

Otto-von-Guericke-Universität Magdeburg



Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

Diplomarbeit

Dynamische Ermittlung und Verwaltung von materialisierten Sichten auf Grundlage des Query Graph Models

Verfasser:

Ronny Bubke

6. September 2007

Betreuer:

Dr.-Ing. Eike Schallehn

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D-39016 Magdeburg
Germany

Bubke, Ronny:

*Dynamische Ermittlung und Verwaltung von
materialisierten Sichten auf Grundlage des
Query Graph Models*

Diplomarbeit, Otto-von-Guericke-Universität
Magdeburg, 2007.

Inhaltsverzeichnis

Inhaltsverzeichnis	i
Abbildungsverzeichnis	v
Tabellenverzeichnis	ix
Verzeichnis der Abkürzungen	xi
1 Einführung	1
1.1 Einleitung	1
1.2 Zielsetzung der Arbeit	3
1.3 Gliederung der Arbeit	3
2 Grundlagen	5
2.1 WATCHMAN: A Data Warehouse Intelligent Cache Manager	6
2.1.1 Cache Replacement (LNC-R)	6
2.1.2 Cache Admission (LNC-A)	7
2.1.3 Kombiniertes Verfahren (LNC-RA)	7
2.2 Verwendung von Anfrageplänen	10
2.2.1 Aufbau des Anfrageplans	11
2.3 Query Graph Model	13
2.3.1 Aufbau des Query Graph Models	13
2.3.2 Zerlegung von Anfragen	14
2.3.3 Matching	15
2.3.4 Matchfunktion	16
2.3.5 Navigator	17
2.3.6 Muster des Query Graph Models	17
2.3.7 Exakter Kind-Match	17

2.3.8	Nicht exakter Kind-Match	22
2.4	Zusammenfassung	31
3	Konzept	33
3.1	Einleitung	33
3.2	Erweiterung der Box-Typen	34
3.2.1	Attributeigenschaften	34
3.2.2	Allgemeine Eigenschaften einer Box	34
3.2.3	Tabellen-Box	35
3.3	Zerlegung und Matching	35
3.4	Mapgraph	36
3.4.1	Aufbau	36
3.4.2	Einfügen einer Box	37
3.4.3	Entfernen einer Box	38
3.5	Navigator	39
3.5.1	Anfragebehandlung	39
3.5.2	Restrukturierung	42
3.5.3	Merging	45
3.6	Muster der Matchfunktion	46
3.6.1	Merging von Select-Boxen	46
3.6.2	Merging von Group-By-Boxen	47
3.6.3	Intervalprädikate	49
3.6.4	Konjunktionen und Disjunktionen	51
3.7	Aging	52
3.8	Profitberechnung	55
3.9	Auswahl der materialisierten Sichten	57
3.10	Indexe	58
3.11	Zusammenfassung	61
4	Implementation und Evaluation	63
4.1	Einleitung	63
4.2	Programmablauf	63
4.2.1	Kompilierung von SQL in QGM	64

4.2.2	Matching der Anfrage	64
4.2.3	Profitberechnung und Ersetzungsverfahren	65
4.3	Verwaltungstools	65
4.3.1	Tabellenverwaltung	65
4.3.2	Darstellung des MapGraphs	66
4.3.3	Absetzen von Anfragen	66
4.3.4	Workload-Emulator	67
4.4	Overhead des Verfahrens	68
4.4.1	Größe des Mapgraphs	68
4.4.2	Traversierung des Mapgraphs	68
4.4.3	Matching	69
4.5	Stresstest (TPC-H)	69
4.5.1	Workloadauswahl	69
4.5.2	Ergebnisse DB2 Design Advisor	75
4.5.3	Ergebnisse Konzept-Applikation	81
4.6	Vorhersagequalität	85
4.6.1	Workload mit vielen Punktanfragen	85
4.6.2	Workload mit wenigen Punktanfragen	85
4.6.3	Vergleich der Verfahren	85
4.7	Zusammenfassung	86
5	Fazit und Ausblick	87
	Literaturverzeichnis	89

Abbildungsverzeichnis

2.1	Optimierung in Anfragebearbeitung [13]	10
2.2	Anfrage des Anfrageplans	11
2.3	Anfrage mit Verwendung einer Sicht	11
2.4	Sicht <i>AUSLEIH_INFO</i>	11
2.5	Darstellung eines unoptimierten Anfrageplans[13]	12
2.6	Anfrage Q1 - Selektion von länderbezogenen Transaktionsdaten	14
2.7	Anfrage Q1 als QGM-Graph	15
2.8	Die Matching-Beziehung zwischen Subsumee und Subsumer	16
2.9	Anfrage Q2 - eigentliche Anfrage	18
2.10	Anfrage AST2 - materialisierte Sicht	18
2.11	Anfrage NeuQ2 - optimierte Anfrage	19
2.12	SELECT-Box mit exaktem Kind-Match	19
2.13	Anfrage Q4 - eigentliche Anfrage	21
2.14	Anfrage AST4 - materialisierte Sicht	21
2.15	Anfrage NeuQ4 - optimierte Anfrage	21
2.16	GROUP-BY-Box mit exaktem Kind-Match	22
2.17	Anfrage Q6 - eigentliche Anfrage	23
2.18	Anfrage AST6 - materialisierte Sicht	23
2.19	Anfrage NeuQ6 - optimierte Anfrage	24
2.20	GROUP-BY-Boxen mit einfacher SELECT-Box-Kompensation	24
2.21	Anfrage Q7 - eigentliche Anfrage	25
2.22	Anfrage AST7 - materialisierte Sicht	25
2.23	Anfrage NeuQ7 - optimierte Anfrage	25
2.24	GROUP-BY-Boxen mit Rejoin-Kompensation	26
2.25	GROUP-BY-Boxen mit GROUP-BY-Kind-Kompensation (allg. Form)	27

2.26	Anfrage Q8 - eigentliche Anfrage	28
2.27	Anfrage AST8 - materialisierte Sicht	28
2.28	Anfrage NeuQ8 - optimierte Anfrage	29
2.29	GROUP-BY-Boxen mit GROUP-BY-Kind-Kompensation	30
3.1	Eigenschaften einer Box	35
3.2	Darstellung einer Table-Box	36
3.3	MapGraph mit einer TabellenBox und zwei konsumierenden Select-Boxen	36
3.4	Einfügung in den Abbildungsgraph	37
3.5	MapGraph nach Einfügung	38
3.6	Mehrdeutigkeit beim Matching	39
3.7	Löschen einer Box	40
3.8	MapGraph nach dem Entfernen des Teilastes	40
3.9	Einfügen einer Anfrage in den Mapgraph	41
3.10	Mapgraph nach der Einfügung	42
3.11	Matchsituation beim Restrukturieren	44
3.12	Matchsituation nach dem Restrukturieren	44
3.13	Merging von Selektionsboxen	48
3.14	Mapgraph nach Merging von Select-Boxen	48
3.15	Merging von Gruppierungsboxen	50
3.16	Mapgraph nach Merging von Group-By-Boxen	50
3.17	Anfrage in konjunktiver Form	51
3.18	Anfrage in disjunktiver Form	51
3.19	Anfrage, die gematcht werden soll	52
3.20	Match disjunktiver Prädikatmengen	52
3.21	Mapgraph vor der Aging-Operation	55
3.22	Mapgraph mit dekrementierten Boxreferenzen	55
3.23	Ermittlung der Indexkandidaten	59
3.24	Materialisierte Sicht mit Elternboxen	60
4.1	TPC-H Query 1	70
4.2	TPC-H Query 2	70
4.3	TPC-H Query 3	71

4.4	TPC-H Query 4	71
4.5	TPC-H Query 5	71
4.6	TPC-H Query 6	71
4.7	TPC-H Query 7	72
4.8	TPC-H Query 8	72
4.9	TPC-H Query 8-1	72
4.10	TPC-H Query 8-2	72
4.11	TPC-H Query 8-3	73
4.12	TPC-H Query 8-4	73
4.13	TPC-H Query 8-5	73
4.14	DB2 - AST-Vorschlag-1	76
4.15	DB2 - AST-Vorschlag-2	77
4.16	DB2 - AST-Vorschlag-3	78
4.17	DB2 - AST-Vorschlag-4	79
4.18	DB2 - AST-Vorschlag-5	80
4.19	DB2 - AST-Vorschlag-6	80
4.20	Konzeptapplikation - AST-Vorschlag-1	82
4.21	Konzeptapplikation - AST-Vorschlag-2	82
4.22	Konzeptapplikation - AST-Vorschlag-3	82
4.23	Konzeptapplikation - AST-Vorschlag-4	83
4.24	Konzeptapplikation - AST-Vorschlag-5	83
4.25	Konzeptapplikation - AST-Vorschlag-6	83
4.26	Konzeptapplikation - AST-Vorschlag-7	84
4.27	Konzeptapplikation - AST-Vorschlag-8	84

Tabellenverzeichnis

3.1	Allgemeine Eigenschaften einer Box	35
3.2	Aging von Boxen	54
3.3	Auswahl von materialisierten Sichten	58
4.1	Laufzeit des TPC-H Workloads ohne Verwendung von ASTs	74
4.2	Laufzeit des TPC-H Workloads bei Verwendung der DB2-ASTs	75
4.3	Laufzeit des TPC-H Workloads bei Verwendung der Konzept-ASTs	81

Verzeichnis der Abkürzungen

AST	Materialisierte Sicht
DBMS	Datenbankmanagementsystem
DNF	Disjunktive Normalform
KNF	Konjunktive Normalform
LNC-A	Least Normalized Cost Admission
LNC-R	Least Normalized Cost Replacement
LNC-RA	Least Normalized Cost Replacement Admission
QGM	Query Graph Model
SQL	Structured Query Language
TPC-H	Transaction Processing Performance Council

Kapitel 1

Einführung

1.1 Einleitung

Der Anspruch, der in einer rechtsstaatlichen Demokratie lebenden Gesellschaft an die Wissenschaft, liegt in der Verbesserung der Lebensbedingungen für jeden darin existierenden Menschen. Um diesem Anspruch gerecht zu werden, versucht jedes Individuum unbefriedigende Aufgaben zu vereinfachen bzw. sie ganz zu vermeiden. Arbeiten, die den Grundbedürfnissen in der heutigen Zeit in übertragenem Sinne dienen, werden, soweit möglich, durch Maschinen erledigt. Der Mensch kann sich dadurch neuen Herausforderungen stellen, die seine Lebensbedingungen in gewisser Weise weiter verbessern.

Waren es in der Eisenzeit noch einfache Handwerksfertigkeiten, die eine Spezialisierung eines Kulturkreises ausmachten, ist das Wissen derzeit viel granulierter verteilt. Dieser Trend setzt sich durch den Erwerb von immer neuem Wissen fort und führt zu einer immer spezialisierteren Gesellschaft, in der immer mehr Menschen das Spezialwissen einer Quelle nutzen möchten.

Dies spiegelt sich in der Geschäftswelt wieder, wo Unternehmen spezialisierte Produkte anbieten und diese an eine breite Masse von Kunden verkaufen. Die Geschäftsdaten solcher Unternehmen müssen durch den gesetzlichen Zwang und dem Wunsch nach Kontrolle und Verbesserung des Geschäftslebens gespeichert und ausgewertet werden. Ohne Computer und entsprechender Software wäre diesem Wunsch in der heutigen Zeit nur schwer nachzukommen, da die Datenmenge manuell kaum auswertbar ist. Es werden automatische Informationssysteme in Form von Soft- und Hardware genutzt, die dem Menschen diese triviale und aufwendige Aufgabe abnehmen, so dass er die Ergebnisse nur noch kognitiv weiterverarbeiten muss und daraus eine weitere Verbesserung erzielen kann.

Automatische Informationssysteme benötigen Abrufsysteme, die Daten möglichst schnell berechnen und integer zur Verfügung stellen. Meist sind diese in Form von Datenbankmanagementsystemen (DBMS) realisiert, die von der Applikation benötigte Daten verwalten. Dabei wird das DBMS nicht nur zur Speicherung der Daten genutzt, sondern auch zur Berechnung komplexer Operationen auf Grundlage dieser Daten. Stellt man sich Datenbanksysteme vor, die sehr große Datenmengen im Terrabytebereich verwalten, führt dies schnell zur Notwendigkeit der Optimierung dieser Operationen. Dabei gilt sowohl die logische Reihenfolge der Operationen, als auch den physischen Zugriff auf die Daten selbst zu optimieren.

Eine Methode der Optimierung ist die Vorberechnung von Operationen, die häufig beim System angefragt werden. Die vorberechneten Daten werden als materialisierte Sichten[9] (AST¹) bezeichnet und in relationalen DBMS als Ergebnistabelle von Operationen gespeichert. Die an das System gestellten Anfragen müssen daraufhin so geändert werden, dass sie die materialisierte Sicht anstatt der semantisch äquivalenten Operation verwenden. Damit entfällt die Zeit, die zur Berechnung dieser Operation nötig wäre, was insgesamt zu einer schnelleren Antwortzeit des Systems führt.

Die Definition einer materialisierten Sicht ist abhängig von den Anforderungen der Applikation, die das DBMS nutzt. Ist eine Anfrage für die Applikation zeitkritisch, so kann eine AST angelegt werden, welche die Beantwortung der Anfrage beschleunigt. Die Anfragen, welche die materialisierte Sicht nutzen, müssen daraufhin um die Verwendung der AST aktualisiert werden. Zusätzlich können Zugriffspfade[13] angelegt werden, die eine weitere Optimierung realisieren.

Problematisch bei der Verwendung von materialisierten Sichten ist die manuelle Erstellung dieser und eventueller Zugriffspfade. Dabei spielt der Zeitaufwand beim Erstellen, als auch eine eventuell nicht optimale Wahl der Sichtdefinition und entsprechender Zugriffspfade eine Rolle. Weiterhin kann sich der Workload² und die Abbildung des Datenbestandes im Laufe der Zeit ändern. Daraufhin müssten dann auch wieder die ASTs geändert werden, was bei dynamischen Applikationen einen ständigen und eventuell fehlerbehafteten manuellen Verwaltungsaufwand bedeuten würde.

Wünschenswert wäre ein System, welches materialisierte Sichten automatisch aufgrund des Datenbestandes und Anfragepools erzeugt und verwaltet. Dazu wird eine Abbildung der Operationen des Workloads benötigt, die es ermöglicht, Materialisierungsentscheidungen zu treffen.

Um die Dynamik des Workloads zu berücksichtigen, wird eine Strategie benötigt, die Anfragen, die erst seit Kurzem im Workload auftauchen, besser bewertet, als solche, die schon länger darin befindlich sind. Dies hat zur Folge, dass schneller auf Workloadänderungen in Bezug auf die Aktualisierung der materialisierten Sichtdefinitionen reagiert werden kann.

Weiterhin sinnvoll ist die automatische Erzeugung der Zugriffspfade auf existierende materialisierte Sichten, um den Zugriff auf diese zu beschleunigen. Dazu ist es notwendig, Operationen, die aufbauend auf die materialisierten Sichten berechnet werden, zu bewerten und geeignete Kandidaten auszuwählen.

Es existieren Werkzeuge einiger DBMS-Hersteller, die das Datenbankdesign vereinfachen, beschleunigen und weniger fehleranfällig machen. Dazu analysiert das Programm den Workload des Systems und entscheidet anhand von statistischen Informationen, welche ASTs vorgeschlagen werden. Dabei werden Techniken verwendet, die zuvor in der IT-Forschung entwickelt wurden. Es werden noch lange nicht alle Möglichkeiten genutzt, die eine optimale Sichtdefinition erlauben. Daher werden viele dynamische Applikation immer noch durch manuelle Erzeugung von ASTs optimiert.

¹Automatic Summary Table

²Menge von SQL-Anfragen

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist die Entwicklung eines Konzeptes, dass sowohl die dynamische Erzeugung von materialisierten Sichten, als auch deren Zugriffspfade ermöglicht und damit die Techniken bisheriger Optimierungsverfahren ergänzt bzw. verbessert.

Es soll eine Abbildung des Workloads gefunden werden, der eine anschließende Bewertung von Anteilen daraus erlaubt. Dazu müssen Gemeinsamkeiten von Anfragen ermittelt und diese stärker bei der Bewertung berücksichtigt werden.

Weiterhin soll das Problem der Workloadänderung bei dynamischen Applikationen gelöst werden, durch die sich die optimale Sichtdefinition ändern kann. Dadurch müssen die ASTs auf die neue Sichtdefinition angepasst werden.

1.3 Gliederung der Arbeit

Die Arbeit untergliedert sich in 5 Kapitel. Das erste Kapitel gibt eine Einführung in die Problematik dieser Arbeit.

Im zweiten Kapitel werden die Grundlagen zum Konzept erläutert. Dabei werden sowohl existierende Verfahren, als auch Voraussetzungen zur Konzeptionierung vorgestellt.

Das dritte Kapitel stellt das Konzept des Optimierungsverfahrens vor. Dabei werden Vorgehensweisen aus dem Grundlagenkapitel genutzt und erweitert.

Die Evaluation und Implementierung des Konzeptes werden im vierten Kapitel vorgestellt. Dabei wird die konzeptionelle Implementation dieser Arbeit mit anderen Ansätzen verglichen.

Das fünfte und letzte Kapitel gibt ein Fazit und einen Ausblick für zukünftige Arbeiten.

Kapitel 2

Grundlagen

Dieses Kapitel beschäftigt sich mit den Grundlagen dieser Arbeit. Im ersten Abschnitt wird ein Verfahren zur dynamischen Verwaltung von materialisierten Sichten [3] vorgestellt, die auf Speicherung, Ersetzung und Profitberechnung von Relationen beruht. Dabei werden zwei Methoden zu einer weiteren kombiniert.

Der Abschnitt 2.2 diskutiert die Verwendung von Anfrageplänen. Diese werden zur Optimierung der Anfragen unter Berücksichtigung der logischen und physischen Optimierung, sowie zur kostenbasierten Auswahl eines DBMS benötigt.

