Universität Kaiserslautern
Fachbereich Informatik
AG DBIS

Seminar

Materialisierte Sichten

vorgelegt von
Sebastian Benz

Betreuer
Boris Stumm

Kaiserslautern, den 10.07.2003

Inhaltsverzeichnis

Einleitung

Kaum ein Unternehmen kommt noch ohne eigene Datenbank aus, in der alle betrieblichen Informationen gespeichert werden, um später zwecks Analyse auf diese zugreifen zu können. Das Problem ist, dass in vielen Fällen jede einzelne Filiale eines Unternehmens ihre eigene Datenbank besitzt. Was ist nun, wenn zur Analyse die Daten aller Filialen gemeinsam ausgewertet werden sollen?

In diesem Punkt kommt das Konzept der Data-Warehouse-Systems zum tragen. Das Data-Warehouse-System ist eine von den verschiedenen operativen Datenbanken der einzelnen Filialen getrennte Datenbank, deren einziger Zweck die effiziente Analyse von Daten verschiedener, unabhängiger Quellen (meistens auch Datenbanken) ist. Nach [ChDa97] wird ein Data-Warehouse-System definiert als eine Sammlung von Technologien zur Unterstützung der Entscheidung des Anwenders (Manager, Analyst), um diese möglichst schnell und gut treffen zu können.

Das Data-Warehouse Konzept ist hauptsächlich auf das „*Online Analytical Processing*“ (OLAP) ausgerichtet. Der Hauptzweck von OLAP-Anwendungen ist die effektive Analyse von Daten und die Unterstützung der Entscheidung des Managements. Im Mittelpunkt stehen weniger aktuelle Detaildaten, wie es bei den herkömmlichen operativen Datenbanken der Fall ist, sondern vielmehr die Analyse historischer Daten. Auf diese Daten wird hauptsächlich lesend zugegriffen, demzufolge ist eine möglichst gute Anfrage- und Antwortperformance wichtig.

Eine der Möglichkeiten, Anfragen schneller beantworten zu können, ist diese als Sichten in dem Data-Warehouse zu materialisieren. Das Prinzip der Materialisierung von Sichten, welche Vorteile man durch den Einsatz hat und wie sie zur Anfrageauswertung genutzt werden, wird in Abschnitt 1 vorgestellt und erläutert.

Allerdings ist es schwer zu entscheiden, welche Anfragen materialisiert werden sollen und wann dieses überhaupt sinnvoll ist. Mit dieser Frage beschäftigt sich der zweite Abschnitt, indem verschiedene Möglichkeiten und Verfahren für eine sinnvolle Auswahl von Sichten zur Materialisierung vorgestellt werden.

Dadurch, dass im Laufe der Zeit einem Data-Warehouse-System immer mehr neue Daten hinzugefügt werden, stellt sich das Problem, wie die materialisierten Sichten am besten aktualisiert werden. Dieses Problem behandelt Abschnitt 3 und stellt einige Lösungsmöglichkeiten und Algorithmen vor.

Durch die Aktualisierung der Sichten ergibt sich wieder ein neues Problem. Dadurch, dass sehr viele materialisierte Sichten innerhalb eines Data-Warehouse existieren und diese teilweise auf den gleichen Daten basieren, stellt sich die Frage der Konsistenz innerhalb des Data-Warehouses. Wobei nicht nur die Konsistenz zwischen Datenquellen und materialisierten Sichten gewahrt werden muss, sondern auch die Konsistenz zwischen den verschiedenen materialisierten Sichten. Verschiedene Ansätze und Konzepte zu diesem Thema werden im Abschnitt 4 behandelt.

1 Grundlagen

1.1 Prinzip der materialisierten Sichten

Die Hauptaufgabe von Data-Warehouse-Systemen ist die Verwaltung und Bereitstellung sehr großer Mengen an Daten. Ziel ist es, diese Daten möglichst effektiv analysieren zu können. Während des Analyse Vorganges treten hauptsächlich lesende Zugriffe und kaum schreibende Zugriffe auf den Datenbestand auf. Diese Zugriffe sollten möglichst effizient möglich sein, um die Analyse nicht unnötig zu verlängern. Das Problem ist, dass die im Data-Warehouses zu analysierenden Daten einen enormen Umfang (momentan um 200 TB, Tendenz steigend) annehmen können. Dementsprechend lange dauert es, bis bei einer Anfrage alle Daten ausgewertet sind.

Hinzu kommt die Tatsache, dass es sich bei den zur Analyse notwendigen Anfragen meistens um Aggregatfunktionen (COUNT, SUM, MAX, MIN) handelt, welche großteils auf der gleichen Menge an Relationen basieren. Diese Aggregatfunktionen haben zudem die Eigenschaft, dass sie verdichtend wirkend, das heißt zum Beispiel durch die SUM-Funktion werden viele Daten zu einem Wert zusammengefasst. Das Ergebnis einer Aggregationsanfrage hat dadurch meistens ein geringeres Datenvolumen als die Tabelle auf der sie angewendet wurde.

Diese Sachverhalte legen nahe, häufig verwendete Anfragen in Form von Sichten direkt im Data-Warehouse-System zu materialisieren. Dieses hat den Vorteil, dass nun Anfragen auf diese materialisierten Sichten zugreifen können, was eine deutliche Verringerung der Anfrageausführungszeit zur Folge hat.

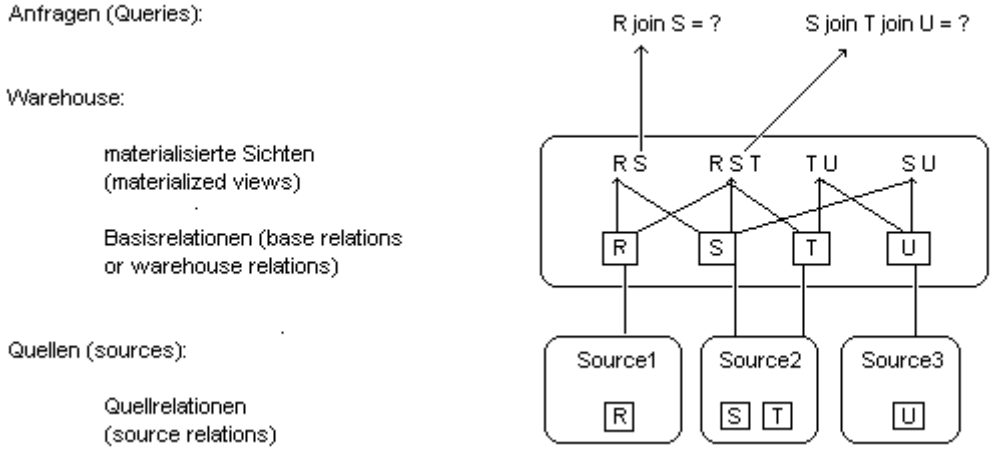

Abbildung 1: Prinzip der materialisierten Sichten

Das Prinzip wird in Abbildung 1 [LQA97] verdeutlicht. Aus den verschiedenen Quellen werden ausgewählte Relationen (R, S, T, U) extrahiert und in dem Data-Warehouse-System als „Basisrelationen“ gespeichert. Eine Sicht im Data-Warehouse ist ein Join-Produkt über diese Basisrelationen. Meistens wird dieses Join-Produkt mit verschiedenen Aggregatfunktionen und/oder Restriktionsbedingungen kombiniert. So erstellte werden daraufhin direkt im Data-Warehouse-System materialisiert ({R, S}; {R, S, T}; {T, U}; {S, U}), um einen effizienteren Zugriff für Anfragen auf die Daten der Basisrelationen zu ermöglichen.

Nach welchen Prinzipien diese Sichten erstellt werden, wird später in dem Abschnitt 2 behandelt.

1.2 Probleme

Eines der Hauptprobleme ist die Auswahl der zu materialisierenden Sichten. Auf der einen Seite sinkt durch eine hohe Anzahl an materialisierten Sichten die Anfrageausführungszeit, da weniger Daten aus den verteilen Quellen gelesen werden müssen. Andererseits steigt mit zunehmender Anzahl der benötigte Speicherplatz zur Ablage der Sichten. Es gilt eine optimale Balance zu finden, zwischen Reduktion der Anfragezeit und dem benötigten Speicher.

Ein weiteres Problem sind Änderungen in den Quelldatenbanken, da die betroffen Sichten mit dem betroffenen Ausgangsdatenbestand möglichst effizient zu synchronisieren sind.

1.3 Anfrageumformulierung („Query Rewrite“)

Voraussetzung für die Nutzung von materialisierten Sichten ist, dass das System eine transparente Nutzung der Sichten gewährleisten muss. Die Formulierung von Anfragen muss weiterhin unabhängig von den Sichten durchgeführt werden können. Für den Anwender darf es nicht ersichtlich sein, ob materialisierte Sichten zur Anfrageauswertung genutzt werden oder nicht. Ansonsten wäre es nötig, die Anfragen auf die bestehenden Sichten anzupassen. Ebenfalls müssten ansonsten Datenbankanwendungsprogramme für die Verwendung der materialisierten Sichten geändert werden, um mit diesen arbeiten zu können.

