED VIEW LOG ON base-table  
WITH SEQUENCE, ROWID (attributes)  
INCLUDING NEW VALUES
```

# Materialisierte Sichten in Oracle /3

- Restriktionen für **FAST** (u.a.)
  - ▶ Alle Basisrelationen mit View-Logs
  - ▶ Zu jedem Aufruf **AGG** (*expr*) korrespondierendes **COUNT** (*expr*)
  - ▶ alle Gruppierungsattribute in **SELECT**-Klausel
  - ▶ Einschränkungen bei **OUTER JOIN** sowie komplexeren Gruppierungen

## Materialisierte Sichten in Oracle /4

- Verwendung existierender Tabellen als materialisierte Sicht

```
CREATE MATERIALIZED VIEW matview  
ON PREBUILT TABLE sum_table ...
```

- Summary Advisor als Administrator-Werkzeug für Auswahl von materialisierten Sichten

# Oracle: Rewriting

- Oracle versucht Rewriting auf mehrere Arten
- Text Match
  - ▶ „Full Text Match“ und „Partial Text Match“
  - ▶ Rein syntaktischer Match
    - ★ Groß-/ Kleinschreibung
    - ★ Leerzeichen
    - ★ Reihenfolge von Bedingungen
  - ▶ Keine logische Implikation, kein Mapping, etc.
- General Query Rewrite
  - ▶ Nicht anwendbar bei „Complex materialized views“
  - ▶ Unterschiedliche Methoden ja nach Art des MV
  - ▶ Viele Einschränkungen → Dokumentation

# Materialisierte Sichten in IBM DB2

- Syntax

```
CREATE SUMMARY TABLE matview AS (  
SELECT ...FROM ...WHERE ...GROUP BY ...  
) DATA INITIALLY DEFERRED REFRESH DEFERRED
```

- Explizites „Füllen“ der mat. Sicht

```
REFRESH TABLE matview
```

- Weitere Aktualisierungsstrategien möglich

# SQL Server

- Indexierung von Sichten
- Materialisierung der betroffenen Daten
- Automatische Aktualisierung bei Änderung der Basisdaten

```
CREATE VIEW Verkäufe2002 AS  
SELECT Stadt, Verkäufe, V.ZeitID, V.GeographieID  
FROM Verkauf V, Zeit Z, Geographie G  
WHERE V.ZeitID = Z.ZeitID AND Z.Jahr = 2002  
      AND V.GeographieID = G.GeographieID;
```

```
CREATE UNIQUE CLUSTERED INDEX V2000_IDX  
  ON Verkäufe2002(ZeitID, GeographieID);
```

# Zusammenfassung

- Materialisierung von Anfrageergebnissen zur mehrmaligen Nutzung und zur Beschleunigung der Anfrageverarbeitung
- Fragestellungen
  - ▶ Nutzung für Anfragen (Query Containment)
  - ▶ Auswahl (Optimierungsproblem)
  - ▶ Aktualisierung
- Unterstützung in kommerziellen DBMS