Der Hauptteil des Grundlagenkapitels beschäftigt sich mit dem Query Graph Model (QGM). Dieses ist, ebenso wie der Anfrageplan, ein Abbildungsmodell für Anfragen. Ähnlich wie beim Anfrageplan wird beim QGM eine Anfrage als Baum [2] abgebildet. Ziel der Benutzung des QGM ist die Realisierung der dynamischen Anfragebearbeitung insofern, dass möglichst vorhandene AST's automatisch als Bestandteile der Anfrage bei der Ausführung genutzt werden und somit der Berechnungsaufwand einer Teilanfrage verbessert wird.

Dabei werden Muster benötigt, die eine Überprüfung der Teilmengenbedingung von Relationen garantieren. Diese werden im darauf folgenden Abschnitt vorgestellt.

2.1 WATCHMAN: A Data Warehouse Intelligent Cache Manager

WATCHMAN [14] ist ein Verfahren zur dynamischen Verwaltung von materialisierten Sichten [3]. Die materialisierten Sichten werden automatisch erstellt und bei Anfragebearbeitung genutzt.

Der Workload des Systems, der aus einer Menge von Anfragen besteht, soll möglichst optimal auf ASTs abgebildet werden und somit die Antwortzeit des Systems einer Anfrage verkürzen. Dazu nutzt die Anfrage die Vorberechnung der materialisierten Sicht, so dass zur Ausführungszeit der Anfrage diese von der AST repräsentierte Operation nicht mehr berechnet werden muss.

Es existiert ein Cache, der die ASTs speichert, und ein Ersetzungsalgorithmus LNC-R (Least Normalized Cost Replacement), der bestehende ASTs durch profitablere ersetzt. Weiterhin existiert ein Algorithmus LNC-A (Least Normalized Cost Admission), der zu ersetzende ASTs auf Antwortzeitverschlechterung überprüft und auswählt. Resultierend existiert eine Kombination LNC-RA (Least Normalized Cost Replacement Admission) aus LNC-R und LNC-A, die beide Vorteile vereint und damit eine verbesserte Abbildungsstrategie ermöglicht. Alle drei Algorithmen haben das Ziel, die Antwortzeit des Systems zu verkürzen.

2.1.1 Cache Replacement (LNC-R)

Der Cache-Ersetzungsalgorithmus basiert auf der Bewertung jeder einzelnen Anfrage Q_i aufgrund der Referenzrate, sich ergebene Relationsgröße und Ausführungskosten:

- λ_i : Durchschnittliche Referenzrate der Anfrage Q_i
- s_i : Größe der Relation erzeugt durch Anfrage Q_i
- c_i : Ausführungskosten von Anfrage Q_i

Jede Anfrage wird mittels folgender Profitfunktion bewertet, wobei RS_i der Ergebnisrelation der Anfrage Q_i entspricht:

$$profit(RS_i) = \frac{\lambda_i * c_i}{s_i}$$

Die Liste der Anfragen wird nach den jeweiligen Profiten sortiert und je nach Implementation ausgewertet. Dazu existieren mehrere Möglichkeiten. Zum Einen kann verlangt werden, dass anhand eines Größenwertes die profitabelsten Anfragen zurückgeliefert werden. Dazu iteriert der Algorithmus die Liste absteigend. Falls die Größe der repräsentierten Relation einer Anfrage kleiner als der Größenwert ist, wird die von ihr repräsentierte Relationsgröße vom Größenwert abgezogen. Falls nicht, wird die Anfrage aus der Liste gelöscht. Danach folgt die in der Liste befindliche nächste Anfrage bis zum Erreichen des Listenendes. Weiterhin kann gefordert werden, dass die unprofitabelsten Anfragen in der Liste anhand eines Größenwertes ermittelt werden. Die Vorgehensweise ist die Gleiche, wie bei der Ermittlung der profitabelsten Anfragen, nur, dass die Liste vom Ende ausgehend iteriert wird.

2.1.2 Cache Admission (LNC-A)

Das Hauptziel von Cache Admission ist die Vermeidung der Speicherung von Anfragen, welche die Antwortzeit des Systems insgesamt verschlechtern. Zum Beispiel könnte die Speicherung einer Mehrspaltenprojektion einer großen Relation die gesamte Cachegröße einnehmen, so dass für kleinere, komplex zu berechnende Statistiktabelle keine Möglichkeit zur Speicherung mehr existiert. Die Neuberechnung zur Anfragezeit der Statistiktabelle würde die Antwortzeit insgesamt verschlechtern.

Im Idealfall sollten nur Anfragen gespeichert werden, die den Gesamtprofit des Systems verbessern. Gegeben ist eine Menge C von Ersetzungskandidaten für eine erhaltene Relationsmenge RS_i . WATCHMAN speichert RS_i nur im Cache, wenn der Profit von RS_i höher als der von C ist:

$$profit(RS_i) > profit(C)$$

Wobei der Profit der Liste von C definiert ist als:

$$profit(C) = \frac{\sum_{RS_j \in C} \lambda_j * c_j}{\sum_{RS_j \in C} s_j}$$

Für Anfragen, die das erste Mal im Workload auftauchen und für die noch keine Referenzrate existiert, muss der Profit ohne Berücksichtigung der Referenzrate berechnet werden:

$$e_profit(RS_i) = \frac{c_i}{s_i}$$

WATCHMAN speichert RS_i nur, wenn die folgende Ungleichung erfüllt ist:

$$e_profit(RS_i) > e_profit(C)$$

Wobei der geschätzte Profit von C sich wie folgt ergibt:

$$e_profit(C) = \frac{\sum_{RS_j \in C} c_j}{\sum_{RS_j \in C} s_j}$$

2.1.3 Kombiniertes Verfahren (LNC-RA)

LNC-RA nutzt die Vorteile von LNC-R und LNC-A kombiniert. LNC-A greift im Zuge der Ermittlung der Ersetzungskandidaten in C auf LNC-R zurück. Der Profit der Relation RS_i muss größer sein, als der Profit von C . Der komplette Algorithmus ist in Listing 2.1 abgebildet.

Dabei erhält LNC-RA vier Eingabeparameter, die Relation RS_i , welche gecacht werden soll, deren Größe s_i und Berechnungskosten c_i sowie den verfügbaren Speicherplatz des Caches *avail*. Es existieren zwei Variablen, die einmal die Referenzrate ri_i im Cache und einmal die Referenzrate λ_i der Anfrage im Workload repräsentieren. Der Algorithmus prüft drei Fälle und reagiert entsprechend. Existiert RS_i bereits im Cache, so wird nur der Referenzzähler des Caches für die Relation inkrementiert. Liegt RS_i nicht im Cache, und steht noch genug freier Cachespeicher zur Verfügung, so wird die Relation in den Cache kopiert und der Cachereferenzzähler aktualisiert. Der dritte eintretende Fall führt zur Ausführung von LNC-A mit RS_i als Übergabeparameter, genau dann, wenn

RS_i noch nicht im Cache liegt und nicht genug Speicher zum Cachen zur Verfügung steht.

In diesem Fall prüft LNC-A, ob der Profit von RS_i größer als der der Ersetzungskandidaten, ermittelt durch LNC-R, ist. Dazu wird als Erstes geprüft, ob ri_i bereits im Cache vorkommt. Dies ist der Fall, falls RS_i bereits in der Vergangenheit im Cache vorkam aber möglicherweise aus Gründen einer Workloadänderung durch andere ASTs ersetzt wurde. In diesem Fall wird der Cachereferenzähler ri_i aktualisiert und danach geprüft, ob der Profit der Relation größer als der der Ersetzungskandidaten ist. Ist dies der Fall, so werden die Ersetzungskandidaten aus dem Cache entfernt und RS_i in den Cache kopiert. Existiert noch kein Cachereferenzähler ri_i , so wird dieser erstellt und aktualisiert. Danach folgt der gleiche Ablauf, nur mit e_profit als Profitfunktion.

Zur Ermittlung der Ersetzungskandidaten wird, wie bereits erwähnt, LNC-R genutzt. Dieser erhält als Eingabeparameter den Wert, um den der Cache geleert werden soll. Dazu werden die Relation nach ihrer Referenzrate aufsteigend sortiert und bis zum Erreichen von Wert s in die Liste C kopiert. Diese enthält nun alle Relationen des Caches mit den geringsten Referenzraten, die der Bedingung $\sum_{RS_j \in C} s_j \geq s$ genügen. Die Liste wird an LNC-A zurückgegeben und dort, wie beschrieben, weiterbehandelt.

Listing 2.1: LNC-RA

```

Algorithm : LNC-RA

Input :
  retrieved set  $RS_i$ 
   $s_i$  - size of  $RS_i$ 
   $c_i$  - cost of execution of query  $Q_i$  corresponding to  $RS_i$ 
   $avail$  - available free space in cache

Variables :
   $ri_i$  - reference information holding last  $K$  reference times to  $RS_i$ 
   $\lambda_i$  - estimate of average inter-arrival rate of references to  $RS_i$  calculated from  $ri_i$ 

case (allocation state of  $RS_i$ )
   $RS_i$  in cache:
    update  $ri_i$ 
   $RS_i$  not in cache and  $avail \geq s_i$ :
    cache  $RS_i$ 
    update  $ri_i$ 
   $RS_i$  not in cache and  $avail < s_i$ :
    LNC-A( $RS_i$ )

Algorithm : LNC-A

Input :
  retrieved set  $RS_i$ 

 $C = \text{LNC-R}(s_i)$ 
if ( $ri_i$  in cache) then
  update  $ri_i$ 
  if ( $\text{profit}(RS_i) > \text{profit}(C)$ ) then
    evict all retrieved sets in  $C$ 
    //retain reference information
    cache  $RS_i$ 
  fi
else
  allocate  $ri_i$ 
  update  $ri_i$ 
  if ( $e\_profit(RS_i) > e\_profit(C)$ ) then
    evict all retrieved sets in  $C$ 
    //retain reference information
    cache  $RS_i$ 
  fi
fi

Algorithm : LNC-R

Input :
   $s$  - space to be freed

Output :
   $C$  - list of candidate retrieved sets for replacement

for  $i=1$  to  $K$  do
   $R_i =$  list of retrieved sets with exactly  $i$  references in  $ri$ 
  arranged in increasing order
od
 $R =$  list of all retrieved sets arranged in order  $RS_1 < RS_2 < \dots < RS_K$ 
 $C =$  minimal prefix of  $R$  such that  $\sum_{RS_j \in C} s_j \geq s$ 

return  $C$ 

```

2.2 Verwendung von Anfrageplänen

Ein wichtiger Bestandteil eines DBMS ist die Optimierung der an das System gestellten Anfragen. Dazu ist es nötig, die Optimierung[10] der Anfrage in mehrere Schritte zu unterteilen. Dabei wird zuvor die Anfrage von SQL in eine algebraische Struktur umgewandelt, die Sichten werden aufgelöst und der Algebraterm dem Optimierer übergeben[13]. Nach der Optimierung wird aus dem Algebraterm erzeugten Anfrageplan Code generiert

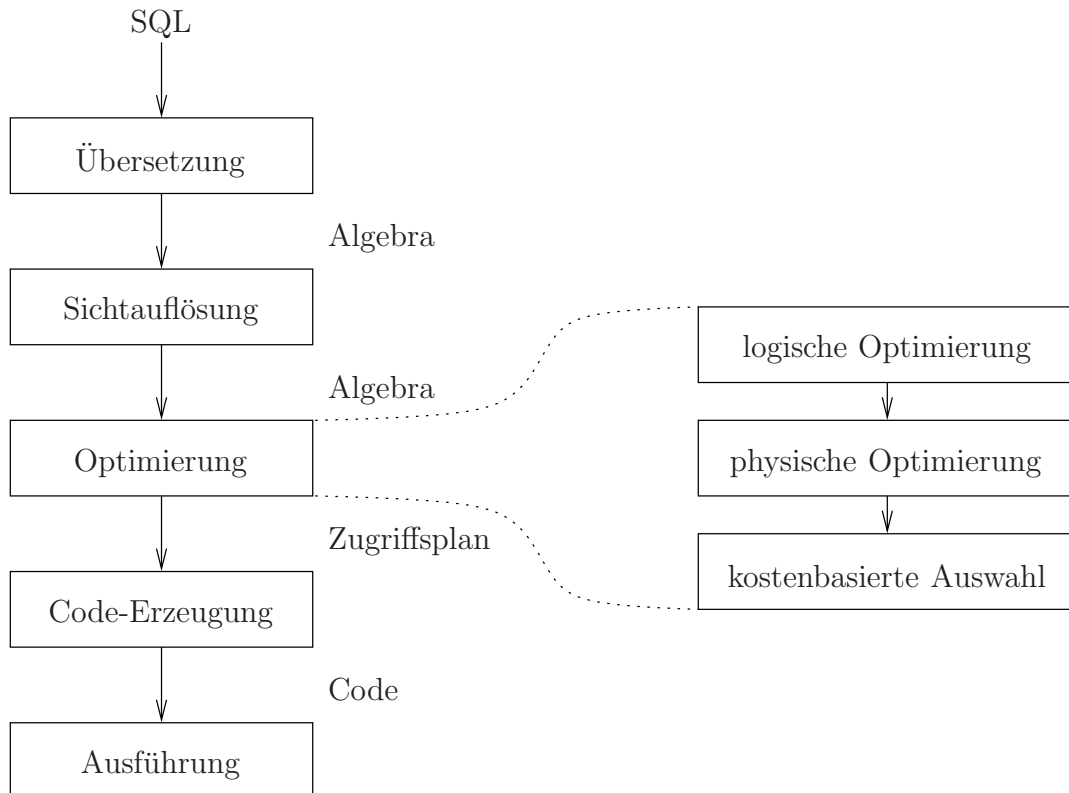


Abbildung 2.1: Optimierung in Anfragebearbeitung [13]

und dieser ausgeführt. Der Ablauf der Anfragebearbeitung ist schematisch in Abbildung 2.1 dargestellt. Das aus dem Code erzeugte Ergebnis wird weiter verarbeitet und dem Anfragersteller übermittelt.

Die Optimierung der Anfrage wird in drei Schritten durchgeführt. Zuerst wird eine logische Optimierung ausgeführt, die unter anderem eine algebraische Umstrukturierung vollzieht[13]. Danach erfolgt die physische Optimierung, die durch Ermittlung von Zugriffspfaden die physische Speicherung mitberücksichtigt und den Algebraterm in eine Menge von möglichen Anfrageplänen umwandelt. Daraus resultieren verschiedene Zugriffspläne, die anhand von Statistiken, wie z. B. Relationsgröße und Selektivität bewertet werden, wobei davon ein Zugriffsplan kostenbasiert ausgewählt wird. Der gewählte Zugriffsplan wird in ausführbaren Code umgewandelt und von einem Interpreter ausgeführt.

2.2.1 Aufbau des Anfrageplans

Der Anfrageplan entspricht einem Baum, dessen Knoten Operationen als auch Relationen bzw. Tabellen der Anfrage repräsentieren. Dabei entsprechen die Blattknoten den Tabellen und die inneren Knoten den Operationen. Die Operationen sind unter anderem vom Typ Projektion, Verbund, Selektion, Gruppierung, Schnitt, Differenz und Vereinigung. Die Knoten sind so angeordnet, dass die Anfrage semantisch korrekt repräsentiert wird. Abbildung 2.5 zeigt einen Anfrageplan der die Anfrage in Listing 2.2 algebraisch korrekt abbildet.

```
SELECT      Titel
FROM        Ausleihe, Entleiher, Bücher
WHERE       Datum = '1.1.88'
           and Autor='Heuer'
```

Abbildung 2.2: Anfrage des Anfrageplans

Hierbei fällt auf, dass eine weitere Projektion als innere Operation existiert. Dies lässt darauf schließen, dass eine Sicht verwendet und diese zuvor aufgelöst wurde. Die vom Benutzer gestellte Anfrage könnte der in Listing 2.3 entsprechen. Es wird eine Sicht *AUSLEIH_INFO* genutzt, deren Definition in Listing 2.4 dargestellt ist.

```
SELECT      Titel
FROM        AUSLEIH_INFO
WHERE       Datum = '1.1.88'
           and Autor='Heuer'
```

Abbildung 2.3: Anfrage mit Verwendung einer Sicht

```
SELECT      Titel, Autor, Verlag, ISBN
FROM        Ausleihe, Entleiher, Bücher
```

Abbildung 2.4: Sicht *AUSLEIH_INFO*

Die zwei Projektionen können nach Verschieben der Selektionsprädikate unter den Verbundoperationen zusammengefasst werden. Weiterhin können Projektionen eingefügt werden, die Zwischenrelationen bei der weiteren Berechnung verkleinern.