Das hat zur Folge, dass das Data-Warehouse-System die eingehende Anfrage transparent so umformulieren muss, dass die bestehenden materialisierten Sichten genutzt werden können. Dieses gilt auch für Sichten, die nicht genau einem Ausschnitt der Anfragen entsprechen. Durch so genannte Kompensationsoperationen werden die Anfragen dahingehend modifiziert, dass sie bestehende Sichten nutzen können.

Abbildung 2 ist ein Beispiel für eine sinnvolle Verwendung einer materialisierten Sicht. Durch die Verknüpfung der Anfrage Q mit der Sicht M ist der Zugriff auf die sehr große Tabelle Verkauf erspart worden. Die Selektionsoperation zur Beschränkung der Fakten σ_F wird auf direkt auf die Sicht M angewendet, während die Fakten σ_P bereits in der Sicht M beschränkt wurden. Zudem muss die neue Anfrage Q’ nur noch nach Region gruppieren.

Diese Ersetzung ist allerdings nur möglich, wenn sichergestellt werden kann, dass die restrukturierte Anfrage Q’ äquivalent zur ursprünglichen Anfrage Q ist. Hierzu gibt es die Definition einer gültigen Ersetzung.

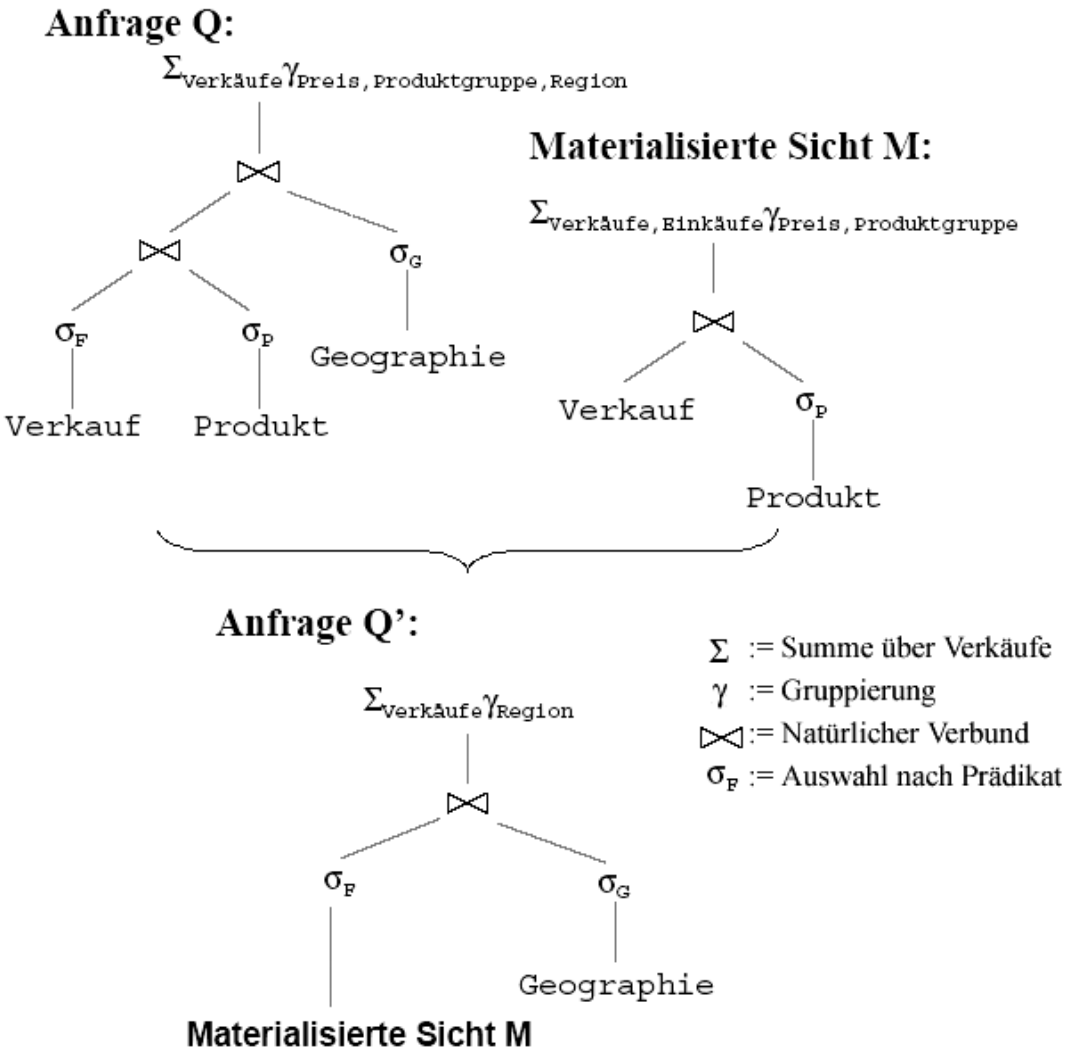

Abbildung 2 : Einsatz der Sicht M in der Anfrage Q durch Query Rewrite [BaGü00]

Definition: *Gültige Ersetzung* [BaGü00]

Eine Anfrage Q' ist eine gültige Ersetzung der Anfrage Q unter Verwendung der Materialisierung M, wenn Q und Q' das gleiche Multimengenergebnis liefern, das heißt jedes Element in der Menge kann mehrfach auftreten.

Die Berechnung einer gültigen Ersetzung für eine beliebige Anfrage ist NP-vollständig. Allerdings kann man Einschränkungen machen, da in der Praxis bei Data-Warehouse-System hauptsächlich Aggregationsanfragen in Form folgender Form:

```
SELECT     <Gruppierungsattribute>, <AGG (Kenngröße)>
FROM       <Faktentabelle>, <Dimensionstabelle>
WHERE      <Verbundbedingung> AND <Restriktionsbedingung>
GROUP BY   <Gruppierungsattribute>
```

Abbildung 3: Standard Aggregationsanfrage

Dieser Typ von Anfrage wird auch als „Star Query“ bezeichnet. Für die Gruppierungsattribute lässt sich die passende materialisierte Sicht mit dem so genannten

Aggregationsgitter finden, was in Abschnitt 2.2 erläutert wird. Eine Möglichkeit der Ableitung der Restriktionsbedingungen wird im nächsten Abschnitt behandelt.

1.4 Verallgemeinerte Projektion

Bei der Verallgemeinerten Projektion [GuHQ95] handelt es sich um eine Restrukturierungstechnik, die eine gültige Ersetzung für eine Star Query findet. Die Idee ist, dass eine verallgemeinerte Projektion

```
SELECT Preis, Produktgruppe, Region
FROM R
GROUP BY Preis, Produktgruppe, Region
```

äquivalent durch eine Projektion der relationalen Algebra ersetzt werden kann:

```
SELECT DISTINCT Preis, Produktgruppe, Region
FROM R
```

Diese muss nur noch für die Behandlung von Aggregationsfunktionen erweitert werden mit $\pi_{A1,...,An,agg(S)}$ (R). Somit hat man die Aggregatanfragen in die relationale Algebra umgewandelt.

Bei dem Verfahren werden alle eingehenden Anfragen, sowie alle bestehenden materialisierten Sichten als Anfragebäume betrachtet, wie in Abbildung 2 dargestellt ist. Diese Anfragebäume müssen in eine Normalenform $\sigma_h \pi \sigma_l \chi$ transformiert werden, die sich folgendermaßen aufbaut:

- Wurzel des Ausführungsgraphen: Selektion σ_h
- gefolgt von einer erweiterten Projektion π
- weiteren Selektionen σ_l (Selektionsprädikate h und l in konjunktiver Normalform)
- anschließend folgen alle Verbundoperationen (χ)

Der Algorithmus arbeitet in dem der Baum der eingehenden Anfrage Q derart umgeformt wird, dass der untere Teil des Baumes gleich der gegebenen materialisierten Sicht ist. Der restliche Teil des Baumes oberhalb wird zu der neuen Anfrage Q'. Die erlaubten Regeln zur Umformung werden in [GuHQ95] näher erläutert. Am Ende ist die materialisierte Sicht in die eingehende Anfrage eingebaut, so dass die Vorteile der materialisierten Sichten genutzt werden können.