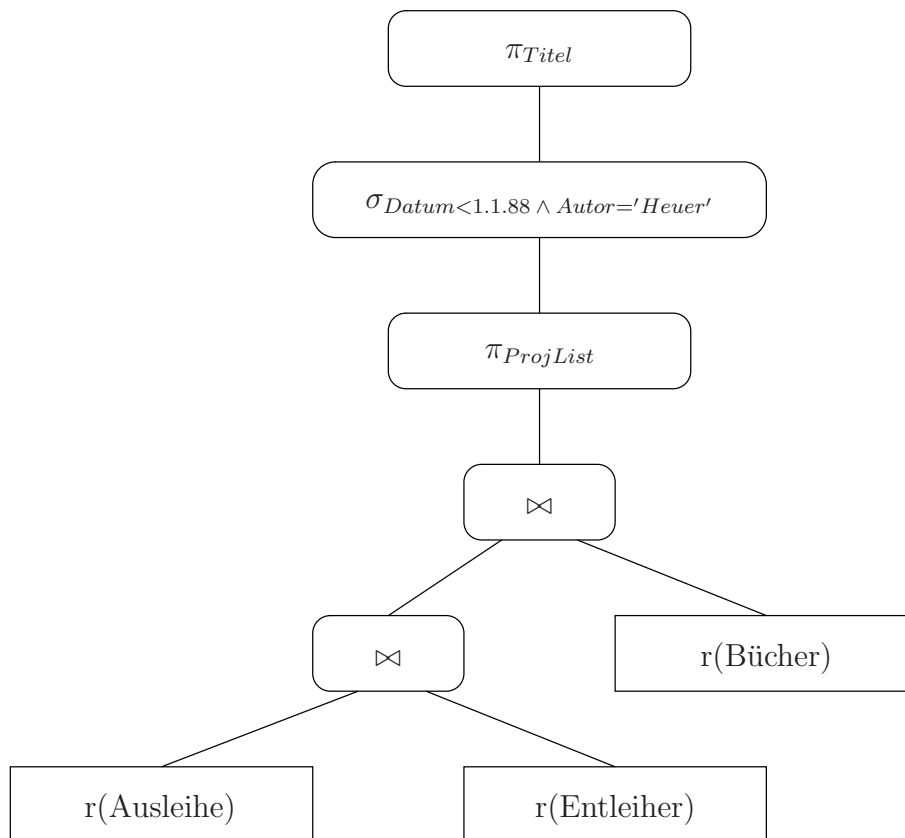


Abbildung 2.5: Darstellung eines unoptimierten Anfrageplans[13]

2.3 Query Graph Model

Das Query Graph Model [19] ist eine Darstellungsform einer SQL-Anfrage, bei der versucht wird, Teile einer Anfrage auf sogenannte Boxen abzubilden. Mit Hilfe dieser Boxen wird versucht, materialisierte Sichten [9] anstelle von Basistabellen für die Anfrageverarbeitung zu finden. Der Optimierer soll eine Anfrage so umstrukturieren, dass die Semantik erhalten bleibt, aber die Zeit für die Anfragebearbeitung sinkt.

Der Query Graph ist dabei ähnlich dem im vorherigen Abschnitt 2.2 vorgestellten Anfrageplan. Die Anfrage wird hierbei auch als Baum dargestellt, wobei die Operationen des Anfrageplans zusammengefasst werden und die Knoten zusätzlich ihre Projektion kennen. Die Ähnlichkeit der beiden Verfahren lässt eine einfache Konvertierung des Anfrageplans ins Query Graph Model zu. Somit steht auch der gemeinsamen Verwendung im Optimierer nichts entgegen.

Ein Beispiel für eine Umstrukturierung wäre das Ersetzen von Teilanfragen durch materialisierte Abfragetabellen. Da die materialisierte Sicht aus vorberechneten Daten besteht, entfällt die Zeit, die für die Berechnung aus den Basistabellen notwendig wäre.

Das QGM wurde entwickelt, um die Verwendung von AST bei der Optimierung von Anfragen zu realisieren. Es basiert auf einem Algorithmus, der auf Teilanfragen materialisierte Sichten *matcht* (Match).

Für eine komplette AST-Lösung müssen weitere Probleme betrachtet werden: a) finde die beste Auswahl von ASTs für jeden Workload unter Speicher- und/oder Update-Overhead Gesichtspunkten, b) welche AST soll aktuell genutzt werden, um eine Anfrage zu beantworten, und c) effiziente Verwaltung von ASTs, wenn Basistabellen aktualisiert werden. Beispiele von existierenden Arbeiten, die diese Probleme bearbeiten, werden in [8], [4] und [12] erläutert. In [7], [16] und [1] werden Ansätze vorgestellt, die das gleiche Ziel verfolgen, wie das Thema dieses Abschnitts. Die Unterschiede werden in [19] diskutiert.

2.3.1 Aufbau des Query Graph Models

Das Modell entspricht einem gerichteten, azyklischen Graph mit einem Wurzelknoten, wobei die Blattknoten die Basistabellen, die inneren Knoten Tabellenoperationen und die Kanten den Fluss der Tupel vom Kindknoten zum Elternknoten repräsentieren. Allgemein wird ein Knoten beim QGM-Modell als *Box* bezeichnet. Jeder Nichtblattknoten erzeugt eine relationale Tabelle, nachdem über seine Eingangsdaten operiert wurde. Bei den Eingangsdaten handelt es sich wiederum um eine Menge von relationalen Tabellen. Der Wurzelknoten erzeugt das endgültige Anfragenergebnis.

Eine QGM-Box ist definiert durch ihren Typ und die Eingabeattribute, Ausgabeattribute und Prädikate. Es gibt zwei Boxtypen, *SELECT* und *GROUP-BY*. Die Prädikate und Ausgabeattribute werden aus Ausdrücken berechnet, die aus den Eingabeattributen gebildet werden. Die Eingabeattribute einer Box sind die Spalten, die von der Box verarbeitet werden. Ihre Werte werden von den Kindern der Box produziert, die mit den Kanten verbunden sind. Die Verarbeitung der Eingabespalten zu Ausdrücken erfolgt mit Hilfe von Funktionen, Operatoren und Konstanten. Die Ausdrücke spezifizieren die Berechnungen für die Ausgabespalten und Prädikate der Box.

Prädikate sind sowohl in *GROUP-BY*, als auch in *SELECT*-Boxen enthalten.

SELECT-Prädikate sind einfache Selektionsprädikate, Verbundprädikate oder Selektionsprädikate mit Subanfragen. Eine SELECT-Box kann daher mehrere Kinder haben, Verbundoperanden oder Subanfragen.

GROUP-BY-Prädikate beschreiben die Gruppierung der Relation und können als Eingabe nur ein Kind haben, da Gruppierungen immer nur auf einer einzigen Relation angewendet werden können.

Die Ausgabeattribute werden von der Box selbst produziert. Die Ausdrücke für SELECT-Boxen können beliebig komplex sein, solange sie keine Aggregatfunktionen enthalten. Für GROUP-BY-Boxen gilt das Gegenteil: ihre Ausgabeattribute enthalten alle gruppierten Eingabeattribute und Aggregatfunktionen über einfache Eingabeattribute.

Auf SELECT-Boxen werden Selektion, Projektion und Verbund abgebildet, außerdem berechnen sie alle skalaren Ausdrücke, die in SELECT- und GROUP-BY-Klauseln auftauchen, sowie die WHERE- und HAVING-Prädikate.

GROUP-BY-Boxen sind für die Gruppierung und Berechnung der Aggregatfunktionen zuständig. Allgemein gilt, dass die Ausgabeattribute einer Box von beliebig vielen Elternboxen genutzt werden können. Daher gilt eine 1:N Beziehung zwischen Ausgabeattribute und Eingabeattribute.

2.3.2 Zerlegung von Anfragen

Folgende Anfrage:

```
SELECT      faid, state, year(date) as year, count(*) as cnt
FROM        Trans, Loc
WHERE       flid = lid
            and country = 'USA'
GROUP BY   faid, state, year(date)
HAVING      count(*) > 10
```

Abbildung 2.6: Anfrage Q1 - Selektion von länderbezogenen Transaktionsdaten

Die Bedeutung der Tabellennamen und Attribute ist für die Demonstration der Zerlegung nicht wichtig. Das Beispiel kann unter [19] nachgelesen werden.

Die Anfrage aus Abbildung 2.6 wird so zerlegt, dass die Teilkonstrukte auf SELECT- und GROUP-BY-Boxen abgebildet werden können. Darstellung 2.7 zeigt die Abbildung der Anfrage *Q1* auf den QGM-Graph.

Die unterste SELECT-Box entspricht dem Verbund der Trans- und Loc-Tabelle. Der untere Teil der SELECT-Box beinhaltet die Prädikate der Selektion, `flid=lid` und `country='USA'`.

Die Ausgabeattribute, also das Schema, das nach oben zum Elternknoten weitergereicht wird, entspricht der Projektion der Anfrage *Q1* in dem Maße, dass es aus den Spalten der Trans- und Loc-Tabelle gebildet werden kann.

Die Attribute werden an das Elternelement, in diesem Fall die darüberliegende GROUP-BY-Box, weitergeleitet, die als Prädikate die Attribute `faid`, `state` und `year` hat, nach denen in der Anfrage gruppiert werden soll.

Als Ausgabeattribute ist bei der GROUP-BY-Box ein Attribut, *count(*) as cnt*,

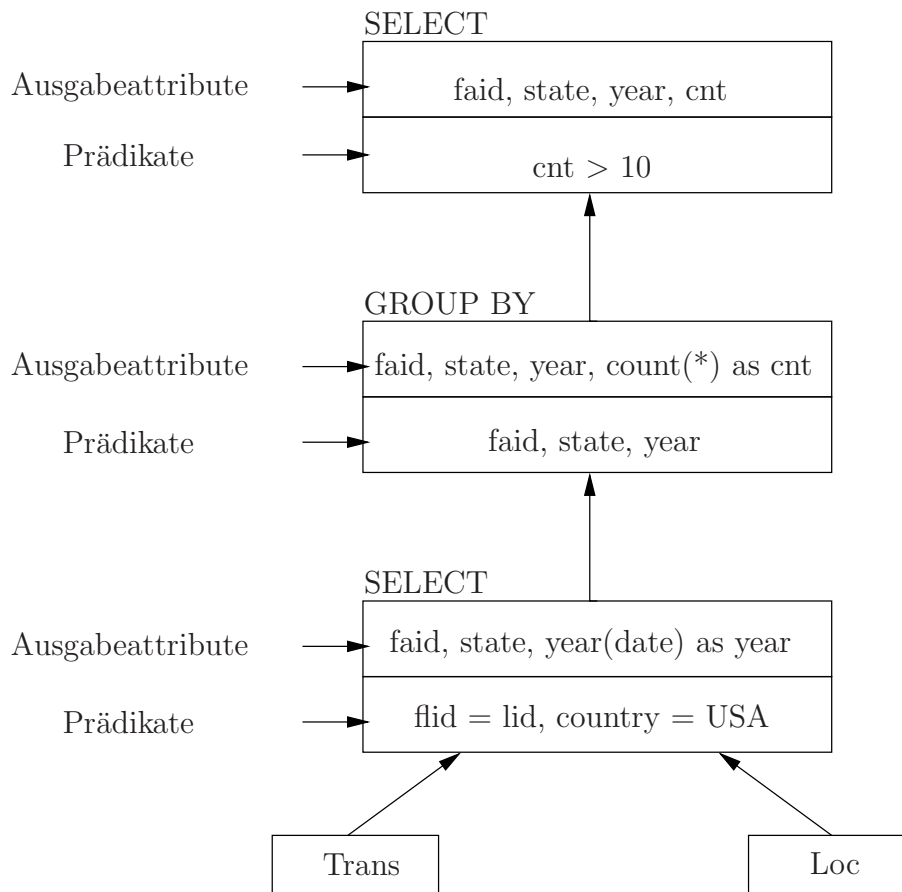


Abbildung 2.7: Anfrage Q1 als QGM-Graph

dazugekommen, da es in der darunterliegenden SELECT-Box aufgrund der noch ausstehenden Gruppierung nicht möglich war, es als Ausgabeattribut zu deklarieren.

Letztendlich wird die Menge von Tupeln semantisch an die Wurzelselectbox weitergeleitet. Die übernimmt schließlich die Selektion der Prädikate der HAVING-Klausel, in dem Fall $cnt > 10$. Die Ausgabeattribute entsprechen denen der Anfrage Q1. Damit ist die Anfrage Q1 komplett auf den QGM-Graph abgebildet.

2.3.3 Matching

Das Ziel der Optimierung ist die Ersetzung von Teilen der Anfrage Q1 durch einen QGM-Graph einer materialisierten Sicht (AST). Da es in den meisten Fällen nicht möglich ist einen 1:1-Match zwischen Teilen der Anfrage Q1 und der AST zu finden, muss eine so genannte Kompensation der Definition der AST stattfinden.

Abstrakt versteht man unter Kompensation einen Adapter, der nötig ist, um einen Match zwischen den Boxen aus Anfrage Q1 und AST zu schaffen. Dabei wird versucht, die Boxen der AST so durch SELECT- und GROUP-BY-Boxen zu erweitern, dass die daraus resultierende Anfrage eine Teilanfrage von Q1 semantisch korrekt ersetzt.

Der Matchingalgorithmus basiert auf dem paarweisen Vergleich von QGM-Boxen. Allgemein matcht eine Box E mit einer anderen Box R, wenn - und nur wenn - ein QGM-Graph $G(E,R)$ konstruiert werden kann, der einen Subgraph $G(R)$ mit Wurzel

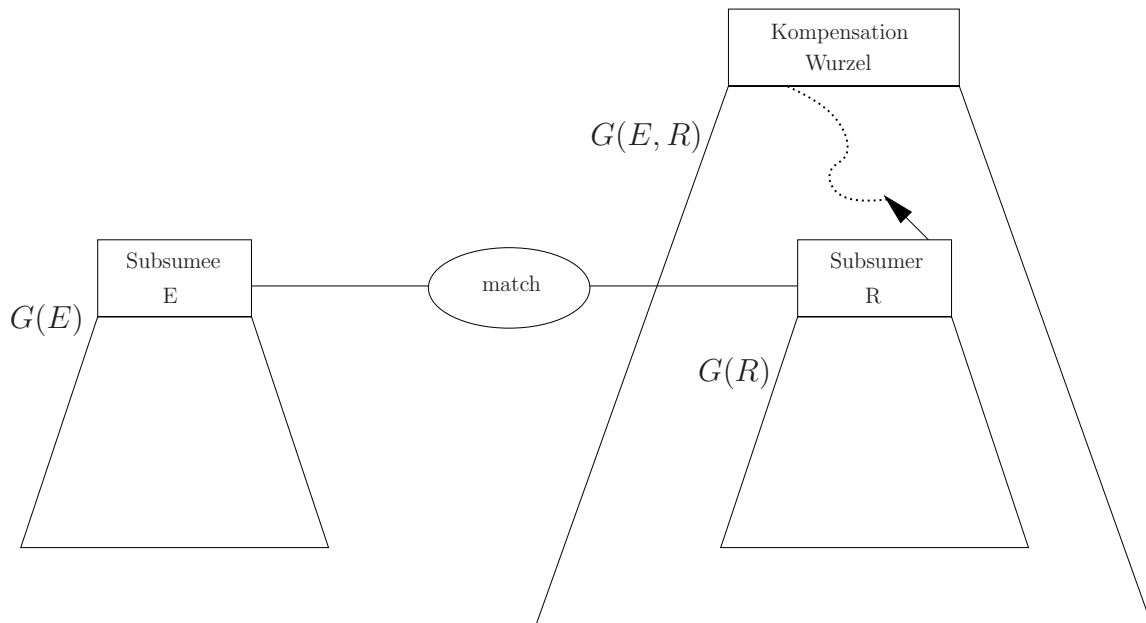


Abbildung 2.8: Die Matching-Beziehung zwischen Subsumee und Subsumer

R hat, und $G(E, R)$ semantisch äquivalent zum Subgraph $G(E)$ mit Wurzel E ist. Mit anderen Worten $G(E, R)$ und $G(E)$ produzieren immer das gleiche Ergebnis.

Wenn Box E mit Box R matcht, dann ist die Differenz von $G(E, R)$ und $G(R)$ die Kompensation. Also die Menge von Operationen, die auf die Ausgabe von R geleistet werden muss, dass diese die gleiche wie die Ausgabe von E ist. Abbildung 2.8 gibt einen grafischen Überblick über diesen Sachverhalt.

Ist die Kompensation leer, so ist der Match exakt und die Boxen E und R sind äquivalent. Wenn nicht, so ist Box E äquivalent zur Wurzelbox der Kompensation.

2.3.4 Matchfunktion

Um die Beziehung zwischen dem paarweisen Vergleich der Boxen und der Kompensation besser ausdrücken zu können, wird Box E als *Subsumee* und Box R als *Subsumer* bezeichnet. Die Matchfunktion hat als Inputparameter zwei QGM-Boxen und bestimmt, ob diese matchen. Im idealen Fall würde die Matchfunktion den allgemeinen Matchingalgorithmus oben implementieren. Leider ist dieser für spezielle SQL-Konstrukte zu allgemein. Es wird daher versucht, die Matchfunktion an die allgemeine Definition soweit wie möglich zu approximieren. Das geschieht durch einfache aber allgemeine Graphmuster, die sowohl aus den gegebenen Subsumee und Subsumer, als auch aus den Kompensationboxen bestehen. Für jedes Muster testet die Matchfunktion eine Anzahl von geeigneten Bedingungen, um zu bestimmen, ob ein Match möglich ist. Es existieren zwei allgemeine Bedingungen für alle Muster:

- Mindestens ein Subsumee-Kind muss mit einem Subsumer-Kind matchen.
- Subsumee und Subsumer müssen vom gleichen Typ sein.

Die erste Bedingung stellt sicher, dass eine minimale Überlappung der Boxen existiert. Die zweite Bedingung dient als Schnelltest, ob der Typ übereinstimmt.

2.3.5 Navigator

Der Navigator scannt die QGM-Graphen der Anfrage und AST von unten nach oben und versucht, mögliche Paare von Matchingboxen zu identifizieren. Dazu wendet er die Matchingfunktion solange auf die Boxenpaare an, bis die Wurzelbox der AST (wenn möglich) erreicht ist. Da alle Möglichkeiten getestet werden müssen, kombiniert der Navigator alle Blattknoten der Graphen miteinander und bildet so eine Menge von Boxenpaaren, wobei die Blattknoten nicht vom selben Graph stammen.