2 Auswahl von Sichten

Voraussetzung für die Wiederverwendung von materialisierten Sichten ist auf der einen Seite die Additivität der verwendeten Aggregatfunktionen (SUM, MAX, COUNT,...), das heißt die Möglichkeit der Kombination mehrerer Aggregationsanfragen. Diese Eigenschaft ist nötig, um für Anfragen, die nicht genau einer materialisierten Sicht entsprechen, trotzdem diese nutzen zu können. Die meisten materialisierten Sichten entsprechen Aggregationsanfragen auf den Basisrelationen. Wenn diese Sichten zur Anfrageauswertung von anderen Aggregationsanfragen des Anwenders genutzt werden, kommt es zu einer Verschachtelung von Aggregationen, was das Ergebnis der Anfrage nicht beeinflussen darf. Die hierzu nötige Definition der Additivität wird im nächsten Abschnitt erläutert.

Auf der anderen Seite muss ein Auswertungskontext für Anfragen mit Aggregationen vorhanden sein, das heißt die Ableitungsbeziehungen zwischen verschiedenen Gruppierungsattributen müssen darstellbar sein. Eine mögliche Form der Darstellung ist das so genannte Aggregationsgitter, welches im übernächsten Abschnitt erläutert wird.

2.1 Additivität von Aggregationsfunktionen

Eine Aggregationsfunktion wirkt verdichtend auf einen Datenbestand, das heißt aus n Einzelwerten wird ein Aggregat berechnet. Daraus lässt sich die Definition der Additivität [Lehn98] herleiten.

Additivität: Gegeben eine Grundmenge $X = \{x_{i,j} \mid 1 \le i \le p, 1 \le j \le q\}$. Eine Aggregationsfunktion H heißt semi-additiv in Bezug auf X, wenn eine Aggregationsfunktion G existiert mit

$H(\{x_{i,j} \mid 1 \le i \le p, 1 \le j \le q\}) = G(\{H(\{x_{i,j} \mid 1 \le i \le p\}) \mid 1 \le j \le q\}) =$

$G(\{H(\{x_{i,j} \mid 1 \le j \le q\}) \mid 1 \le i \le p\})$

Ist die Aggregationsfunktion G identisch zu der Aggregationsfunktion H, so ist H additiv. Von indirekt-additiv spricht man, wenn sich eine Aggregationsfunktion H, zu einer Grundmenge X, durch einen endlichen und konstanten algebraischen Ausdruck über (semi-)additive Aggregationsfunktionen rekonstruieren lässt.

Beispiele:

- additiv: Beispiel die SUM-Funktion, da sie beliebig kombiniert werden kann und das Ergebnis sich nicht ändert
- semi-additiv: Beispiel die COUNT-Funktion (H ≡ COUNT und G ≡ SUM)
- indirekt-additiv: Beispiel die AVG-Funktion, da sie durch eine Division von einer Summe (additiv) und einer COUNT-Funktion (semi-additiv) rekonstruiert werden kann

2.2 Aggregationsgitter

Bei einem Aggregationsgitter handelt es sich um einen azyklischen Abhängigkeitsgraph, der anzeigt, für welche Kombinationen von Gruppierungsattributen sich Aggregatsfunktionen direkt oder indirekt aus anderen ableiten lassen (im Graphen selber durch einen Pfeil gekennzeichnet, siehe Abbildung 3). Im Beispiel ist die Gruppierung nach (A_2)(Region) aus den Gruppierungen nach (A_1, A_2) (Monat, Region) und (A_2, A_3) (Region, Gruppe) ableitbar. Jeder Knoten im Graphen entspricht der Möglichkeit, eine materialisierte Sicht zu bilden. Das oberste Element im Graphen wird als Superaggregat bezeichnet und entspricht einer Aggregation über alle eingehenden Einzelwerte, das heißt keiner Gruppierung.

In Beispiel existieren drei Gruppierungsattribute Monat, Region, Gruppe in der untersten Ebene wird nach allen drei Attributen also (Monat, Region, Gruppe) gruppiert. Aus dieser lassen sich nun direkt drei Zweierkombinationen ableiten. Eine Gruppierung nach einem Attribut lässt sich aus diesen Zweierkombinationen ableiten und letztendlich das Superaggregat aus den Einzelgruppierungen.

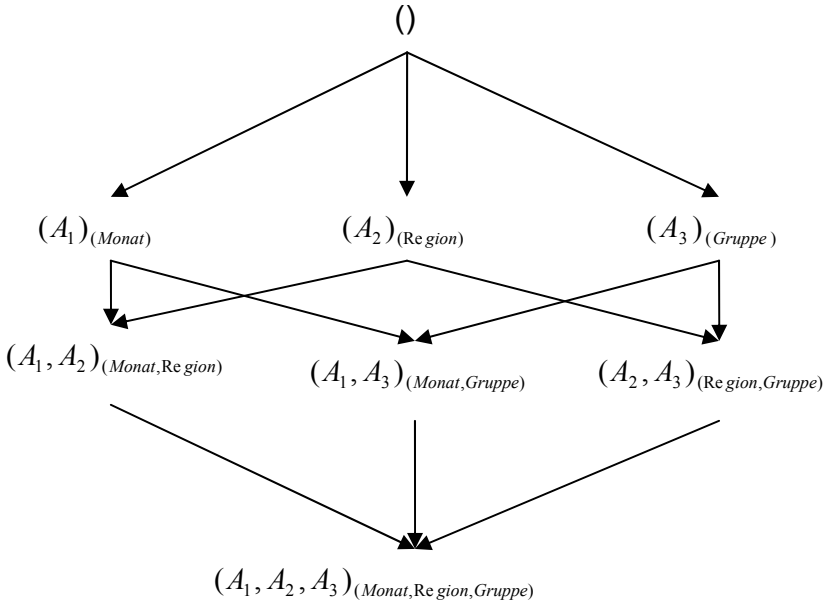

Abbildung 4: Beispiel eines Aggregationsgitters [BaGü00]

Daraus ergibt sich für n Gruppierungsattribute eine Knotenmenge von:

$$| N |= \prod_{j=1}^{n} 2 = 2^n$$

Das heißt bei 16 möglichen Gruppierungsattributen gibt es 65534 mögliche Kombinationen der Gruppierung. Nun stellt sich die Frage, welche Teilmenge dieser Knotenmenge für eine Materialisierung ideal ist, da eine Materialisierung aller Kombinationen zu aufwendig wäre.

2.3 Statische Auswahl materialisierter Sichten

Eine Möglichkeit der Auswahl einer möglichst optimalen Menge an Materialisierungen ist die statische Auswahl, das heißt die Auswahl der zu materialisierenden Sichten findet zu einem bestimmten Zeitpunkt statt, der von einem Algorithmus oder dem Administrator bestimmt wird. Der Nachteil ist, dass das aktuelle Anfrageverhalten nicht in die Berechnung einbezogen wird, sondern höchstens nur die Informationen, welche Anfragen in der Vergangenheit auftraten. Das aktuelle Anfrageverhalten wird bei dem dynamischen Auswahlverfahren in die Berechnung mit einbezogen, die im Abschnitt 2.4 erläutert werden.

2.3.1 Nutzen einer Materialisierungskonfiguration

Um später einen Algorithmus nutzen zu können, ist es nötig, ein Maß für den Nutzen einer materialisierten Sicht zu schaffen, um mehrere Gitterknoten vergleichen zu können.

Definition: Eine Funktion $C_q(n)$ heißt Kostenfunktion, wenn sie die Kosten zur Berechnung der Anfrage q aus dem Gitterpunkt n liefert. Falls q nicht aus dem Gitterpunkt berechenbar liefert $C_q(n) = \infty$

***Definition*:** Sei M die Menge der bereits materialisierten Knoten eines Aggregationsgitters und Q die Menge von Anfragen. Die Nutzabschätzung eines zusätzlichen Gitterknotens n ist dann bestimmt durch:

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

2.3.2 Beispielalgorithmus

Um das Prinzip des statischen Auswahlverfahrens zu veranschaulichen wird der Algorithmus (Abbildung 5) nach [HaRU96] erläutert. Der Algorithmus arbeitet nach dem Greedy-Prinzip und liefert zu einer gegebenen Menge von Gitterknoten N und einem maximalen Speicheraufwand S eine Menge zu materialisierender Knoten M mit dem größten Nutzen.

Zu Beginn besteht die Menge der zu materialisierenden Knoten nur aus den Detaildaten, dass heißt dem untersten Knoten des Aggregatgitters. Nun wird iterativ solange in jedem Schritt der Knoten mit dem größten Nutzen hinzugefügt, bis der benötigte Speicherbedarf den maximalen Speichermehraufwand erreicht hat.

```
Eingabe:    Menge aller Gitterknoten N; erwarteter Speicheraufwand |n|
            bei Materialisierung von Knoten n_i ∈ 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)
            // berechne Gitterpunkt mit max. Nutzen bzgl. M
            n ist der Gitterpunkt mit  ∀n ∈ (N / M) = max_{n_i ∈ M} (B_{n_i}(M))
            M = M ∪ {n} //Füge n der Materialisierungsmenge hinzu
            s = s + |n| //zusätzlich benötigter Speicher
      End While
      Return M
End
```

Abbildung 5: Beispielalgorithmus zum statischen Auswahlverfahren

2.3.3 Analyse des Verfahrens

Der Auswahlalgorithmus ist relativ rechenaufwändig mit einer Komplexität von $O(n^3)$ mit n als Anzahl der Knoten des Aggregationsgitters. Allerdings lassen sich noch einige Optimierungen durchführen.