Während jeder Iteration entfernt der Navigator ein Paar aus der Menge und leitet es zur Matchfunktion. Wenn ein Match zustande kommt, werden alle Paare der Elternknoten von Subsumee und Subsumer gebildet. Diese werden dann nacheinander zur Matchfunktion geleitet. Dabei wird vorher geprüft, ob bereits alle Kindknoten gematcht wurden. Aus dieser Bedingung folgt, dass die Matchfunktion bereits die Kompensation aus dem Match zwischen den Kindknoten zu diesem Zeitpunkt kennt. Als Ergebnis muss die Matchfunktion nicht die gesamten Subgraphen der Inputboxen betrachten und kann sich somit auf Subsumee, Subsumer und die Kompensation der Kindboxen konzentrieren, wobei Subsumee vom QGM-Graph der Anfrage und Subsumer vom QGM-Graph der AST stammt.

2.3.6 Muster des Query Graph Models

In den folgenden Abschnitten werden die Muster der Matchfunktion vorgestellt. Dabei besteht jedes Muster aus Match-Bedingungen und der Kompensation. Zu jedem Muster wird nach der Definition ein Beispiel gegeben.

Die folgenden Muster bestehen aus SELECT und/oder einfachen GROUP-BY-Boxen, d. h. GROUP-BY-Boxen ohne *Supergroup*-Funktionen¹. Im Folgenden wird gezeigt, dass es die Möglichkeit gibt, dass ein Subsumee-Kind mit keinem der Subsumer-Kinder matcht. So ein Subsumee-Kind wird auch als *Rejoin-Kind* bezeichnet. Außerdem ist es möglich, dass ein Subsumer-Kind mit keinem passenden Subsumee-Kind matcht. So ein Kind wird als *Extra-Kind* bezeichnet. Ein Verbund zwischen einem Extra-Kind und dem Rest des Subsumers wird als *Extra-Join* bezeichnet.

2.3.7 Exakter Kind-Match

SELECT-Boxen mit Eins-zu-Eins Kind-Match

Muster

Subsumee und Subsumer sind SELECT-Boxen, wobei

- (a) jedes Subsumee-Kind matcht mit höchstens einem Subsumer-Kind und
- (b) zwei Subsumee-Kinder matchen nicht mit dem selben Subsumer-Kind.

¹Als Supergroup-Funktionen werden Klauseln bezeichnet, die von der GROUP-BY-Klausel abgeleitet wurden. Zum Beispiel *ROLLUP* und *CUBE*.

Match-Bedingungen

1. Jeder Extra-Join ist verlustfrei, d.h. es wird kein Tupel vom Subsumer dupliziert oder gelöscht.
2. Jedes Subsumer-Prädikat, das kein Extra-Join-Prädikat ist, ist semantisch äquivalent (matcht) mit einem Subsumee-Prädikat.
3. Jedes Subsumee-Prädikat matcht mit einem Subsumer-Prädikat oder ist ableitbar von den Attributen des Subsumers und/oder eines Rejoin-Kindes.
4. Alle Subsumee-Attribute sind ableitbar von den Subsumer-Attributen und/oder den Rejoin-Kindern. (Ein Subsumee-Ausdruck ist ableitbar, wenn er als Funktion des Subsumer und/oder der Rejoin-Attribute geschrieben werden kann.)

Kompensation

Die Kompensation besteht aus dem Rejoin-Kind (wenn existent) und einer SELECT-Box, die:

- (a) den Subsumer mit den Rejoin-Kindern neu verbindet,
- (b) alle Subsumee-Prädikate behandelt, die nicht mit Subsumer-Prädikate matchen und
- (c) alle Subsumee-Attribute von den Subsumer- und/oder den Rejoin-Attribute ableitet.

Beispiel

Abbildung 2.12 zeigt den Match zwischen einer Anfrage Q2 (Abb. 2.9) und einer materialisierten Sicht AST2 (Abb. 2.10).

```

SELECT      aid, status, qty, price * (1-disc) as amt
FROM        Trans, PGroup, Acct
WHERE       pgid = fpgid
            and faid = aid
            and price > 100
            and disc > 0.1
            and pgroup = TV

```

Abbildung 2.9: Anfrage Q2 - eigentliche Anfrage

```

SELECT      tid, faid, fpgid, status, country,
            price, qty, disc, qty * price as value
FROM        Trans, Loc, Acct
WHERE       lid = flid
            and faid = aid
            and disc > 0.1

```

Abbildung 2.10: Anfrage AST2 - materialisierte Sicht

Der QGM-Graph der beiden Anfragen besteht aus einer SELECT-Box, die drei Basistabellen verbindet (join). Die beiden SELECT-Boxen erfüllen die gerade genannten Bedingungen. Daraus folgt, dass sie mit der Kompensation matchen. Die Kompensation besteht aus einer SELECT-Box (Sel1-1C1) und der PGroup-Tabelle. Abb. 2.11 zeigt

```

SELECT      faid as aid, status, value * (1-disc) as amt
FROM        AST2, PGroup
WHERE       pgid = fpgid
            and price > 100
            and pname = TV
  
```

Abbildung 2.11: Anfrage NeuQ2 - optimierte Anfrage

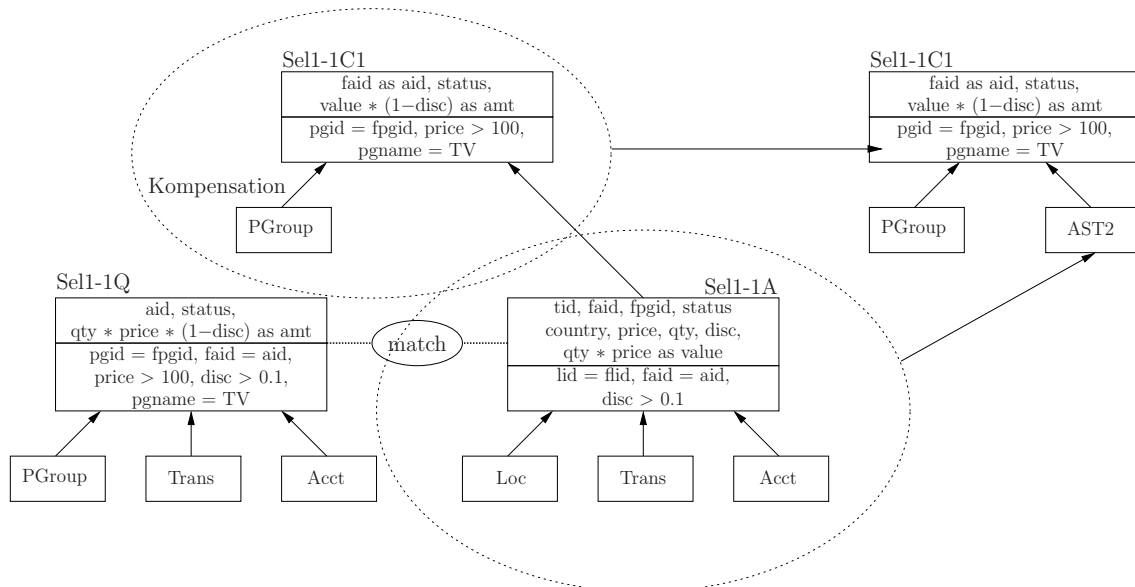


Abbildung 2.12: SELECT-Box mit exaktem Kind-Match

die resultierende neue Anfrage *NeuQ2*, die nur noch aus zwei Basistabellen besteht. Tabelle *PGroup* ist das Rejoin-Kind und *Loc* ist das Extra-Kind. Die Bedingung 1 wird erfüllt, da durch die Einschränkung von $flid = lid$ der Verbund zwischen *Trans* und *Loc* verlustfrei bleibt. Da die Prädikate $faid = aid$ und $disc > 0.1$ des Subsumers ebenso im Subsumee vorkommen, ist auch Bedingung 2 erfüllt. Die relevanten Prädikate $fpgid = pgid$, $price > 100$, $pname = TV$ für Bedingung 3 sind allesamt ableitbar und somit Teil der Kompensation. Bedingung 4 fordert die Ableitbarkeit der Attribute. Die Kompensation leitet das Attribut *aid* von Anfrage Q2 vom Attribut *faid* der AST2 ab. Aufgrund des Prädikats $faid = aid$ ist sichergestellt, dass die Ergebnisse äquivalent sind. Beim Attribut *amt* gibt es zwei Möglichkeiten es abzuleiten, durch Benutzung der Attribute *qty*, *price*, *disc* oder *disc*, *value*. Wenn der Algorithmus mehrere Möglichkeiten zum Ableiten eines Attributes entdeckt, wählt er diejenige mit den am wenigsten benutzten Operanden aus.

GROUP-BY-Boxen

Muster

Subsumee und Subsumer sind einfache GROUP-BY-Boxen deren Kinder exakt matchen.

Match-Bedingungen

1. Jede Subsumee-Gruppierungsspalte ist semantisch äquivalent (matcht) mit einer Subsumer-Gruppierungsspalte.
2. Wenn die Subsumee- und Subsumer-Gruppierungsmenge exakt matchen, d. h. jede Subsumee-Gruppierungsspalte matcht mit einer Subsumer-Gruppierungsspalte und umgekehrt, dann matcht auch die Prädikatmenge von Subsumee und Subsumer, andernfalls ist jede Prädikatmenge vom Subsumee ableitbar von der des Subsumers.

Kompensation

Es wird keine Kompensation benötigt, wenn die Prädikatmenge von Subsumee und Subsumer exakt matchen. Wenn kein Exakt-Match vorliegt, besteht die Kompensation aus einer GROUP-BY-Box, die anhand der Gruppierungsspalten des Subsumees neu gruppiert und die Ausgabeattribute des Subsumees von denen des Subsumers ableitet.

Für Aggregatfunktionen existieren spezielle Ableitungsregeln, die für die Basisfunktionen im Folgenden aufgeführt sind. Alle anderen Funktionen können durch Verknüpfung der Basisfunktionen abgeleitet werden.

Es existiert eine Menge von Spalten $\{x, y, z\}$, wobei

- $x \in \text{Subsumee-Spalten}$ und
- $y, z \in \text{Subsumer-Spalten}$
- z ist vom Typ Nichtnull, und x und y sind semantisch äquivalent

Mit den Regeln:

- (a) COUNT(*) wird nach SUM(cnt) abgeleitet, wobei cnt das COUNT(*) oder COUNT(z) Subsumer-Attribut ist.
- (b) COUNT(x) wird nach SUM(cnt) abgeleitet, wobei cnt die COUNT(y) Subsumer-Spalte ist. Wenn x vom Typ Nichtnull ist, dann kann cnt auch COUNT(z) Subsumer-Attribut sein.
- (c) SUM(x) wird nach SUM(sm) abgeleitet, wobei sm das SUM(y) Subsumer-Attribut ist. Wenn y eine Gruppierungsspalte ist, dann kann SUM(x) auch nach SUM(y*cnt) abgeleitet werden, wobei cnt das COUNT(*) Subsumer-Attribut ist. In diesem Fall enthält die Kompensation eine zusätzliche SELECT-Box, die den Ausdruck 'y*cnt' vor der Umgruppierung berechnet.

- (d) $\text{MAX}(x)$ wird nach $\text{MAX}(\text{max})$ oder $\text{MAX}(y)$ abgeleitet. In der ersten Ableitung ist max die $\text{MAX}(y)$ Subsumer-Spalte. In der zweiten Ableitung muss y eine Gruppierungsspalte sein.
- (e) $\text{MIN}(x)$ gleicht $\text{MAX}(x)$.
- (f) $\text{COUNT}(\text{distinct } x)$ wird nach $\text{COUNT}(y)$ abgeleitet, wenn y eine Gruppierungsspalte ist.
- (g) $\text{SUM}(\text{distinct } x)$ wird nach $\text{SUM}(y)$ abgeleitet, wenn y eine Gruppierungsspalte ist.

Beispiel

Abbildung 2.16 zeigt ein Match zwischen den beiden SELECT-Boxen Sel-1Q und Sel-1A nach den Bedingungen aus Sektion 2.3.7. Die Anfrage und Sichtdefinition sind in den Abbildungen 2.13 und 2.14 dargestellt. Obwohl Box Sel-1A mehr Spalten produziert als Box Sel-1Q, ist der Match exakt, da die Anzahl der Tupel gleich ist. Wäre Sel-1A die Wurzelbox, so wäre eine Kompensation nötig, um die Extraspalten zu eliminieren. Da hier der SELECT-Box noch eine GROUP-BY-Box vorangestellt ist, wird die Extraspalte durch die Projektion der Gruppierungsbox eliminiert.

```
SELECT    year(date) as year, sum(qty * price) as value
FROM      Trans
GROUP BY  year(date)
```

Abbildung 2.13: Anfrage Q4 - eigentliche Anfrage

```
SELECT    year(date) as year, month(date) as month, sum(qty * price) as value
FROM      Trans
GROUP BY  year(date), month(date)
```

Abbildung 2.14: Anfrage AST4 - materialisierte Sicht

```
SELECT    year, sum(value) as value
FROM      AST4
GROUP BY  year
```

Abbildung 2.15: Anfrage NeuQ4 - optimierte Anfrage

Die GROUP-BY-Boxen GB-2Q und GB-2A matchen nach den gerade genannten Bedingungen. Da die Gruppierung von GB-2A noch die Spalte *month* enthält, besteht die Kompensation aus einer GROUP-BY-Box, die die Extragruppierung löscht und mit Box GB-2Q matcht. Zusätzlich leitet die Kompensation die jährliche Summe durch Summierung der monatlichen Summen, unter Benutzung der Regel (c), ab. Diese Ableitung ist korrekt, da die monatlichen Summen Teilsummen der jährlichen Summe sind. Die Anfrage nach der Optimierung ist in Abbildung 2.15 dargestellt.

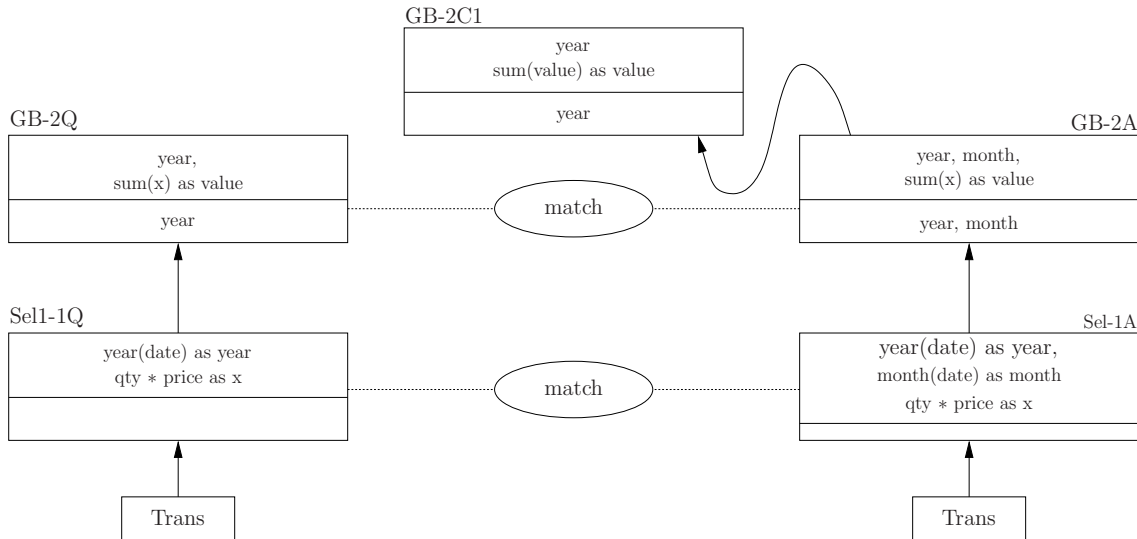


Abbildung 2.16: GROUP-BY-Box mit exaktem Kind-Match

2.3.8 Nicht exakter Kind-Match

Wenn Subsumee und Subsumer nicht exakt matchen, muss die Abweichung kompensiert werden. Dazu muss die Kompensation der Elternknoten die der Kindknoten mit berücksichtigen. Das Weitergeben der Kompensation zum Elternteil wird als *Pulling up* bezeichnet.

GROUP-BY-Boxen mit SELECT-Only-Kindkompensation

Muster

Subsumee und Subsumer sind GROUP-BY Boxen, dessen Kinder mit der Kompensation matchen. Die Kompensation besteht aus einer einzelnen SELECT-Box, die auch einen Verbund realisieren kann. Es wird angenommen, dass wenn $AGG(x)$ eine Subsumee-Aggregatfunktion ist, dass dann die Spalte x nicht von einer Verbundtabelle stammt ([19]).

Match-Bedingungen

1. Jede Subsumee-Gruppierungsspalte ist ableitbar von einer Subsumer-Gruppierungsspalte oder, wenn vorhanden, einer Verbundspalte.
2. Wenn keine Umgruppierungskompensation benötigt wird, dann matcht jede Subsumee-Gruppierungsspalte mit einer Subsumer-Gruppierungsspalte. Ansonsten ist jede Subsumee-Gruppierungsspalte ableitbar von einer Subsumerspalte.
3. Pull-up-Bedingung: Jedes Prädikat der Kompensation der Kindknoten ist ableitbar von den Subsumer-Gruppierungsspalten bzw. von einer Verbundtabellenspalte.

Kompensation

Die Kompensation enthält die Pulled-Up-SELECT-Box, eventuell gefolgt von einer darüberliegenden GROUP-BY-Box. Wenn die Kompensation der Kinder keinen Verbund beinhaltet, dann ist die Regel zum Einfügen einer GROUP-BY-Box die gleiche, wie in Abschnitt 2.3.7. Andernfalls kann eine Neugruppierung nur verhindert werden, wenn die beiden Gruppierungsmengen die gleichen sind und der Rejoin das Verhältnis 1:N hat, wobei die Rejointabelle die linke Seite repräsentiert. Sollte eine Neugruppierung nötig sein, werden die Aggregatfunktionen nach den Bedingungen von Sektion 2.3.7 abgeleitet.