Wenn zwischen verschiedenen Gruppierungsattributen funktionale Abhängigkeiten existieren, sind Gruppierungskombinationen die beide Attribute enthalten äquivalent zu denen die nur eines der beiden Attribute enthalten, da beide identische Ausprägungen besitzen. Zum Beispiel wenn gilt (Artikel → Produktgruppe) dann sind die beiden Gruppierungen (Artikel) und (Artikel, Produktgruppe) identisch. Das ist der Fall, wenn die Produktgruppe durch den Artikel bestimmt wird. Der umgekehrte Fall, dass nur die Gruppierung (Produktgruppe) genommen wird, ist allerdings nicht möglich, da aus der Produktgruppe keinerlei Rückschlüsse auf den Artikel möglich sind. Dadurch lassen sich

einige Gitterknoten zusammenfassen und müssen nicht mehr in dem Algorithmus betrachtet werden.

Eine andere Möglichkeit der Optimierung ist die Betrachtung der auftretenden Anfragen und einer Berücksichtigung nur der Knoten, deren Attribute in den Anfragen vorkommen.

Zuletzt ist eine Betrachtung des Verdichtungsfaktors der einzelnen Knoten möglich, dass heißt, wenn nicht genügend Daten durch einen Knoten zusammengefasst werden, wird dieser nicht betrachtet.

Ein derart reduziertes Aggregationsgitter ist dann der Ausgangspunkt zur Berechnung mit dem obigen Auswahlverfahren.

2.3.4 Auswahl von Partitionen materialisierter Sichten

Das Problem bei diesem Verfahren ist, dass noch keinerlei Restriktionsbedingungen in den Anfragen berücksichtigt werden. Es kommt aber häufig vor, dass nur bestimmte Teile eines kompletten Datensatzes häufig abgefragt werden. Zum Beispiel bei der Berechnung der Verkaufssumme sind meistens nur die Artikel aus dem letzten Monat oder aus bestimmten Regionen von Interesse. In der materialisierten Sicht sind aber alle Daten des letzten Jahres (abhängig von der Vorhaltezeit) und alle Regionen vorhanden. Dadurch wird eine große Menge an Daten in dem Data-Warehouse-System gehalten, welche nicht unbedingt nötig sind.

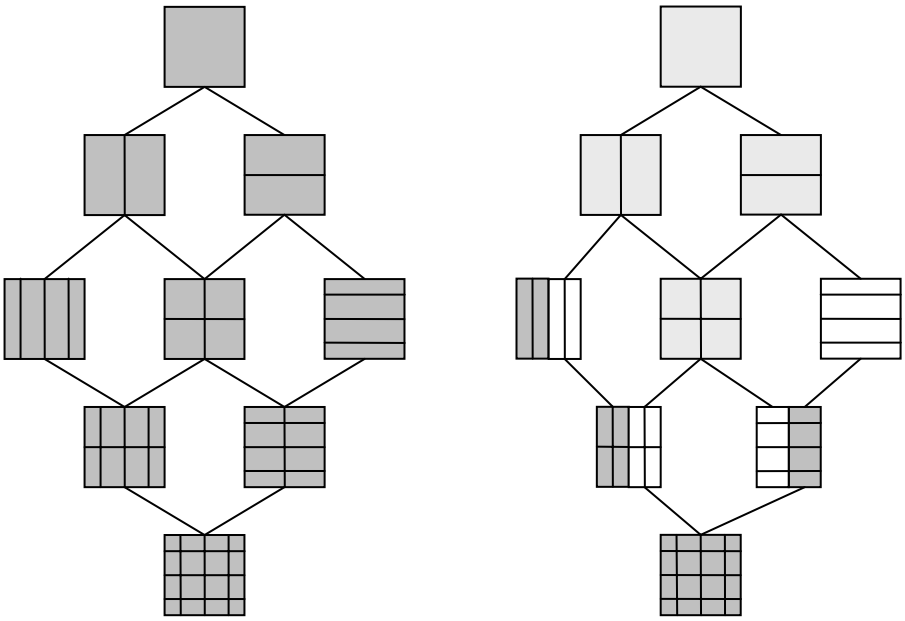

Abbildung 6: Vollständige und partielle Materialisierung

Die Idee ist, nur den Teil der Daten in Sichten zu materialisieren, auf den häufig zugegriffen wird. Das Prinzip wird in Abbildung 6 verdeutlicht. Die dunkelgrauen Bereiche verdeutlichen materialisierte Daten auf bestimmten Aggregationsstufen. Die Detaildaten sind vollständig materialisiert, allerdings sind auf den höheren Aggregationsebenen nur Teilbereiche, wie zum Beispiel die Verkaufsdaten für Süddeutschland pro Region und für Norddeutschland pro Produktgruppe. Wenn sich jetzt die meisten Anfragen entweder auf Süddeutschland oder Norddeutschland beziehen, können die Daten effektiver gelesen werden.

Eine Erweiterungsmöglichkeit ist, falls eine Anfrage alle Daten der nicht materialisierten, hellgrauen Bereiche abfragt, diese aus den bestehenden materialisierten Sichten zu berechnen, obwohl keine der Sichten alleine dazu ausreicht. Hierzu sind aber Algorithmen nötig, die erkennen, ob es sich um eine gültige Ersetzung handelt.

2.3.5 Probleme bei der statischen Auswahl

Das statische Auswahlverfahren basiert auf der Annahme, dass die ausgewählten Sichten über einen gewissen Zeitraum (z.B. nachts) materialisiert und aktualisiert werden. Durch statische Materialisierung entstehen große Performancesteigerungen allerdings entstehen durch alleinige Anwendung auch Nachteile:

- Oft werden bestimmte Zusammenhänge gezielt untersucht, so dass zu einem bestimmten Zeitpunkt ähnliche Anfragen gestellt werden. Die Folge ist, dass das typische Anfragemuster aus mehreren nicht vorhersehbaren Anfragen besteht.
- Wenn die Daten häufig verändert werden, veralten die materialisierten Sichten schnell, was einen erhöhten Aktualisierungsaufwand zur Folge hat
- Das Anfragemuster ändert sich beständig, so dass ständig eine Anpassung der materialisierten Sichten stattfinden müsste.
- Aufgrund der zunehmenden Globalisierung, kann es vorkommen, dass aus unterschiedlichen Zeitzonen auf ein Data-Warehouse zugegriffen wird. Dadurch ergibt sich kein Zeitraum mehr, indem die Sichten materialisiert werden können, ohne das Anfrageverhalten zu beeiträchtigen.

Aufgrund der obigen Punkte sollte auch das aktuelle Anfrageverhalten bei der Materialisierung betrachtet werden. Verfahren hierzu werden im nächsten Abschnitt behandelt.

2.4 Dynamische Auswahl materialisierter Sichten

Das Problem der statischen Auswahl von materialisierten Sichten, dass nur das Anfrageverhalten aus der Vergangenheit genutzt wird zur Entscheidung welche Sichten zu materialisieren sind, macht es nötig Verfahren zur Anpassung der Sichten an das aktuelle Anfrageverhalten zu ergänzen. Insbesondere bei OLAP-Anwendungen ist die Wahrscheinlichkeit für die Wiederverwendbarkeit eines Anfrageergebnisses relativ hoch, da es oft vorkommt, dass innerhalb einer OLAP-Sitzung die Anfragen aufeinander aufbauen. Das heißt Ergebnisse aus einer vorangegangenen Anfrage werden als Grundlage bei der nächsten Anfrage genutzt. Deshalb ist es nötig, einen Teil der Anfrageergebnisse in einem reservierten Speicherbereich zur Wiederverwendung zu materialisieren. Für dieses Verfahren gibt es noch einen weiteren Vorteil. Oft sind die Ergebnismengen von Anfragen relativ klein, so dass eine Zwischenspeicherung nicht übermäßig viel Platz benötigt, aber eine deutliche Beschleunigung einer Folge von Anfragen ermöglicht.

2.4.1 Semantisches Caching

Das Prinzip eines Caches oder Pufferspeichers wird in vielen Bereichen verwendet. Jedes Datenbanksystem verfügt über einen Cache, der häufig benutzte Daten im Hauptspeicher zwischenlagert. Allerdings basiert die Verwaltung des Caches meistens darauf, dass der Inhalt des Caches nicht bekannt ist, sondern allgemeine Kriterien wie Einlagerungszeit

und Referenzierungshäufigkeit sind Bestandteil der Verdrängungsstrategie. Im Gegensatz dazu wird bei dem semantischen Caching die Entscheidung über Verdrängung aus dem Puffer anhand der Semantik und des Zusammenhangs der gepufferten Daten getroffen.

Dieses Verfahren ist insbesondere sinnvoll bei der Entscheidung, ob materialisierte Sichten verdrängt werden oder nicht, da hier Ziel ist, den Nutzen der Daten in Zukunft, anhand des Inhalts der materialisierten Sichten, abzuschätzen. Hinzu kommt die Möglichkeit, wenn wie bei OLAP das Anwendungsgebiet bekannt ist, eine noch effektivere Abschätzung getroffen werden kann.

Die Zwischenspeicherung findet nicht nur explizit im Hauptspeicher, sondern auch auf der Festplatte statt, da eine Anfragebeschleunigung nicht nur durch das schnellere Laden der Daten, sondern auch durch ein schnelleres Finden der Ergebnismenge erreicht wird.

Es darf, ebenso wie bei der statischen Auswahl materialisierter Sichten, die Gesamtgröße der durch semantisches Caching materialisierten Sichten ein bestimmtes Maximum nicht überschreiten. Das macht Verdrängungsverfahren notwendig, um gegebenenfalls auf eine Änderung des Anfrageverhaltens reagieren zu können und neue Sichten materialisieren zu können. Im Gegensatz zu normalen Verdrängungsverfahren kommt hier erschwerend hinzu, dass verschiedene Anfrageergebnisse nicht zwingend gleich groß sind. Es kann passieren, dass drei gepufferte Sichten verdrängt werden müssen damit eine neue Sicht materialisiert werden kann.

Neben der Daten- und Anwendungsbezogenen Analyse der Daten werden auch weiterhin klassische Verdrängungsfaktoren wie Referenzierungshäufigkeit und Verweildauer im Cache betrachtet. Folgende Faktoren zur Beurteilung des Nutzens einer materialisierten Sicht haben sich herauskristallisiert ([KoRo99], [ScSV99] und [ABD+99]):

- Zeit des letzten Zugriffs
- Referenzierungshäufigkeit
- Größe der materialisierten Sicht
- Kosten, die durch eine Neuberechnung oder Aktualisierung der materialisierten Sicht verursacht wurden
- Anzahl der Anfragen, die in der Vergangenheit mit dieser Sicht hätten beantwortet werden können oder beantwortet worden sind
- Anzahl der Anfragen, die prinzipiell mit dieser Sicht beantwortet werden können

Diese Faktoren gehen unterschiedlich gewichtet in die Berechnung des Nutzwertes einer materialisierten Sicht ein. Mit Hilfe des Nutzwertes werden die bereits materialisierten Sichten sortiert und je nach Bedarf werden die Sichten mit dem geringsten Nutzen verdrängt.

3 Aktualisierung materialisierter Sichten

Nach der Erzeugung mehrerer materialisierter Sichten stellt sich die Frage, wie diese möglichst effizient aktualisiert werden bei einer Änderung der Detaildaten. Hinzu kommt das Problem, dass aufgrund von Redundanzen zwischen verschiedenen Sichten, meistens mehrere Sichten gleichzeitig aktualisiert werden müssen.

3.1 Rematerialisierung

Eine Möglichkeit der Aktualisierung von Sichten ist die Rematerialisierung, das heißt eine komplette Löschung und Neuberechnung der zu aktualisierenden materialisierten Sicht. Allerdings ist dieses sehr ineffizient, wenn sich nur ein kleiner Teil der Daten geändert hat. Sinnvoller ist es, wenn die Änderungen einer Sicht berechnet werden und dann inkrementell eingefügt werden. Dieses Verfahren wird im nächsten Abschnitt beschrieben.

3.2 Inkrementelle Aktualisierung

Die Idee bei der inkrementellen Aktualisierung ist erst festzustellen, welche der Detaildaten sich geändert haben und diese Änderungen in den materialisierten Sichten nachzuvollziehen.

Bei der Aktualisierung einer Sicht V wird der neue Zustand V_{neu} mit Hilfe des alten Zustandes und der durchgeführten Änderungen mit folgender Formel berechnet:

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

Basisrelation R; Sicht V = f(R) Δ^+R: neu eingefügte Tupel; Δ^-R: gelöschte Tupel

Mit der Formel können sowohl Lösch- als Einfügeoperationen durchgeführt werden. Änderungsoperationen werden einfach als Löschen mit anschließendem Einfügen betrachten. Das Prinzip der inkrementellen Aktualisierung wird an nachfolgendem Beispiel verdeutlicht:

Gegeben:

- Zwei Basisrelationen (Quellen) link1(W, X) und link2(X, Y), wobei link1 = {(1, 2)} und link2 = {(2, 4)}
- Sicht HOP auf diesen Basisrelationen mit:

  HOP = π_W (link1(W, X) $\bowtie$ link2(X, Y)) → HOP = {(1)}

Ablauf der Aktualisierung:

1. Update U_1 = insert(link2, {(2,3)}
   → Quellen schicken U_1 als Benachrichtigung an das DW (Data-Warehouse)
2. DW empfängt U_1;
   → Algorithmus zur inkrementellen Aktualisierung sendet Anfrage an die Quelle
   i. $Q_1 = \pi_W$ (link1(W, X) $\bowtie$ {(2,3)})
3. Quelle empfängt Q_1
   → berechnet das Ergebnis A_1 = {(1)}
4. DW empfängt A_1 und aktualisiert HOP $\cup$ A_1 = {(1),(1)}

3.2.1 Klassifizierung von Aktualisierungsalgorithmen

Die Effizienz der Algorithmen hängt von verschiedenen Einflussfaktoren (nach [BaGü00]) ab:

- Die Menge der zur Verfügung stehenden Informationen hat Einfluss auf die Auswahl eines Algorithmus. Allerdings muss die Definition einer Sicht und ihr Ausprägung immer bekannt sein. Auch wichtig ist die Frage, ob ein Zugriff auf die Basisrelationen überhaupt möglich ist, beziehungsweise wenn nicht, welche Integritätsbedingungen existieren.
- Je komplexer die möglichen Anfragekonstrukte zur Definition einer Sicht sind, desto schwieriger wird es auch später diese Sichten zu aktualisieren.
- Wichtig ist, welche Modifikationstypen von den Algorithmen verarbeitet werden können. Im einfachsten Fall sind es nur Einfüge- und Löschoperationen die verwendete werden. Änderungsoperationen werden demnach wie eine aufeinander folgende Einfüge- und Löschoperationen behandelt.
- Ein entscheidender Parameter ist die Granularität der Aktualisierung, also ob Sichten einzeln aktualisiert werden können, was eine sehr flexible Aktualisierung ermöglicht oder im Extremfall, dass immer das ganze Data-Warehouse auf einmal aktualisiert werden muss.
- Der Zeitpunkt der Aktualisierung:
  - Sofortige Aktualisierung: Bei einer Änderung der Basisdaten werden gleichzeitig die abgeleiteten Daten im Data-Warehouse-System aktualisiert. Der Vorteil ist, dass das Data-Warehouse immer aktuell ist, allerdings entstehen hohe Kosten durch die häufigen Modifikationen.
  - Verzögerte Aktualisierung: Eine materialisierte Sicht wird erst dann aktualisiert, wenn auf sie zugegriffen wird. Dadurch werden unnötige Aktualisierungen vermieden, allerdings kann Wartezeit entstehen, wenn auf mehrere veraltete Sichten zugegriffen wird.
  - Snapshot Aktualisierung: Die Aktualisierung erfolgt asynchron nach Änderungen der Basisdaten. Der Zeitpunkt kann nach anwendungsspezifischen Kriterien ausgewählt werden, allerdings wird der Zugriff auf veraltete Daten in gewissen Maßen toleriert.

3.2.2 Autonome Aktualisierbarkeit

Die meisten Aktualisierungsalgorithmen verwenden zur Berechnung der Änderung einer materialisierten Sicht sowohl die Definition der Sicht, als auch die an der Basisrelation aufgetretenen Änderungen. Problematisch wird es, wenn auch auf die Ausprägung der Basisrelation selbst zugegriffen wird. Diese Zugriffe aus dem Data-Warehouse-System auf die Basisrelationen (siehe Abbildung 7) sind sehr teuer aufgrund der großen Datenmengen in den Quellen. Ziel ist es, neue Verfahren zu implementieren, die diese Zugriffe verringern bzw. ganz vermeiden.