Beispiel (ohne Rejoin)

Abbildung 2.20 zeigt den Match zweier Anfragen Q6 (Abb. 2.17) und AST6 (Abb. 2.18) mit Kompensation. SELECT-Box Sel-1A matcht mit Kompensation Sel-1C1 mit SELECT-Box Sel-1Q. Die Kompensation wird nach oben weitergereicht (pulled up) und sorgt dafür, dass auch GROUP-BY-Box GB-2Q und GB-2A miteinander matchen, sofern die genannten Match-Bedingungen erfüllt werden. Die GROUP-BY-Boxen GB-2Q und GB-2A erfüllen die Bedingungen, die als Ergebnis die Kompensation, bestehend aus SELECT-Box Sel-2C1 und GB-2C2, hervorbringt.

Die SELECT-Box Sel-2C1 ist die weitergereichte Version von Box Sel-1C1, wobei nur das Prädikat sich nicht verändert hat. Bei der Projektion der GROUP-BY-Box GB-2A wurde das Attribut x der darunter liegenden SELECT-Box Sel-1A durch die Aggregatfunktion SUM(x) auf die Spalte value abgebildet. Diese Änderung muss auch in der weitergereichten Kompensation berücksichtigt werden, wodurch die Spalte von x nach value umbenannt wird.

Durch das Prädikat 'month >= 6' der Kompensation ist sichergestellt, dass genau die gleichen Reihen auf beiden Seiten herausselektiert werden. Aus Bedingung 1 folgt, dass jede Subsumergruppierung eine Teilgruppierung von genau einer Subsumee-gruppierung ist. Als Ergebnis produziert die Umgruppierung der GROUP-BY-Box GB-2C2 das korrekte Ergebnis. Abbildung 2.19 zeigt die neue Anfrage *NeuQ6*.

```
SELECT    year(date) % 100 as year, sum(qty * price) as value
FROM      Trans
WHERE     month(date) >= 6
GROUP BY  year(date) % 100
```

Abbildung 2.17: Anfrage Q6 - eigentliche Anfrage

```
SELECT    year(date) as year, month(date) as month, sum(qty * price) as value
FROM      Trans
GROUP BY  year(date), month(date)
```

Abbildung 2.18: Anfrage AST6 - materialisierte Sicht

```

SELECT    year % 100 as year, sum(value) as value
FROM      AST6
WHERE     month >= 6
GROUP BY  year % 100

```

Abbildung 2.19: Anfrage NeuQ6 - optimierte Anfrage

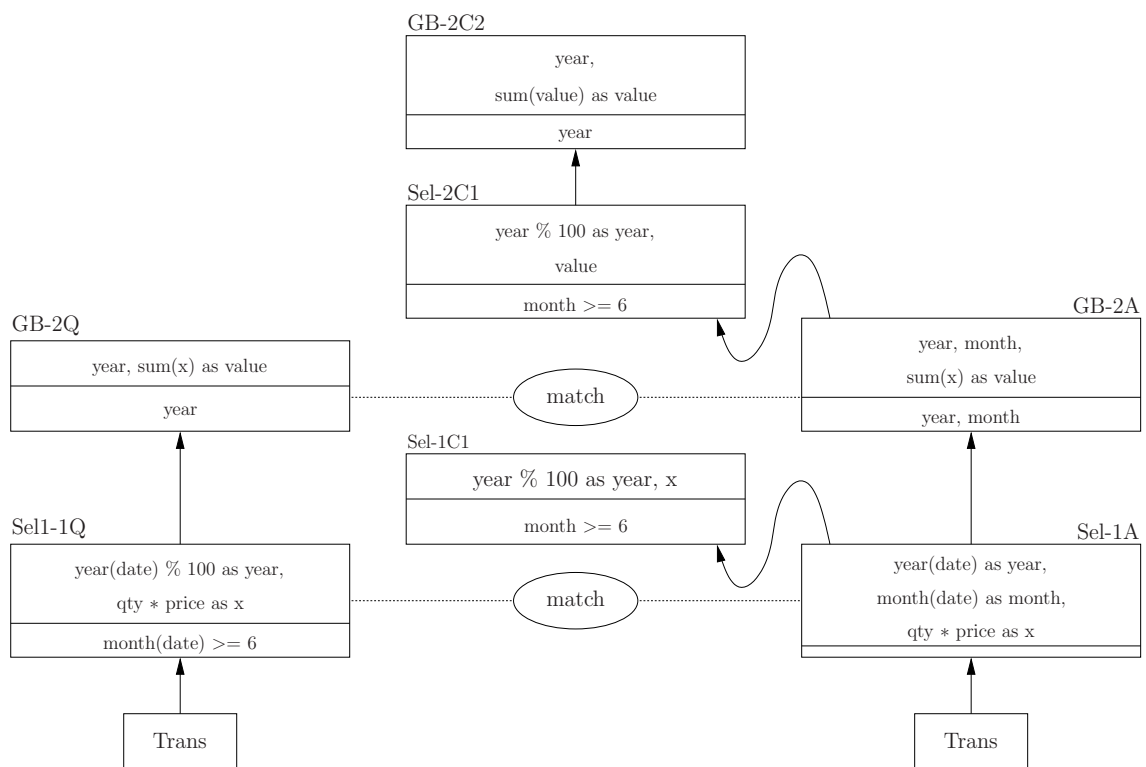


Abbildung 2.20: GROUP-BY-Boxen mit einfacher SELECT-Box-Kompensation

Beispiel (mit Rejoin)

Abbildung 2.24 zeigt den Match zwischen Anfrage Q7 (Abb. 2.21) und einer automatischen Abfragetabelle AST7 (Abb. 2.22). Da die AST7 nur von der Trans-Tabelle abstammt und Anfrage Q7 aus einem Join zwischen den Relationen Trans und Loc besteht, ist es nötig einen Rejoin von Anfrage AST7 durchzuführen. Wenn man davon ausgeht, dass der Verbund von Trans und Loc ein N:M Join ist, muss eine neue Gruppierungsbox zur Kompensation hinzugefügt werden, die die Duplikate löscht, die beim nachträglichen Verbinden entstehen. Vergleicht man Box Sel-1Q mit Box Sel-2C1, so unterscheiden sie sich nur in der Multiplizität der Reihen, da für jede Reihe von Loc ein Duplikat einer Reihe der AST7, die die Selektionsbedingung erfüllt, entsteht. Die Eliminierung solcher Tupel wird mit anschließender Gruppierung durch Box GB-2C2 erreicht. Unter realen Umständen wäre die Beziehung zwischen Loc und Trans 1:N. Somit könnte auf Box GB-2C2 verzichtet werden, da keine Duplikate entstehen können. Falls die Beziehung allgemein N:M ist, wird die Box jedoch benötigt, um erstens die Duplikate zu entfernen und zweitens die cnt-Spalte korrekt zu berechnen. Dies geschieht mit der Aggregatfunktion SUM, die die Werte der cnt-Spalte der eliminierten Duplikate summiert. Damit wäre die neue Anfrage NeuQ7 äquivalent zur Anfrage Q7.

```
SELECT      lid, year(date) as year, count(*) as cnt
FROM        Trans, Loc
WHERE       flid = lid and country = USA
GROUP BY   lid, year(date)
```

Abbildung 2.21: Anfrage Q7 - eigentliche Anfrage

```
SELECT      flid, year(date) as year, count(*) as cnt
FROM        Trans
GROUP BY   flid, year(date)
```

Abbildung 2.22: Anfrage AST7 - materialisierte Sicht

```
SELECT      lid, year, sum(cnt) as cnt
FROM        AST7, Loc
WHERE       flid = lid and country = USA
GROUP BY   lid, year
```

Abbildung 2.23: Anfrage NeuQ7 - optimierte Anfrage

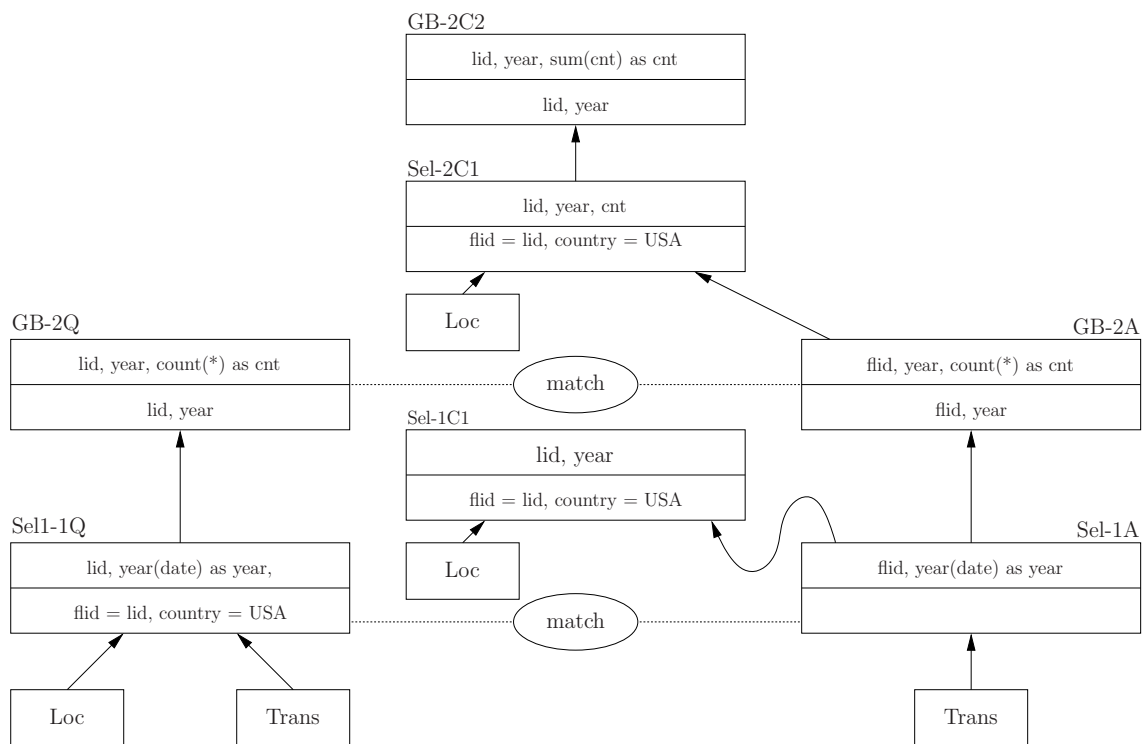


Abbildung 2.24: GROUP-BY-Boxen mit Rejoin-Kompensation

GROUP-BY-Boxen mit GROUP-BY-Kindkompensation

Muster

Abbildung 2.25 zeigt die allgemeine Form dieses Musters. Subsumee und Subsumer sind GROUP-BY-Boxen (GB-Q und GB-A). Die Kompensation der Kinder enthält eine Anzahl von SELECT-Boxen, möglicherweise null und mindestens eine GROUP-BY-Box. GB-cC2 ist die niedrigste GROUP-BY-Box der Kindkompensation.

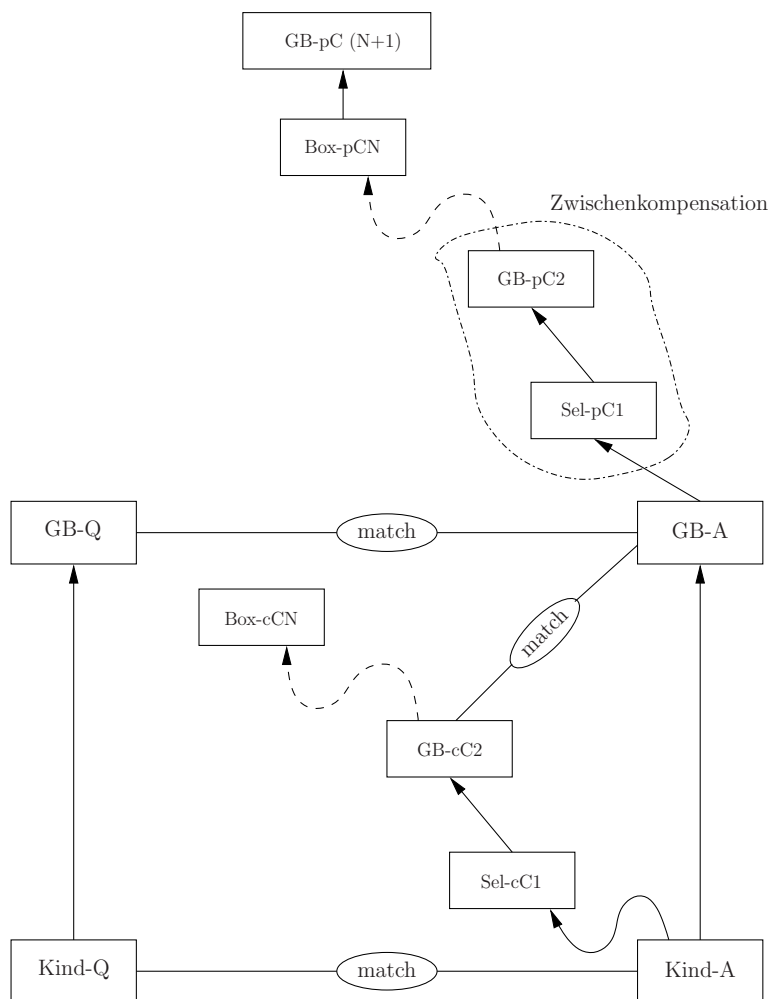


Abbildung 2.25: GROUP-BY-Boxen mit GROUP-BY-Kind-Kompensation (allg. Form)

Match-Bedingung

Die Match-Funktion versucht, die GROUP-BY-Boxen der Kindkompensation nacheinander mit GROUP-BY-Box GB-A zu matchen. Dabei werden die Boxen, beginnend mit der Niedrigsten (GB-cC2), rekursiv aufgerufen, wobei Box GB-cC2 als Subsumee, GROUP-BY-Box GB-A als Subsumer und Sel-cC1 (falls vorhanden) als Kindkompensation verstanden wird. Wenn dieser Zwischenmatch erfolgreich ist, dann ist auch der Originalmatch zwischen GB-Q und GB-A erfolgreich. Dieser rekursive Aufruf der Matchfunktion ist konform zu den Mustern aus Abschnitt 2.3.7 und 2.3.8.

Kompensation

Als Konsequenz des Matches von GB-cC2 und GB-A beginnt die Kompensation mit der Zwischenkompensation von GB-cC2 und GB-A. Danach werden alle Kindkompensationsboxen, die sich GB-cC2 anschliessen, zur Kompensation hinzugefügt. Zum Beispiel ist Box-pCN eine exakte Kopie von Box-cCN. Danach wird Subsumee GB-Q ans Ende der Kompensation kopiert.

Um zu überprüfen, dass diese Aktion korrekt ist, betrachtet man den Match zwischen GB-cC2 und GB-A. Danach sind GB-cC2 und GB-pC2 laut Matchdefinition äquivalent. Daraus folgt, dass alle Boxen (Box-cCN), die über GB-cC2 in der Kindkompensation liegen, äquivalent zu ihren Kopien (Box-pCN) in der Elternkompensation sind. Da Box-cCN aufgrund des Matches zwischen Kind-Q und Kind-A äquivalent zu Kind-Q ist, impliziert dies, dass Kind-Q auch äquivalent zu Box-pCN ist. Als Ergebnis ist GB-Q ebenso äquivalent zu GB-pC(N+1), da sie Kopien voneinander und ihre Kinder äquivalent sind.

Beispiel

Abbildung 2.29 zeigt den Ablauf des QGM-Graph-Matches der Anfrage Q8 (Abb. 2.26) und AST8 (Abb. 2.27). Beide sind Histogrammanfragen, wobei Q8 nach der Anzahl der Transaktionen und der Menge der Jahre, die diese Anzahl an Transaktionen hatten, gruppiert. AST8 selektiert die gleichen Tupel, gruppiert jedoch zusätzlich nach Anzahl der monatlichen Transaktionen.

Box GB-2C1 ist die Kompensation des Matches zwischen den inneren zwei GROUP-BY-Boxen. Box GB-3C2 und Sel-3C1 ist die Kompensation für den Match zwischen GB-2C1 und GB-3A, wobei die Bedingungen und Regeln aus Abschnitt 2.3.7 gelten. Den Abschluss der Kompensation bildet Box GB-3C3, die eine Kopie von GB-3Q ist. Die resultierende Anfrage NeuQ8 ist in Abbildung 2.28 zu sehen.

```
SELECT      tcnt, count(*) as ycnt
FROM        (
             SELECT year(date) as year, count(*) as tcnt
             FROM Trans
             GROUP BY year(date)
            )
GROUP BY tcnt
```

Abbildung 2.26: Anfrage Q8 - eigentliche Anfrage

```
SELECT      year, tcnt, count(*) as mcnt
FROM        (
             SELECT year(date) as year, month(date) as month, count(*) as tcnt
             FROM Trans
             GROUP BY year(date), month(date)
            )
GROUP BY year, tcnt
```

Abbildung 2.27: Anfrage AST8 - materialisierte Sicht

```
SELECT      tcnt, count(*) as ycnt
FROM        (
            SELECT year, sum(tcnt * mcnt) as tcnt
            FROM AST8
            GROUP BY year
            )
GROUP BY tcnt
```

Abbildung 2.28: Anfrage NeuQ8 - optimierte Anfrage

SELECT-Boxen mit SELECT-Only-Kindkompensation

Muster

Subsumee und Subsumer sind SELECT-Boxen und ihre Kinder matchen mit Kompensation, die keine Gruppierung enthält.

Match-Bedingungen

Die Bedingungen ähneln denen von Abschnitt 2.3.7, jedoch müssen bezüglich der Kindkompensation einige Anpassungen gemacht werden.

1. siehe Abschnitt 2.3.7
2. Jedes Subsumerprädikat, das kein Extra-Join-Prädikat ist, matcht mit einem Subsumee- oder Kindkompensationsprädikat.
3. siehe Abschnitt 2.3.7
4. siehe Abschnitt 2.3.7
5. Pull-up-Bedingung: Jedes Kindkompensationsprädikat, das kein passendes Subsumerprädikat hat, ist ableitbar von den Subsumerattributen und/oder falls vorhanden, von den Rejoinattributen.