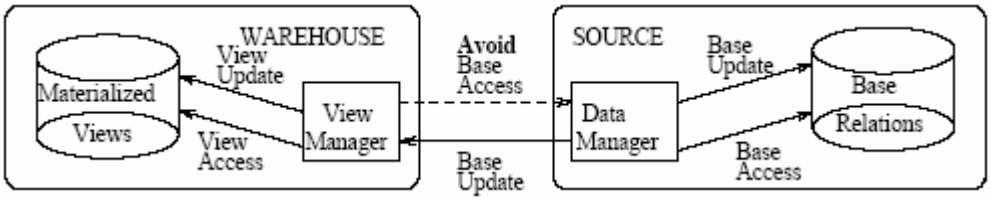

Abbildung 7: Schnellere Aktualisierung der Sichten durch Vermeidung unnötiger Zugriffe auf Basisdaten [Huyn97]

Wenn eine materialisierte Sicht nur mit Hilfe der Definition der Sicht, der Änderung der Basisrelation sowie der bestehenden Instanz der Sicht eine inkrementelle Aktualisierung durchführen kann bei Einfügen, Löschen oder Änderung der Basisrelation, so spricht man von *autonom aktualisierbar*.

In der Praxis ist dieses allerdings kaum zu erreichen, deswegen wird meistens die partielle Autonomie realisiert, in der nicht gefordert wird, dass alle Änderungsoperationen autonom aktualisiert werden müssen. Prinzipiell gilt, je mehr Informationen über die Basisrelationen vorhanden sind, desto weniger Zugriffe sind auf diese nötig. Mögliche Zusatzinformationen sind:

- Schemainformationen: Die meisten Algorithmen verwenden Schemainformationen wie Primärschlüsseleigenschaften, aber auch aus Fremdschlüsselbeziehungen lassen sich Informationen ableiten.
- Count: Die Anzahl der Tupel aus der Basisrelation, aus denen ein Tupel in der materialisierten Sicht abgeleitet wird (siehe Abschnitt 3.2.4).
- Verwendung von Hilfssichten: Die Materialisierung weiterer Informationen, die eine (partielle) autonome Aktualisierbarkeit ermöglichen.

3.2.3 Counting-Algorithmus

Die Idee beim Counting-Algorithmus ist, die Anzahl des Vorkommens eines Tupels in der Basisrelation als Zusatzinformation in der materialisierten Sicht zu speichern. Er kann auf Sichten angewendet werden, die durch SQL Anfragen erstellt wurden, unabhängig ob Duplikate innerhalb der Sicht erlaubt sind oder nicht. Diese Anfragen können sowohl Aggregate, UNION und Negationen verwenden.

Der Algorithmus berechnet, mit der in Abschnitt 3.2 geschilderten Methode der inkrementellen Aktualisierung, aus einer gegeben Formel T, die eine Menge von Sichten $V_1,\ldots.V_n$ definiert, eine neue Formel ΔT. Diese neue Formel ΔT berechnet aus der alten Basisrelation, den durchgeführten Änderungen auf dieser Relation und der Sichtdefinition die Änderungen $\Delta(V_1),\ldots,\Delta(V_n)$, welche auf den einzelnen Sichten durchzuführen sind. Wobei die einzelnen $\Delta(V_k)$ die Änderungen der Anzahl der einzelnen Tupel innerhalb der Sicht K sind. Bei einer Einfügeoperation wird die Anzahl des betroffenen Tupels erhöht und umgekehrt bei einem Löschen die Anzahl verringert. Ein Tupel mit der Anzahl eins wird hinzugefügt und alle Tupel mit einer Anzahl ≤ 0 werden gelöscht.

Beispiel:

Gegeben sei eine Sicht HOP auf der Basisrelation LINK, wobei HOP folgendermaßen in SQL geschrieben ist:

```
CREATE VIEW hop(S,D)AS

(SELECT DISTINCT l1.S, l2.D
     FROM link l1, link l2
     WHERE l1.D = l2.S)
```

Sei Link = {(A,B), (B,C), (B,E), (A,D),(D,C)} daraus ergibt sich HOP = {(A,C),(A,E)}, wobei count(A,C) = 2 und count(A,E) = 1.

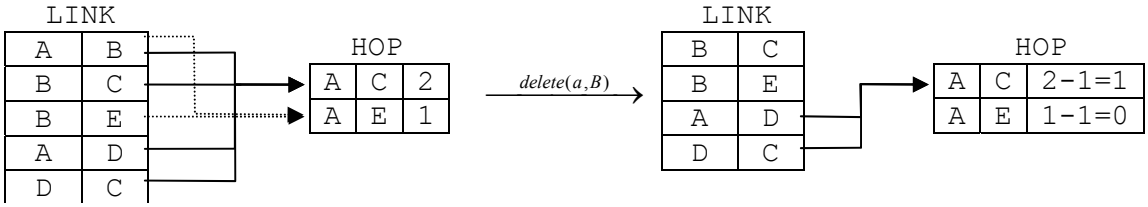

Abbildung 8: Beispiel des Counting-Algorithmus

Wenn (A,B) aus der Basisrelation gelöscht wird, berechnet der Counting Algorithmus, dass jeweils einmal (A,E) und (A,C) gelöscht werden muss (siehe auch Abbildung 8).

3.2.4 Eager-Compensation-Algorithm

Im folgenden Abschnitt wird der Eager-Compensation-Algorithmus nach [HGWZ95] erläutert. Der Vorteil des Algorithmus ist, dass er die Entstehung von Anomalien bei der inkrementellen Aktualisierung verhindert. An folgendem Beispiel wird verdeutlicht, wie Anomalien bei der Aktualisierung entstehen können. Die Situation ist, dass zwei parallele Updates der Basisrelation auftreten:

Gegeben:

- Zwei Basisrelationen (Quellen) LINK1(W, X) und LINK2(X, Y), wobei LINK1 = {(1, 2)} und LINK2 = {(2, 4)}
- Sicht HOP auf diesen Basisrelationen mit:

  HOP = π_W (link1(W, X) $\bowtie$ link2(X, Y)) → HOP = {(1)}

Ablauf der Aktualisierung:

1. Update U_1 = insert(link2, {(2,3)}
   → Quellen schicken U_1 als Benachrichtigung an das DW (Data-Warehouse)
2. DW empfängt U_1;
   → Algorithmus zur inkrementellen Aktualisierung sendet Anfrage

   $Q_1 = \pi_W$ (link1(W, X) $\bowtie$ {(2,3)})
3. Update U_2 = insert(link1, {(4,2)}
   → Quellen schicken U_2 als Benachrichtigung an das DW (Data-Warehouse)
4. DW empfängt U_2;
   → Algorithmus zur inkrementellen Aktualisierung sendet Anfrage

   $Q_2 = \pi_W$ ({(4,2)}$\bowtie$link2(W, X))
5. Quelle empfängt Q_1
   → berechnet auf link1 = {(1,2), (4,2)} das Ergebnis A_1 = {(1), (4)}
6. DW empfängt A_1 und aktualisiert HOP $\cup$ A_1 = {(1),(4)}
7. Quelle empfängt Q_2
   → berechnet auf link1 und link2 das Ergebnis A_2 = { (4)}

8. DW empfängt A_2 und aktualisiert HOP $\cup$ A_2 = {(1), (4), (4)}

   → Anomalie: korrekt wäre HOP = {(1), (4)}

Um diese Anomalien zu vermeiden gibt es mehrere Möglichkeiten. Die einfachste ist eine komplette Neuberechnung der materialisierten Sicht, was aber, wie in Abschnitt 3.1 behandelt, sehr aufwändig ist.

Eine weitere Möglichkeit ist eine Kopie aller Basisrelationen, die von einer materialisierten Sicht verwendet werden, zu erstellen und auf diesen die Anfrageauswertung durchzuführen. Der Nachteil ist auf der einen Seite der erhöhte Speicherbedarf und auf der anderen Seite hat sich das Problem nur verschoben, da nun die Basisrelationskopien aktualisiert werden müssen.

In diesem Punkt greift der Eager-Compensation-Algorithmus, indem bei Quellanfragen Kompensationsanweisungen hinzugefügt werden, um mehrere parallele Änderungen ausgleichen zu können.

Unter der Voraussetzung, dass Nachrichten in ihrer Reihenfolge ankommen, kann das Data-Warehouse, wenn es von der Quelle ein Update U_2 empfängt, aber noch keine Antwort A_1 auf die Anfrage Q_1 erhalten hat, daraus schließen, dass die Anfrage Q_1 auf einem inkorrekten Zustand ausgeführt wird. In der Quelle wurde bereits das Update U_2 ausgeführt, obwohl die Anfrage Q_1 auf dem Zustand der Quelle vor dem Update basiert. Nachdem das Data-Warehouse dieses erkannt hat, wird die Anfrage Q_2 um eine Kompensationsoperation erweitert, die das falsche Ergebnis von Anfrage Q_1 im Ergebnis der Anfrage Q_2 ausgleicht.

Der Algorithmus speichert alle unbeantworteten Anfragen Q_i in der Menge UQS (unanswered-query-set), um anhand dieser Menge feststellen zu können, zu welchen Anfragen bei einem Update U_i Kompensationsanfragen notwendig sind. Um Inkonsistenzen innerhalb der materialisierten Sicht zu vermeiden, werden alle Antworten A_i in einer temporären Relation COLLECT gespeichert. Die Änderungen werden erst in die Sicht übertragen, wenn alle ausstehenden Anfragen Q_i abgearbeitet wurden, dass heißt in der Menge UQS befinden sich keine Anfragen mehr.

Algorithmus:

Die Quelle verhält sich wie im obigen Beispiel. Basisrelationen und Sicht ebenfalls wie oben.

```
COLLECT = Ø;

DW:   empfange Ui;
      → sende  Qi = HOP(Ui) − Σ_{Qj∈UQS} Qj(Ui)  an Quelle
DW:   empfange Ai
      COLLECT = COLLECT ∪ Ai
      if UQS = Ø
            then  HOP = HOP ∪ COLLECT
                  COLLECT := Ø
            else  tue nichts
```

Es wird nur das Verhalten des Data-Warehouses während der Aktualisierung mit dem ECA-Algorithmus betrachtet. Das Verhalten der Quelle entspricht dem im vorherigen Beispiel. Die Situation ist, dass drei Updates auftreten, bevor es zu einer Antwort kommt:

Gegeben:

- Basisrelationen:

link1(W, X), link2(X, Y), link3(Y, Z), wobei link1 = {(1,2)}, link2 = Ø und link3 = Ø ist

- Materialisierte Sicht:

  HOP = π_W (link1 ⋈ link2 ⋈ link3) , wobei HOP=Ø und COLLECT=Ø

Ablauf der Aktualisierung:

1. DW empfängt U_1 = insert (link1, (4,2))
   DW sendet $Q_1 = HOP(U_1) = \pi_W$ ((4,2) ⋈ link2 ⋈ link3)
2. DW empfängt U_2 = insert (link3, (5,3))
   UQS = { Q_1}
   DW sendet $Q_2 = HOP(U_2)$
   = π_W (link1 ⋈ link2 ⋈ (5,3)) - π_W ((4,2) ⋈ link2 ⋈ (5,3))
3. DW empfängt U_3 = insert (link3, (2,5))
   UQS = { Q_1, Q_2}
   DW sendet $Q_3 = HOP(U_3) – Q_1(U_3) – Q_2(U_3)$
   = π_W (link1 ⋈ (2,5) ⋈ link3) - π_W ((4,2) ⋈ (2,5) ⋈ link3)
   – π_W ((link1 - (4,2)) ⋈ (2,5) ⋈ (5,3))
4. DW empfängt A_1 = (4)
   COLLECT = Ø ∪ COLLECT = {(4)}; UQS = { Q_2, Q_3}
5. DW empfängt A_2 = (1)
   COLLECT = COLLECT ∪ (1) = {(4), (1)}; UQS = {Q_3}
6. DW empfängt A_3 = Ø
   COLLECT = COLLECT ∪ Ø = {(4), (1)}; UQS = Ø
   DW aktualisiert HOP = Ø ∪ COLLECT = {(4), (1)}

   Ergebnis ist korrekt!

Der ECA-Algorithmus spart Zeit und Kosten im Vergleich zu einer vollständigen Neuberechnung der Sichten. Ein weiterer Vorteil ist, dass es nicht nötig ist die materialisierten Sichten zu sperren während der Aktualisierung. Allerdings sind bei diesem Algorithmus immer noch Zugriffe aus den materialisierten Sichten auf die Basisrelationen vorhanden.

Eine Modifikation ist der ECA-Key-Algorithmus, der fähig ist bei Löschoperationen auf diese Zugriffe zu verzichten, indem die Menge COLLECT am Anfang nicht leer ist, sondern mit der bestehenden Sicht initialisiert wird. Auf dieser Kopie der Sicht werden im weiteren die Löschoperationen angewendet, ohne eine Anfrage an die Basisrelation zu schicken. Allerdings funktioniert dieses nur unter der Einschränkung, dass die Sicht derart definiert ist, dass sie jeden Schlüssel der Basisrelation enthält.

4 Konsistenz

Dadurch das in großen Data-Warehouse-System tausende von materialisierten Sichten keine Seltenheit sind [WeCa96] und oft Redundanzen bestehen zwischen den verschiedenen Sichten, sind Konsistenzaspekte nicht nur zwischen materialisierten Sichten und Basisrelation wichtig, sondern auch zwischen den einzelnen materialisierten Sichten von großer Bedeutung.

4.1 Konsistenzaspekte

Es existieren verschiedene Aspekte der Konsistenz eines Data-Warehouse-Systems. Welche Aspekte letztendlich angewendet werden, hängt von den Anforderungen ab, die an das Data-Warehouse-System gestellt werden. Hinzu kommt, dass je strenger die Anforderungen an die Konsistenz der Daten sind, desto mehr steigt auch die Komplexität des Data-Warehouse-Systems. In den folgenden Unterabschnitten werden verschiedene Anforderungen besprochen.

4.1.1 Anwenderdefinierte Aktualitätsanforderungen

Der gewünschte Grad der Konsistenz ist zum großen Teil von der jeweiligen Anwendung abhängig. Mögliche Maße für die Abweichung des Zustandes der materialisierten Sichten zu dem Zustand der Basisrelationen sind:

- Zeitlicher Abstand: Die Zeitspanne, um die die materialisierte Sicht älter sein darf als die Basisrelation.
- Wertemäßiger Abstand: Die maximale und minimale Grenze, die Werte der materialisierten Sicht zu denen der Basisrelation abweichen dürfen.
- Versionsbezogener Abstand: Die Anzahl erlaubter Änderungen (Versionen) der Basisrelation, bevor die materialisierten Sicht wieder aktualisiert werden muss.

Diese Maße sind stark von dem jeweiligen Bereich abhängig, indem das Data-Warehouse genutzt wird. Wenn es darum geht die Daten der letzten zehn Jahre zu analysieren, fällt es nicht sonderlich ins Gewicht, wenn die materialisierten Sichten seid einem Tag nicht aktualisiert wurden. Im Gegensatz dazu fällt dieser eine Tag sehr wohl ins Gewicht, wenn nur die Daten der letzten Woche analysiert werden müssen.

4.1.2 Anfragekonsistenz und Sitzungskonsistenz

Die Minimale Anforderung aus Anwendersicht ist, dass alle an der Anfrage beteiligten materialisierten Sichten gleich aktuell sind, was aber nicht bedeutet, dass diese der neusten Version der Basisrelation entsprechen müssen. Diese Anforderung ist zwingend nötig, da ansonsten Anfragen neue Daten aus einer Sicht, mit alten Daten aus einer anderen Sicht vermischen können und dadurch inkorrekte Anfrageergebnisse entstehen können.

Noch stärker als die Anfragekonsistenz ist die Sitzungskonsistenz, da nicht nur einzelne Anfragen, sondern die ganze Sitzung (mehrere Anfragen hintereinander) untereinander konsistent sein müssen. Dieses ist wichtig, wenn für Analysen komplexe Anfragen verwendet werden, die Ergebnisse mehrere Anfragen miteinander verrechnen. Ansonsten können inkorrekte Ergebnisse entstehen, durch Verwendung von Ergebnissen von Anfragen auf älteren Daten und Ergebnisse von Anfragen auf aktuelleren Daten.

4.1.3 Aktualisierungsgranulat

Die Flexibilität der Aktualisierung hängt entscheidend mit der Konsistenz des Data-Warehouse-Systems zusammen, umso flexibler die Aktualisierung ist, desto schwieriger wird es die Konsistenz zu wahren. Die Flexibilität der Aktualisierung hängt von den Teilen des Data-Warehouse-System ab, die zeitgleich aktualisiert werden müssen:

- gesamtes Data-Warehouse: Die einfachste Möglichkeit der Konsistenz-Wahrung ist, dass das komplette Data-Warehouse-System während der Aktualisierung für Anwenderanfragen gesperrt wird und komplett aktualisiert wird. Der Vorteil ist, dass alle Inkonsistenzen nahezu ausgeschlossen werden können, allerdings ist keine nebenläufige Aktualisierung möglich.
- einzelne Sichten: Die flexibelste Möglichkeit, da so für jede einzelne Sicht eine individuelle Aktualisierungsstrategie zugeordnet werden kann. Zur Vermeidung von Inkonsistenzen muss allerdings beachtet werden, dass alle von der betroffenen Sicht abhängigen Sichten ebenfalls aktualisiert werden.
- Teilmengen von Sichten: Um den Verwaltungsaufwand bei obigem Verfahren zu verringern werden Sichten zu Gruppen zusammengefasst, die einer bestimmten Aktualisierungsstrategie zugeordnet werden.