Kompensation

Die Kompensation enthält die Rejoin-Kinder (falls vorhanden) und eine einzelne SELECT-Box, die alle Subsumee- und/oder Kindkompensationsprädikate enthält, die keine passenden Subsumerprädikate haben.

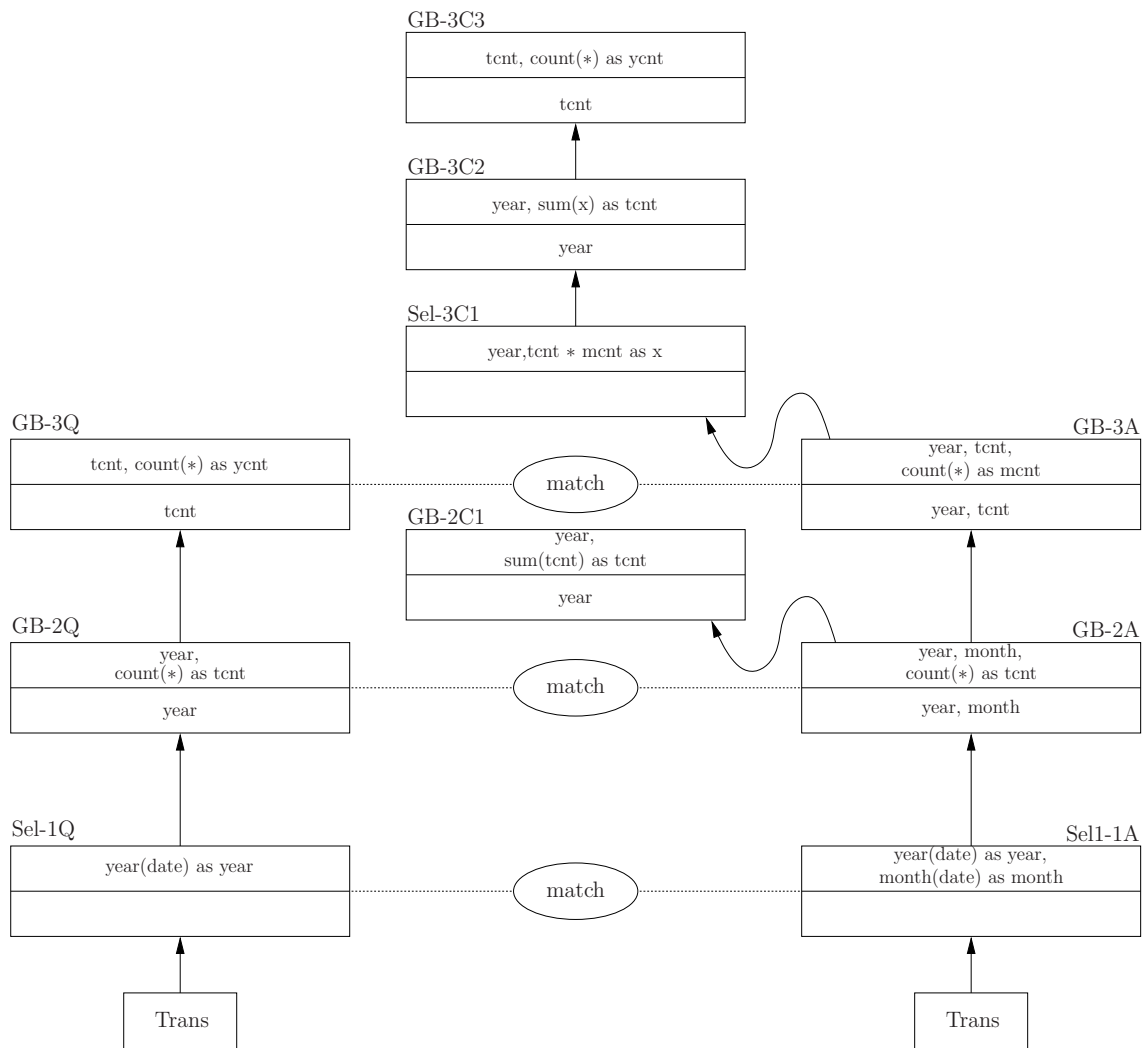


Abbildung 2.29: GROUP-BY-Boxen mit GROUP-BY-Kind-Kompensation

SELECT-Boxen mit GROUP-BY-Kindkompensation ohne Verbund

Muster

Subsumee und Subsumer sind SELECT-Boxen ohne überlappenden Verbund und mit höchstens einem Kindmatch, bei dem die Kompensation eine Gruppierung enthält.

Match-Bedingungen

Die Match-Bedingungen sind die gleichen wie in Abschnitt 2.3.8. Zusätzlich gilt eine Pull-up-Bedingung für GROUP-BY-Boxen der Kindkompensation. Dabei ist jedes Prädikat (Gruppierungsspalte) einer GROUP-BY-Box jeder Kindkompensation ableitbar von einem Subsumer- oder Rejoinattribut.

Kompensation

Die Kompensation besteht aus drei Schritten. Zuerst wird jedes Nichtgruppierungschild, wie in Abschnitt 2.3.8 beschrieben, nach der Pull-up-Bedingung behandelt. Dadurch wird eine SELECT-Box in der Elternkompensation erzeugt. Danach wird die Gruppierungschildkompensation an die Spitze der SELECT-Box gesetzt (Pull up). Hinzu kommt eine weitere SELECT-Box, die die Prädikate des Subsumees kompensiert.

2.4 Zusammenfassung

WATCHMAN, ein Datawarehouse-Cachemanager, ist ein Verfahren zum dynamischen Verwalten von materialisierten Sichten. Es werden zwei Algorithmen genutzt, die zu einem Dritten kombiniert werden. Der Cache-Ersetzungsalgorithmus (LNC-R) basiert auf der Bewertung jeder einzelnen Anfrage Q_i aufgrund von Referenzrate, sich ergebene Relationsgröße und Ausführungskosten. Das Hauptziel von Cache Admission (LNC-A) ist die Speicherung von Anfragen zu vermeiden, die die Antwortzeit des Systems insgesamt verschlechtern. LNC-RA nutzt LNC-R und LNC-A kombiniert und berücksichtigt somit die Vorteile von beiden Verfahren. Leider nutzt das Verfahren bei der Materialisierung nur komplette Anfragen. Das heißt es werden keine Teilmengen von Anfragen materialisiert, von denen andere Anfragen profitieren könnten. Die Erzeugung von Zugriffspfaden auf ASTs wird ebenso nicht berücksichtigt.

Der Anfrageplan ist eine algebraische Struktur, der als Baum dargestellt wird. Die Knoten entsprechen Operationen und Relationen. Dabei werden Tabellen als Blattknoten und die Operationen als innere Knoten abgebildet. Bei der physischen Optimierung werden anhand des Anfrageplans mehrere Zugriffspläne generiert, die die Speicherform der Relationen miteinbeziehen. Die Zugriffspläne werden bewertet und einer davon zur Ausführung ausgewählt.

Das Query Graph Model wird für eine Optimierungsmethode verwendet, bei der versucht wird, Teile einer Anfrage durch eine materialisierte Abfragetabelle (AST) zu ersetzen. Zum Beispiel könnte eine Anfrage Q , die aus einem Verbund von drei Basistabellen besteht, durch eine AST, die aus einem Verbund von zwei Basistabellen besteht, die auch in Q enthalten sind, teilweise ersetzt werden. So würde der Verbund von Q nicht

mehr aus den drei Basistabellen bestehen, sondern nur noch aus einer Basistabelle und der AST. In diesem Fall würde eine Verbundoperation eingespart werden. Man könnte sich sicherlich noch aufwendigere Operation vorstellen, deren Berechnungsaufwand bei entsprechend genutzter AST noch geringer ausfallen würde.

Abstrakt ist das QGM ein Baum mit einem Wurzelement und Blattknoten. Eine Anfrage wird so zerlegt, dass Aggregatfunktionen wie SUM, CNT, MAX, MIN usw. auf einen Typ GROUP-BY und Verbünde, Selektion und Projektion auf einen Typ SELECT abgebildet werden. Die Teile der Anfrage werden dann auf Boxen aufgeteilt, wobei jede Box einen Typ hat. Die Blattknoten entsprechen den relationalen Tabellen, die ihre Daten an die Elternbox weitergeben. Durch Anordnung der Boxen im Baum kann eine Anfrage auf dem Graph abgebildet werden. So wird jeweils ein Graph einer Anfrage und AST gebildet und miteinander verglichen. Dieses findet mittels eines Navigators statt, der Boxenpaare bildet und miteinander über eine Matchfunktion vergleicht. Die Matchfunktion kann auf ein Repertoire von Mustern zum Vergleich zugreifen und durch Probieren einen Match finden.

Werden alle Boxen komplett gematcht, entsteht durch die Anwendung der Muster eine neue Anfrage, die jetzt die materialisierte Sicht anstatt des Verbundes von einigen Basistabellen benutzt und so zu einer wesentlich kürzeren Anfragebearbeitung führt.

Dank der modularen Aufteilung des Query Graph Models in Navigator, Matchfunktion und zugehörige Muster, ist es auch in Zukunft möglich, Muster zu definieren, die einem neuen SQL-Standard genügen. Der Navigator bildet Boxenpaare und leitet diese zur Matchfunktion, die darauf die vorhandenen Muster anwendet.

Der Aufwand für die Optimierung kann vernachlässigt werden, da dieser für das sukzessive Testen der Muster linear und für das Bilden der Boxenpaare quadratisch ist. Da die Anzahl an möglichen Paaren jedoch als verschwindend gering, relativ zum vorhanden Speicher eines Rechners angesehen werden kann und die nötige Prozesszeit im Vergleich zur Berechnung mehrerer Verbünde als kleiner Anteil verstanden wird, ist die Verwendung eines solchen Optimierungsverfahrens sehr effektiv.

Es gibt zwei grundlegende Mustertypen mit exaktem und nichtexaktem Kindmatch. Bei exaktem Kindmatch besteht die Kompensation nur aus der Ableitung des Subsumees vom Subsumer. Diese kann aus SELECT-Boxen bzw. GROUP-BY-Boxen bestehen. Bei einem nicht exaktem Kindmatch muss zusätzlich die Kompensation der Kindknoten mitberücksichtigt werden. Dabei wird die Kindkompensation auf verschiedene Weise angepasst bzw. muss erneut mit dem Elternelement kompensiert werden. Die kompensierten Boxen werden dann an den Anfang der Elternkompensation kopiert, wo von ihnen weiter abgeleitet wird, so dass ein Match mit dem Subsumee zustande kommt.

Es existieren weitere Muster, die auf Supergruppierungsfunktionen wie ROLLUP, CUBE [6] und Gruppierungsmengen [18] matchen. Diese Muster sollen an dieser Stelle nur erwähnt werden, da deren Aufbau ähnlich der bereits vorgestellten Muster ist. Das Muster kann in [19] nachgelesen werden.

Kapitel 3

Konzept

3.1 Einleitung

Die Verwendung der dynamischen Anfragebearbeitung unter Zuhilfenahme des Query Graph Models macht es immer noch notwendig, die materialisierte Sichtdefinition statisch zu erarbeiten und zu erstellen. Wünschenswert wäre ein Datenbanksystem, das materialisierte Sichten dynamisch bzw. transparent verwaltet und diese Funktionalität damit abstrakt für den Benutzer zur Verfügung stellt. Eine dynamische Verwaltung beinhaltet neben der Anfrageoptimierung durch materialisierte Sichten auch die automatische Erstellung der AST's.

WATCHMAN 2.1 behandelt bereits die automatische Erzeugung und Verwaltung von materialisierten Sichten. Dies jedoch auf einem Level, der nur komplette Anfragen materialisiert. Häufig ist es jedoch gerade sinnvoll, nur Obermengen einer Anfrage als AST zu erzeugen, um eventuell mehrere Anfragen, anstatt nur einer, auf eine AST abzubilden. Außerdem wird die Ermittlung von physischen Zugriffsstrukturen außen vorgelassen. Ein Ansatz zur Realisierung dieses Aspektes soll in diesem Kapitel vorgestellt werden. Verschiedene Kriterien müssen dazu näher betrachtet werden.

Der Workload ist die Anfragemenge, die an das System gestellt wird. Aus ihm kann die Häufigkeit der einzelnen Anfragen abgeleitet werden. Da sich der Workload dynamisch über die Zeit ändern kann, ist auch die „Alterung“ (Aging[14]) der Anfragen von Bedeutung.

Jede Anfrage kann in verschiedene Teiloperationen zerlegt werden, die alle einen unterschiedlichen Berechnungsaufwand haben. Die Zerlegung folgt dem im Grundlagenkapitel (Kapitel 2) beschriebenen Query Graph Model. Dabei werden sowohl Selektion, als auch Gruppierung auf Boxen unterschiedlichen Typs verteilt. Der entstehende Graph repräsentiert die Anfrage und ermöglicht damit ein Matching mit anderen Anfragen.

Die Idee bei der Erstellung von materialisierten Sichten, die den Workload optimal abbilden, ist, einen Graph zu konstruieren, der den Workload komplett repräsentiert. Aus dem Graph sollen anhand der vorhandenen Informationen wie Operationsaufwand, Anzahl der Aufrufe und resultierende Relationsgröße der Anfragen eine möglichst optimale Menge von Relationen als materialisierte Sichten gewonnen werden, so dass eine manuelle Erstellung von ASTs nicht mehr nötig sein sollte.

Ein weiterer Aspekt bei der Verwaltung der AST's ist die Aktualität der Sichtde-

definitionen. Im Zusammenspiel mit dem Aging-Verfahren müssen Entscheidungsfaktoren getroffen werden, die die Sichtdefinition möglichst schnell an einen sich verändernden Workload anpassen, aber zugleich durch kurzzeitige Workload-Drifts kein ständiges Alternieren der Sichtdefinitionen bewirken.

Es müssen Zugriffsmechanismen auf die erstellten ASTs erzeugt werden. Dazu werden Indexe benötigt, die den Zugriff optimieren. Es wird ein Verfahren vorgestellt, Indexe automatisch zu erzeugen und an einen sich ändernden Workload anzupassen.

3.2 Erweiterung der Box-Typen

Im Unterschied zum QGM-Verfahren wird zusätzlich zur Selektion und Gruppierung ein weiterer Box-Typ „Tabelle“ benötigt. Dieser wird beim QGM-Verfahren durch den Namen der Basistabelle repräsentiert. Die Änderung ist nötig, da zusätzlich Eigenschaften zur späteren Profitberechnung gekapselt werden müssen.

Die Eigenschaften zur Profitberechnung werden sowohl in den Attributen einer Box, als auch in der Box selbst gespeichert.

3.2.1 Attributeigenschaften

Jedes Attribut einer Box hält zwei Informationen zur Berechnung der Boxeigenschaften bereit. Die Attributgröße spiegelt den Typ bzw. die maximale Größe des Attributs wieder und wird für die Tupelgrößenberechnung gebraucht.

Als zweite Eigenschaft wird die Kardinalität der Wertemenge bzw. die Anzahl der möglich annehmbaren Werte abgespeichert. Diese wird zur Berechnung der Tupelanzahl benötigt.

Attribut(Größe, Kardinalität)

3.2.2 Allgemeine Eigenschaften einer Box

Jede Box erhält zusätzlich allgemeine Eigenschaften, die zur Aufwandsberechnung benötigt werden. Diese Eigenschaften (Tabelle 3.1) werden aus statistischen Informationen des Systems gewonnen.

Abbildung 3.1 zeigt den strukturellen Aufbau einer Box. Die zusätzlichen Eigenschaften werden für jede Box abhängig von den Prädikaten, Attributen und Werten der Kindboxen berechnet.

Die Tupelanzahl entspricht der Anzahl der Tupel der Box, die bis zu diesem Zeitpunkt entstanden sind. Joins, Selektionen und Gruppierungen erhöhen bzw. verringern die Anzahl. Dabei wird bei der Berechnung die Tupelanzahl der Kindboxen genutzt.

Die Tupelgröße richtet sich nach der Anzahl und der Größe der Boxattribute. Die Berechnung erfolgt mittels Summation der Größe aller Attribute.

Die Kosten für die Berechnung der Box leitet sich von der Tupelanzahl, Anzahl der Kindboxen und vom Typ der Box ab. Gehen wir vom folgenden vereinfachten Modell aus: Es sei n das Produkt der Tupelanzahlwerte der Kindknoten. Für Gruppierungen sei der Aufwand $n * \log(n)$, für Selektion n und für ausschließliche Projektion gleich 0.

Tupelanzahl	→ Anzahl der Tupel der Relation (TpCnt)
Tupelgröße	→ Summe der Größe der einzelnen Attributtypen (TpSz)
Berechnungsaufwand	→ Entspricht der Operationsanzahl bei der Berechnung der Prädikate (OPC)
Referenzanzahl	→ Entspricht einem Integer-Wert, der beim erfolgreichen Match inkrementiert wird (RefCnt)

Tabelle 3.1: Allgemeine Eigenschaften einer Box

Der Referenzzähler startet bei der Erzeugung der Box mit dem Wert 1 und wird bei jedem erfolgreichen Match um 1 inkrementiert. Er ist ein Indikator für die Verwendungshäufigkeit der Box und spielt somit eine große Rolle bei der Profitberechnung.