4.2 Anforderungen an ein Data-Warehouse-System

Die Methoden im vorherigen Abschnitt behandelten eine möglichst effiziente Aktualisierung einzelner Sichten. Die nun folgenden Anforderungen betten diese Methoden in ein Gesamtkonzept zur Aktualisierung des Data-Warehouse-Systems. Ziel ist es, die Gesamtperformance zu steigern, ohne Beeinträchtigung der Konsistenz der Daten. Zwei wichtige Anforderungen an ein Data-Warehouse-System:

- *Nebenläufige Aktualisierung*: Auf der einen Seite sollte das Data-Warehouse-System jederzeit für Anfragen zur Verfügung stehen. Andererseits muss es auf Änderungen der Basisrelationen reagieren und eine Aktualisierung der Sichten durchführen. Ziel ist es, eine möglichst kleine Einschränkung der Anfragebearbeitung während der Aktualisierung zu erreichen.
- *Unterstützung individueller Aktualisierungsstrategien*: Je nach Anwendung ist es nicht immer nötig, dass das komplette Data-Warehouse-System ständig auf den aktuellsten Stand ist. Je nach Aktualisierungsgranulat ist es möglich, die Aktualisierung in verschiedene Teilprozesse aufzuteilen. Die einzelnen Prozesse können dann je nach Anforderung der Anwendung aktualisiert werden.

Mit zunehmender Nebenläufigkeit steigt zwar die Performance des Data-Warehouse-Systems, allerdings wird auch die Komplexität des Aktualisierungsprozess immer größer. Ziel ist es, je nach Anforderung an das Data-Warehouse-System den besten Kompromiss zwischen Performance und Konsistenz zu finden.

4.3 Konzepte zur Sicherung der Konsistenz

Im Folgenden werden zwei Konzepte vorgestellt, die in kommerziellen Systemen eingesetzt werden und versuchen die vorherigen Forderungen zu erfüllen. Sie unterscheiden sich in der Anzahl der Versionen der Datenobjekte, auf die zugegriffen wird. Bei dem ersten Ansatz existiert nur eine Version von jedem Datenobjekt, im Gegensatz zum zweiten Verfahren, in dem von jedem Datenobjekt mehrere Versionen existieren können und dadurch die Nebenläufigkeit erhöht werden kann.

4.3.1 Aktualisierung im Einversionenfall

Bei der Aktualisierung im Einversionenfall werden die materialisierten Sichten nicht bei einer Änderung der Basisrelation aktualisiert, sondern erst wenn sie zu einer Anfrage

herangezogen werden. In dem so genannte Viewgroup Konzept [CLK+97] werden materialisierte Sichten mit gleicher Konsistenzanforderung zu einer Viewgroup zusammengefasst. Diese werden immer zusammen aktualisiert und befinden sich dadurch auf dem gleichen Aktualitätsniveau. Solange Anfragen sich nur auf eine Viewgroup beziehen, ist auch immer die Konsistenz gewährleistet. Während der Aktualisierung einer Viewgroup, ist auf diese kein Anfragezugriff möglich, allerdings kann auf alle anderen zugegriffen werden.

Der Vorteil ist, dass so eine gewisse Nebenläufigkeit erreicht wird, sowie individuelle Aktualisierungsstrategien unterstützt werden. Ein Problem ist, dass Konsistenz nur gewahrt werden kann, wenn keine Anfragen verwendet werden, die sich auf mehrere Viewgroups beziehen. Ein weiterer Nachteil ist, dass die Aktualisierung erst beginnen kann, wenn alle Lesezugriffe auf der Sicht beendet worden sind, dieses kann bei langen Anfragen erhebliche Verzögerungen hervorrufen.

4.3.2 Aktualisierung im Mehrversionenfall

Die Probleme beim Einversionenfall lassen sich durch das Einführen mehrerer Versionen eines Datenobjektes beseitigen. Jedes Mal, wenn ein Datenobjekt in der materialisierten Sicht geschrieben wird, wird eine neue Version dieses Datenobjekts erzeugt, welche vom Datenbanksystem verwaltet wird. Anfragen können so ältere Objekte lesen, obwohl parallel eine Aktualisierung des Objektes stattfindet. Diese wiederum müssen nicht warten, bis alle Lesezugriffe auf das Objekt beendet wurden, sondern können direkt mit dem Schreibvorgang beginnen. Dieses führt zu einer Erhöhung der Nebenläufigkeit und es werden keine Vorgänge mehr durch die Aktualisierung blockiert.

Allerdings kann es vorkommen, dass Anfragen auf veraltete Objekte zugreifen, dadurch ist eine Anwenderdefinierte Aktualitätsanforderung nicht möglich. Die erhöhte Nebenläufigkeit führt zu einer geringeren Aktualität der Daten.

Zusammenfassung

Durch die große Menge an auszuwertenden Daten und die hohe Komplexität der Anfragen zur Analyse von Daten, sind die Anforderungen an die Performance eines Data-Warehouse-Systems sehr groß. Durch den Einsatz von materialisierten Sichten ist es möglich die Antwortzeit auf Anfragen stark zu verringern. Allerdings treten durch den Einsatz auch wieder neue Probleme und Anforderungen auf.

Letztendlich gilt es, aufgrund der unterschiedlichen Anforderungen des Anwenders, für jeden Einsatzbereich den möglichst idealen Kompromiss zu finden, zwischen der geforderten Aktualität und Konsistenz der materialisierten Sichten und dem damit verbundenen mehr an Speicherplatzbedarf und Verwaltungsaufwand.

Die Entwicklung im Bereich der materialisierten Sichten ist aber noch lange nicht abgeschlossen. Insbesondere der Bereich der Selbstpflege von Sichten ist noch nicht ausgeschöpft. Das Ziel ist es neue Verfahren zu finden, die materialisierte Sichten mit möglichst wenigen Informationen möglichst effektiv und korrekt aktualisieren können. Zudem ist die Frage, welche Informationen noch zur Pflege verwendet werden können, wie zum Beispiel funktionale Abhängigkeiten innerhalb der Daten.

Es existieren oft viele verschiedene Verfahren und Ansätze zur Pflege der materialisierten Sichten. Alle Verfahren haben ihre Vor- und Nachteile. Die Frage ist, ob ein kostenbasiertes Modell entwickelt werden kann, dass situationsbedingt abschätzen kann, welches Verfahren zu einem bestimmten Zeitpunkt das günstigste ist und dieses anwendet.

Literatur

ABD+99 Dirk Ansorge, Klaus Bergner, Bernd Deifel, Nicholas Hawlitzky, Andreas Rausch, Marc Sihling, Veronika Thurner, and Sascha Vogel: Managing componentware development – software reuse and the V-Modell process. In *Proceedings of CAiSE '99*, Lecture Notes in Computer Science. Springer-Verlag, 1999

BaGü00 Bauer, Günzel (Hrg.): Data Warehouse – Architektur, Entwicklung, Anwendung; dpunkt.verlag, 2000

ChDa97 Chaudhuri, S.; Dayal, U.: An Overview of Data Warehousing and OLAP Technology. In: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD'1997, Tucson, USA, 13.-15. Mai), 1997

CLK+97 Cheung, D. W., Lee S. D., Kao, B.: A General Incremental Technique for Maintaining Discovered Association Rules. In DASFAA'97, Melbourne, Australia, 1997

GuHQ95 Gupta A.; Harinarayan, V.; Quass, D.: Aggregate-Query Processing in Data Warehousing Environments (Elektronisch verfügbar unter http://dbpubs.stanford.edu:8090/pub/1995-33)

GuMu95 Gupta, A.; Mumick, I. S.: In IEEE Data Engineering Bulletin, Special Issue on Materialized Views & Data Warehousing, June 1995

HaRU96 Harinarayan, V.; Rajaraman, A.; Ullman, J.: Implementing Data Cubes Efficiently, in: Proceedings of the 1996 ACM International Conference on Management of Data (SIGMOD 1996, Montreal, Canada, 4.-6. Juni), 1996

Huyn97 Huyn, N.: Multiple-View Self-Maintenance in Data Warehousing Environments (Elektronisch verfügbar unter http://www-db.stanford.edu/pub/papers/mvsm.ps)

HGWZ95 Zhuge, Y.; Garcia-Molina, H.; Hammer, J.; Widom, J.:View Maintenance in a Warehousing Environment, Proc. SIGMOD Conf., 1995

KoRo99 Kotidis, Y.; Roussopoulus, N.: A Dynamic View Managment System for Data Warehouses. In: Proceedings ACM SIGMOD International Conference on the Management of Data, 1999

LaQA97 Labio, W.; Quass, D.; Adelberg, B.: Physical Database Design for Data Warehouses, Proc. ICDE, 1997

Lehn98 Lehner, W. (Hrsg.): Erweiterte Konzepte und Techniken der Anfrageoptimierung in Datenbanksystemen. Arbeitsbericht des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik), Friedrich-Alexander Universität Erlangen-Nürnberg, 1998

ScSV99 Scheuermann, P; Shim, J.; Vingralek, R.: A Data Warehouse Intelligent Cache Manager. In: Proceedings of 22th International Conference on Very Large Data Bases, 1996

WeCa96 Wells, D.; Carnelly Philip: Ovum Evaluates: The Data Warehouse; IT-Verlag 1996