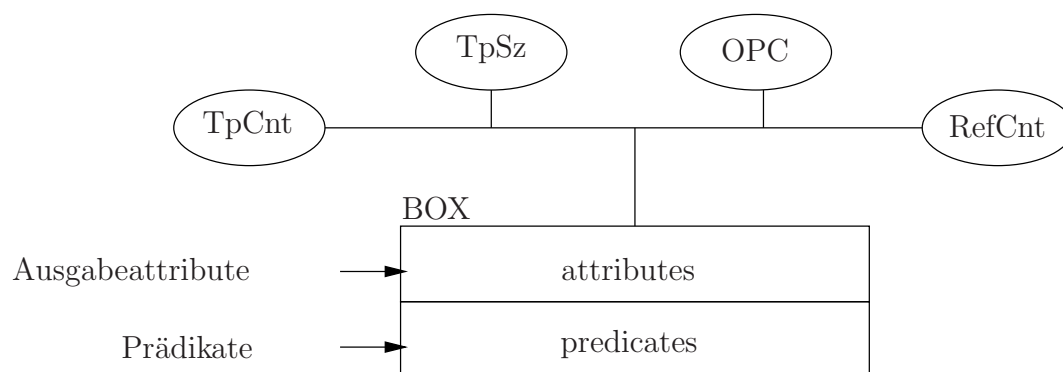


Abbildung 3.1: Eigenschaften einer Box

3.2.3 Tabellen-Box

Eine Tabellenbox, siehe Abbildung 3.2, hat als Attribute die Attribute der Basistabelle einer Anfrage. Als einziges Prädikat erhält sie den Namen der Basistabelle. Somit bleibt eine Tabellenbox ableitbar von den anderen Boxtypen, da die allgemeine Definition einer Box weiterhin genutzt werden kann.

3.3 Zerlegung und Matching

Die Zerlegung der Anfragen folgt weitestgehend der im Grundlagenkapitel 2.3.2 beschriebenen Zerlegung. Dabei wird die SQL-Anfrage anhand ihrer Operationen Selektion und Gruppierung mit Boxen selbigen Typs auf einen QGM-Graph abgebildet. Wie bereits

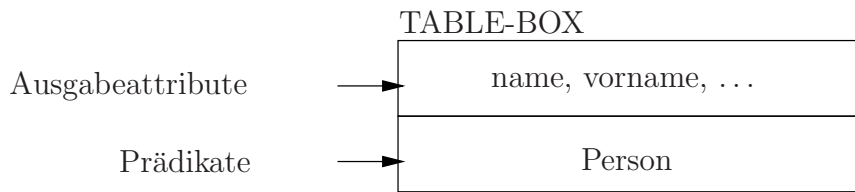


Abbildung 3.2: Darstellung einer Table-Box

beschrieben, wird anstatt der Zeichenkette für den Namen der Basistabelle die Tabellenbox bei der Erzeugung des Anfragegraphs verwendet. Der Grund sind die zusätzlichen Eigenschaften, die gekapselt werden müssen.

Das Matching folgt äquivalent zur Zerlegung dem Matching (Abschnitt 2.3.3) der dynamischen Anfragebeantwortung aus dem Grundlagenkapitel. Jedoch müssen für die Optimierung des Mapgraphs zusätzliche Muster für die Mergingoperation 3.5.3 verglichen werden. Diese werden in Abschnitt 3.6 vorgestellt und beispielhaft erklärt.

3.4 Mapgraph

Der Abbildungsgraph bzw. Mapgraph ist ein ungerichteter, azyklischer Graph und repräsentiert eine Darstellung des Workloads. Die Anfragen müssen in den Graph eingefügt bzw. mit vorhandenen Boxen gematcht werden.

3.4.1 Aufbau

Im Unterschied zur dynamischen Anfragebeantwortung (Abschnitt 2.3) existieren zusätzlich zur Gruppierung und Selektion sogenannte Tabellenboxen. Diese entsprechen den Basistabellen der Anfragen. Von den Basistabellen ausgehend führen beliebig viele Kanten zu unterschiedlichen Boxen, d. h. die von der ausgehenden Box repräsentierte Relation wird von diesen Boxen „konsumiert“.

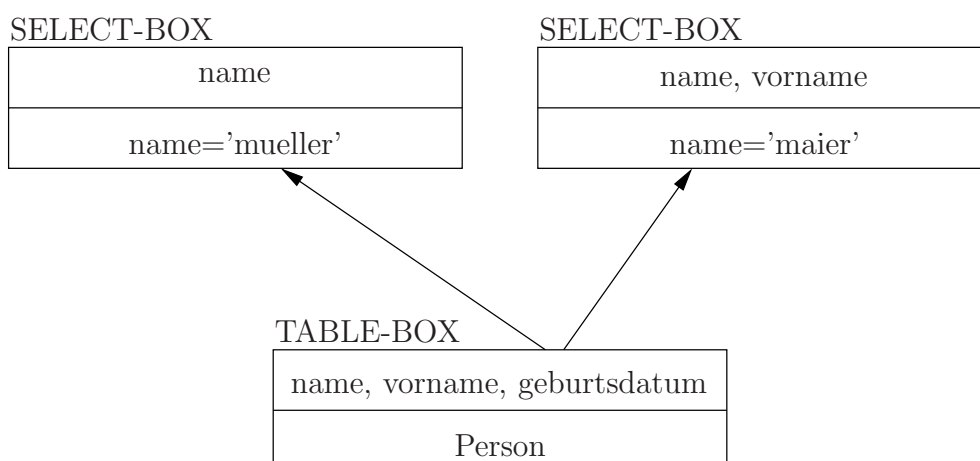


Abbildung 3.3: MapGraph mit einer TabellenBox und zwei konsumierenden Select-Boxen

In Abbildung 3.3 ist ein einfacher Abbildungsgraph mit einer Tabellenbox und zwei Select-Boxen dargestellt. Die beiden Select-Boxen konsumieren die von der Tabellen-

Box ausgehende Relation und wenden ihre Prädikate auf die Attribute der Tupel an. Die Select-Boxen liefern ihrerseits wieder eine Relation, die den Prädikaten genügt. Aufbauend könnten weitere Boxen die Relationen der Boxen konsumieren, um speziellere Anfragen abzubilden.

Der MapGraph besteht aus einer Menge von Basistabellen. Falls der Workload alle Basistabellen abfragt, deckt auch der Graph alle Tabellenboxen ab.

Die Größe des Graphen hängt von der Anzahl und Ähnlichkeit der Anfragen des Workloads ab. Da ähnliche oder gleiche Boxen aufeinander abgebildet werden können, reduziert sich die Anzahl der benötigten Boxen. Umgekehrt gilt das ebenso für einen unähnlichen Anfragemix, der die Anzahl der Boxen im Verhältnis erhöht.

3.4.2 Einfügen einer Box

Falls kein Match einer Anfragebox mit einer Menge von MapGraph-Boxen zustande kommt, muss die Anfragebox an die Kindbox der MapGraph-Box angefügt werden. Dafür bietet der MapGraph eine Einfügeoperation an, die die gegenseitige Referenzierung der Boxen sicherstellt. Es registrieren sich Kindbox und Elternbox gegenseitig. Somit kann der Graph in jede Richtung traversiert werden.

Abbildung 3.4 zeigt den Einfügeablauf einer Anfrage in den MapGraph. Die Graphen von Anfrage und MapGraph werden von unten nach oben durchlaufen und verglichen. Bei beiden Graphen sind die Tabellenboxen identisch, was sich positiv auf das weitere Matchen der Elternboxen äußert. Dazu vergleicht die Matchfunktion jede Select-Box der Anfrage (links) mit jeder Select-Box des Graphen (rechts) in der Ebene.

Die Select-Box *Box-Sel3* der Anfrage matcht mit keiner der beiden Boxen des MapGraphs. *Box-Sel2* enthält nicht die Prädikate des Subsumee und *Box-Sel1* hat eine kleinere Attributmenge als die Anfragebox. Da kein Match mit einer Graphbox weiter existiert, wird die *Box-Sel3* an *Box-Table-MG* angehängt.

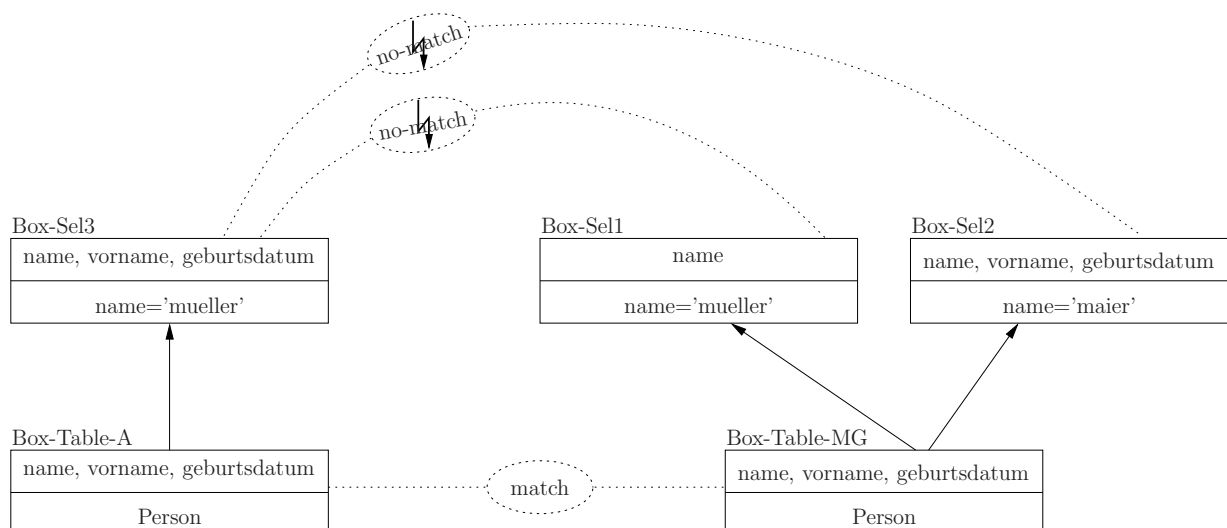


Abbildung 3.4: Einfügen in den Abbildungsgraph

Das Ergebnis der Einfügung ist in Abbildung 3.5 dargestellt. Bei genauerer Betrachtungsweise fällt auf, dass *Box-Sel1* aufgrund des gleichen Prädikats und der kleineren

Attributmengene eine Teilrelation von *Box-Sel3* darstellt. Wäre die Einfügereihenfolge vertauscht, würde der Graph eine andere und günstigere Gestalt haben. *Box-Sel1* wäre dann Elternbox von *Box-Sel3* und würde die Relation von *Box-Sel3* konsumieren. Um diesem

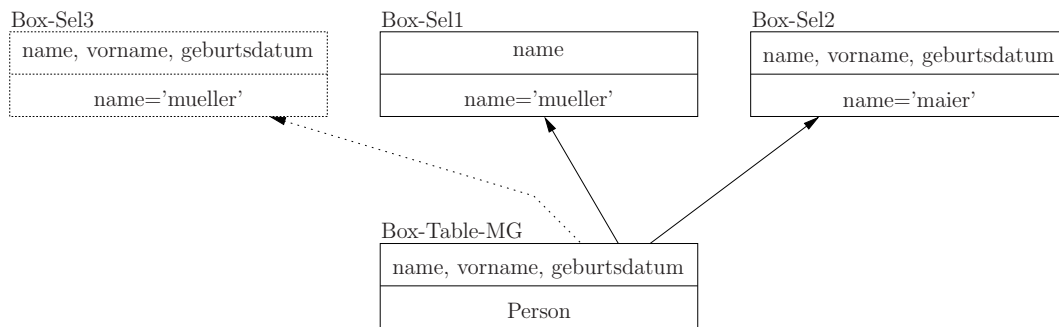


Abbildung 3.5: MapGraph nach Einfügung

Problem zu begegnen, wird nach jeder Einfügeoperation eine Restrukturierung (siehe Abschnitt 3.5.2) durchgeführt, die den Graph in eine verbesserte Anordnung überführt.

Mehrdeutigkeiten

Beim Einfügen der Anfragen in den Mapgraph kann es zu Mehrdeutigkeiten beim Matching kommen. Nämlich genau dann, wenn Prädikate einzeln eingefügt werden, die zu einem späteren Zeitpunkt gemeinsam auftreten. Abbildung 3.6 verdeutlicht dieses Problem. Die Anfrage auf der linken Seite mit den Boxen *Box-A1*, *Box-A2* und *Box-A3* soll in den Mapgraph auf der rechten Seite eingefügt werden. Der Mapgraph besteht vor dem Einfügen der Anfrage aus den Boxen *Box-M1*, *Box-M2*, *Box-M3* und *Box-M4*. Beim Einfügen stellt sich die Frage, welcher Pfad für das Matching verwendet werden soll. Dabei matcht bereits *Box-A1* mit *Box-M1* und *Box-A2* mit *Box-M2* ohne Kompensation. *Box-A3* matcht sowohl mit *Box-M3*, als auch mit *Box-M4*. Die sich daraus ergebenden Kompensationen wären *Box-M5* mit *Box-M2* als Rejoinkind oder *Box-M6* mit *Box-M1* als Rejoinkind. Um solche Mehrdeutigkeiten und die möglicherweise damit verbundene unkorrekte Berechnung der Materialisierung zu vermeiden, ist es nötig, die Prädikate vor dem Matching alphabetisch nach Attributname zu sortieren und dann mit den schon sortierten Boxen im Mapgraph zu matchen. Durch diesen Prozess ist die Eindeutigkeit des Mapgraphs sichergestellt.

3.4.3 Entfernen einer Box

Muss eine Box entfernt werden, werden automatisch auch deren Elternknoten entfernt. Aufgrund des Referenzierverfahrens müssten die Elternboxen im nächsten Schritt sowieso entfernt werden. Diese Operation kann somit eingespart werden.

Eine Entfernung einer Box aus dem Graph entspricht dem Entfernen dieser Box aus den Verknüpfungslisten der Kindboxen. Der Graph wird immer von den Tabellenboxen aus traversiert. Somit kann die abgeschnittene Box nicht mehr erreicht werden. Eine Löschung der Box und ihrer Elternknoten steht dann nichts mehr im Wege.

In Abbildung 3.7 ist ein Graph dargestellt, aus dem die Box *Box-ToDelete* gelöscht werden soll. Die Löschoption geht von der Box, die gelöscht werden soll, aus. Ein

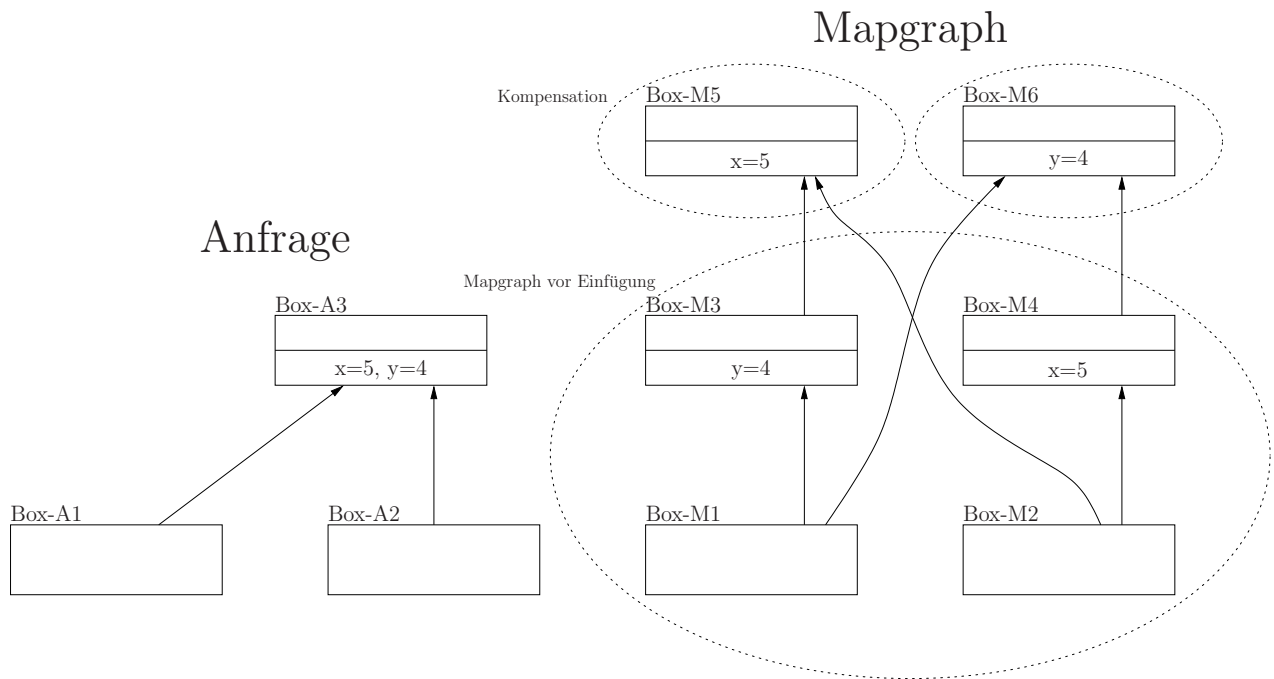


Abbildung 3.6: Mehrdeutigkeit beim Matching

Aufruf der Elternboxlöschoperation der Kindboxen *Box-Child1* und *Box-Child2*, mit der zu löschenden Box als Parameter, bewirkt eine Entfernung der Verlinkung der Kindboxen mit der zu löschenden Box. Der abgeschnittene Teilast wird aufgrund seiner fehlenden Bindung zum MapGraph gelöscht.

Abbildung 3.8 zeigt den MapGraph nach dem Löschen des Teilastes. Dieser besteht jetzt nur noch aus *Box-Child1* und *Box-Child2*.

3.5 Navigator

Der Navigator sorgt für das Matching bzw. Einfügen der Anfragen in den Mapgraph. Dabei müssen möglicherweise Restrukturierung und Vermischung von Graphboxen vorgenommen werden.

3.5.1 Anfragebehandlung

Soll eine Anfrage auf den Mapgraph abgebildet werden, iteriert der Navigator das Matching aufsteigend, beginnend bei den Tabellenboxen. Bei Anfragen, die aus mehreren Basistabellen bestehen und somit eine Verbundtabelle besitzen, wählt der Navigator eine Basistabelle aus und versucht bis zur nächsten Joinbox zu matchen. Dann bricht das Matching an dieser Stelle ab und es folgt die nächste Basistabelle und so weiter. Entsteht beim Matching eine Kompensation, wird diese vor der Behandlung der nächsten Anfragebox mit den Elternboxen der Graphbox verglichen. Das Matching wird dann mit den Elternboxen der durch die Kompensation gematchten oder erzeugten Mapgraphbox weitergeführt.

Kann eine Box nicht gematcht werden, greift der Einfügealgorithmus des Mapgraphs.

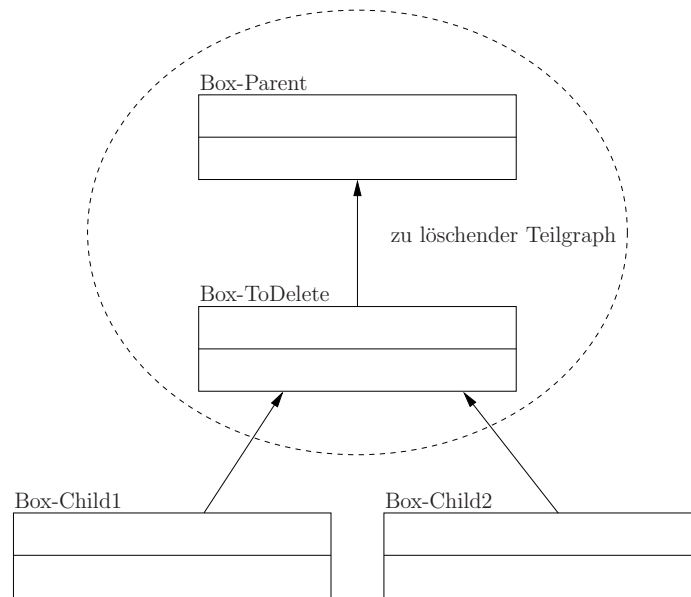


Abbildung 3.7: Löschen einer Box

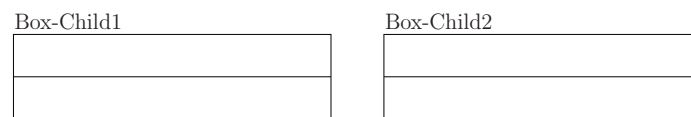


Abbildung 3.8: MapGraph nach dem Entfernen des Teilastes

Dabei wird die Anfragebox ohne Boxverbindungen geklont und an die letzte gematchte Mapgraphbox als Elternbox angefügt. Handelt es sich um eine Tabellenbox, wird diese der Basistabellenkollektion des Mapgraphs hinzugefügt. Der darüberliegende Teilgraph der Anfrage wird ebenso bis zur nächsten Joinbox eingefügt.

Eine Joinbox kann erst gematcht oder eingefügt werden, wenn alle Kindboxen verglichen wurden. Aus diesem Grund wird der Zustand der Einfügung oder des Matching der Kindboxen abgespeichert. Der Zustand beinhaltet eine Anfragebox, die zugehörige Mapgraphbox und die eventuell beim Matching entstandene Kompensation. Wurden alle Kindboxen der Verbundbox positiv gematcht, kann die Verbundbox selbst verglichen werden. Dazu wird die Verbundbox der Anfrage und des Mapgraphs zusammen mit den Kindkompensationen zur Matchfunktion geleitet. Ist der Match positiv, folgt die nächste Elternbox der Anfrage.

Ist der Match der Verbundboxen negativ, wird geprüft, ob die Kindkompensationsboxen der Anfrageverbundbox nach der Pull-up-Bedingung ableitbar sind. Ist dies der Fall, wird die Verbundbox, mit den Kindkompensationsboxen als Elternboxen, in den Mapgraph eingefügt. Ist die Pull-up-Bedingung nicht erfüllt, werden die Kindkompensationsboxen, die die Verbundbox als Elternbox erhalten, an die zugehörigen Kindboxen angehängen. Die Elternboxen der Verbundbox werden bis zur nächsten Verbundbox der Anfrage, falls existent, in den Graph eingefügt. Danach beginnt der Einfügealgorithmus von neuem.

Beispiel

Abbildung 3.9 zeigt schematisch die Einfügesituation einer Anfrage in den Mapgraph. Dabei repräsentiert der linke Graph die Anfrage und der rechte den Mapgraph. Der Navigator beginnt mit der *Table-Box-A1* der Anfrage und versucht diese mit den Tabellenboxen des Mapgraphs zu matchen. Der Match mit *Table-Box-M1* ist positiv. Der Navigator

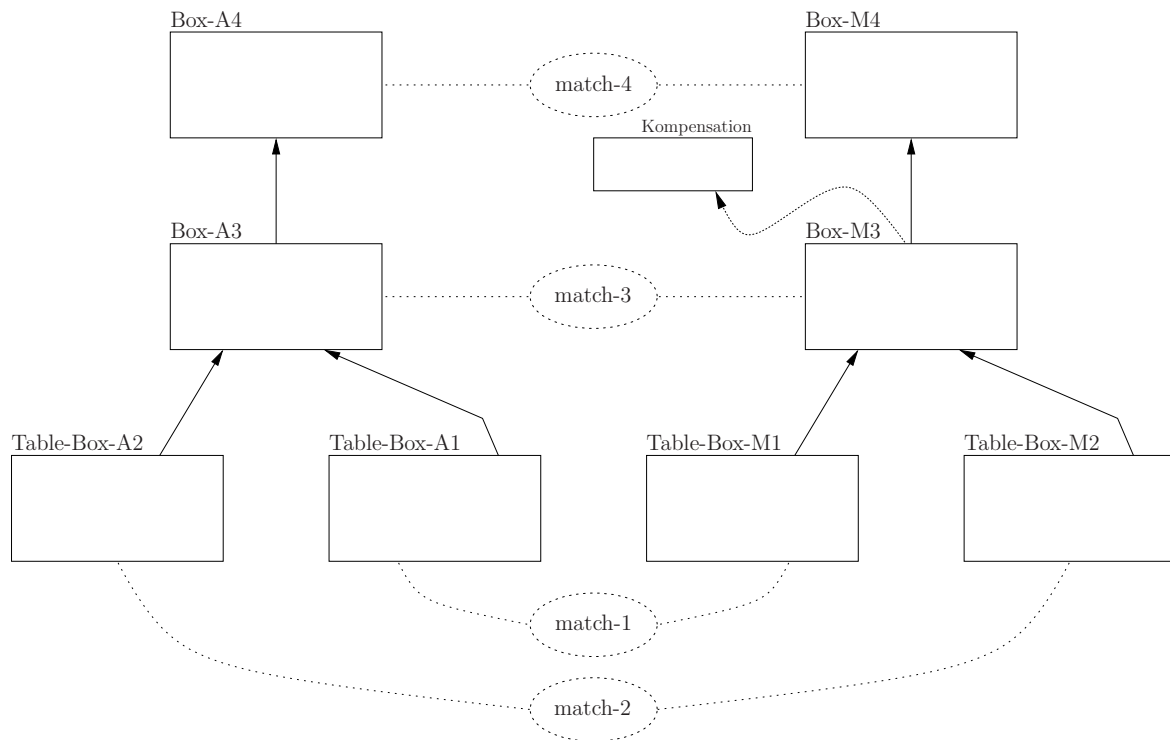


Abbildung 3.9: Einfügen einer Anfrage in den Mapgraph

versucht dann die einzige Elternbox *Box-A3* zu matchen. Da diese eine Verbundbox ist und noch nicht alle Kindboxen gematcht wurden, wird der Zustand des Matchings, bestehend aus Mapgraphbox *Table-Box-M1* und Anfragebox *Table-Box-A1*, gespeichert. Der Navigator führt dann das Matching mit *Table-Box-A2* und den Tabellenboxen aus dem Mapgraph fort. *Table-Box-M2* wird von der Matchfunktion als äquivalent angesehen. Die Elternbox von *Table-Box-A2* ist wieder die Verbundbox *Box-A3*, die nun aufgrund der Tatsache, dass alle Kindboxen gematcht worden sind, selbst zur Matchfunktion geleitet werden kann. Bevor dieses geschieht, müssen noch eventuelle Kompensationen aus den gespeicherten Zuständen gematcht werden. In diesem Beispiel existiert keine Kompensation der Kindboxen, so dass *Box-A3* mit der einzigen Elternbox *Box-M3* der beiden gematchten Mapgraphboxen verglichen werden kann. Die Matchfunktion positiviert den Vergleich, aus dem eine Kompensation hervorgeht. *Box-A4* und *Box-M4* werden ebenfalls von der Matchfunktion zusammen mit der Kindkompensation positiviert. Die Kindkompensation wird als neue Elternbox *Box-M5* an *Box-M4* angefügt. Da keine weiteren Elternboxen in der Anfrage existieren, ist die Einfügeoperation abgeschlossen. Abbildung 3.10 stellt den veränderten Mapgraph nach der Einfügeoperation dar.

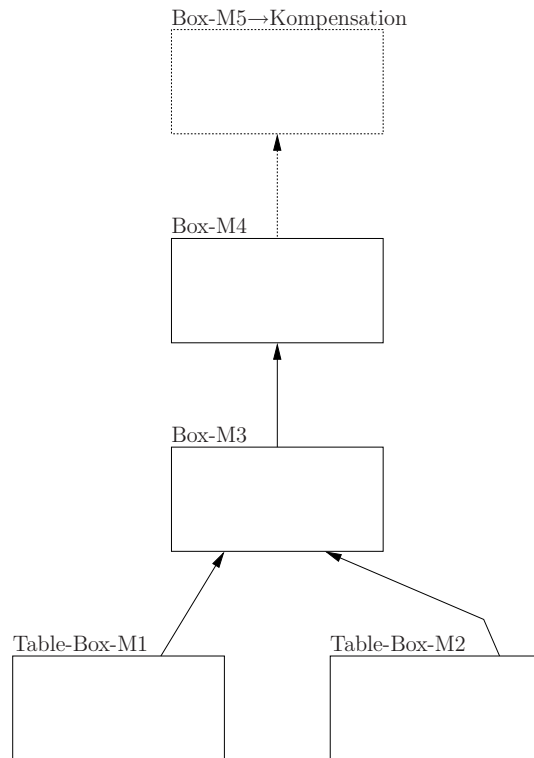


Abbildung 3.10: Mapgraph nach der Einfügung

3.5.2 Restrukturierung

Der Graph sollte immer so gestaltet sein, dass möglichst viele Anfragen einen gemeinsamen Teilgraph nutzen. Da aufgrund der Materialisierungsentscheidung immer eine möglichst „breite“ Sicht der Anfragen als Ergebnis entstehen soll, ist es wichtig, dass nach dem Einfügen einer Box die Struktur geprüft und gegebenenfalls verändert werden muss. Der resultierende Mapgraph sollte nach der Restrukturierung die Teilmengenbedingung erfüllen, d. h. es existieren keine Boxen in der Elternboxebene, die Teilmengen voneinander abbilden.

Eine Umstrukturierung des MapGraphs wird möglicherweise nach jeder Einfügeoperation einer Anfrage nötig. Je nachdem, ob Boxen eingefügt wurden, die eine Obermenge einer verglichenen Box darstellen. Da aufgrund der Obermenge kein Match zustande kommen konnte, wurde die Obermengenbox neben der Teilmengenbox eingefügt. Eine Umstrukturierung wird damit notwendig.

Die Umstrukturierung erfolgt ausgehend von den Tabellenboxen bis zu den „Enden“ des Graphen. Damit wird sichergestellt, dass vereinigte Elternboxen zweier gematchter Boxen auch restrukturiert werden können.

Der Algorithmus der zum Restrukturieren der Boxen im Graph genutzt wird, ist in Auflistung 3.1 dargestellt. Alle Elternboxen werden miteinander in beide Richtungen gematcht. Bei Erfolg wird eine Hilfsfunktion aufgerufen, die die Elternboxen der Teilmengenbox an die gematchte Box oder entstandene Kompensation anhängt. Gegebenfalls werden Rejoinkinder an eine gebildete Kompensation angefügt. Hierfür gelten die Muster (Abschnitt 2.3.6) der dynamischen Anfragebeantwortung aus dem Grundlagenkapitel. Nach der Restrukturierung ist eine Neuberechnung der Operationskosten des

Listing 3.1: Restrukturierungsfunktion

```

function restructureMapGraph (childBox)
  boxPairs = getAllBoxPairs(childBox)
  foreach (pair in boxPairs)
    lBox = pair.left
    rBox = pair.right
    if (MatchFunction.match(lBox, rBox, null))
      compensation = MatchFunction.getCompensation()
      manageBoxRestructuring (lBox, rBox, compensation)
    else
      if (MatchFunction.match(rBox, lBox, null))
        compensation = MatchFunction.getCompensation()
        manageBoxRestructuring (rBox, lBox, compensation)

function manageBoxRestructuring(boxToRemove, mGraphBox, compensation)
  if (compensation == null)
    mGraphBox.addParentBoxes(boxToRemove.getParentBoxes())
  else
    mGraphBox.addParentBox(compensation)
    compensation.addParentBoxes(boxToRemove.getParentBoxes())
  boxToRemove.removeAllParentBoxes()
  boxToRemove.removeAllChildBoxes()

```

möglicherweise entstandenen Subsumee nötig, da Prädikate dieser Box auf den Subsumer abgebildet wurden.

Beispiel

Das Beispiel in Abbildung 3.11 zeigt einen Mapgraph, der ausgehend von *Box-child1* versucht, die Elternboxen *Box-Sr* und *Box-Se* zu restrukturieren. *Box-Sr* ist der Subsumer und *Box-Se* der Subsumee. Da beide Boxen eine Überlappung *Box-child1* und *Box-child2* haben, die Prädikate vom Subsumer als Teilmenge vom Subsumee und die Attribute vom Subsumee als Teilmenge vom Subsumer angenommen werden, matchen die beiden Boxen. Dabei ist der Extrajoin mit *Box-ExJ* nach der Musterbedingung verlustfrei.

Es entsteht eine Kompensation, die die Prädikate des Subsumee ohne Subsumerprädikate, das Rejoinkind *Box-ReJ* und die Subsumeeattribute enthält (Abbildung 3.12). Der Restrukturierungsalgorithmus fügt danach die Elternboxen, repräsentiert vom Teilgraph *Graph-Parent-Se*, an die gebildete Kompensation an. Abschließend wird der Subsumee aus dem Mapgraph herausgelöst, indem alle Kindboxen (*Box-child1*, *Box-child2*, *Box-ReJ*) und alle Elternboxen *Graph-Parent-Se* ihre Verlinkung zum Subsumee trennen.

Sollte keine Kompensation beim Match der Boxen entstehen, wird anstatt der Kompensation der Subsumer als Kindbox für die Elternboxen des Subsumees genutzt. Allerdings ist das nur der Fall, falls Subsumee und Subsumer identisch und die Kindboxen des Subsumees eine Teilmenge der Kindboxen des Subsumers sind.

Diese Restrukturierung wird rekursiv von den Tabellenboxen aus auf alle Boxen angewendet. Bei erfolgreicher Verschmelzung der Boxen wird der Referenzzähler des Subsumers um die Referenzanzahl des Subsumees erweitert. Somit erhält diese Box einen größeren Einfluss bei der Profitberechnung (Abschnitt 3.8).

Die Komplexität für das Durchlaufen des Graphen ist linear, da jede Box nur einmal besucht wird. Der Aufwand für das Permutieren und Matching der Elternboxen einer Box ist quadratisch. Da die Permutation immer nur auf die Elternboxen jeder Box angewendet

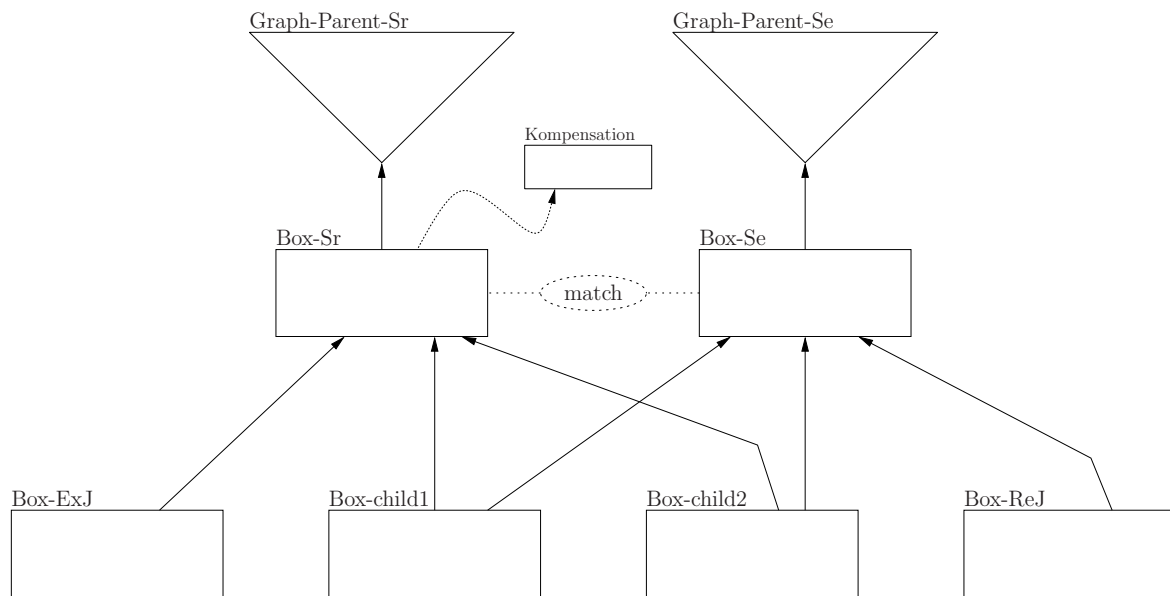


Abbildung 3.11: Matchsituation beim Restrukturieren

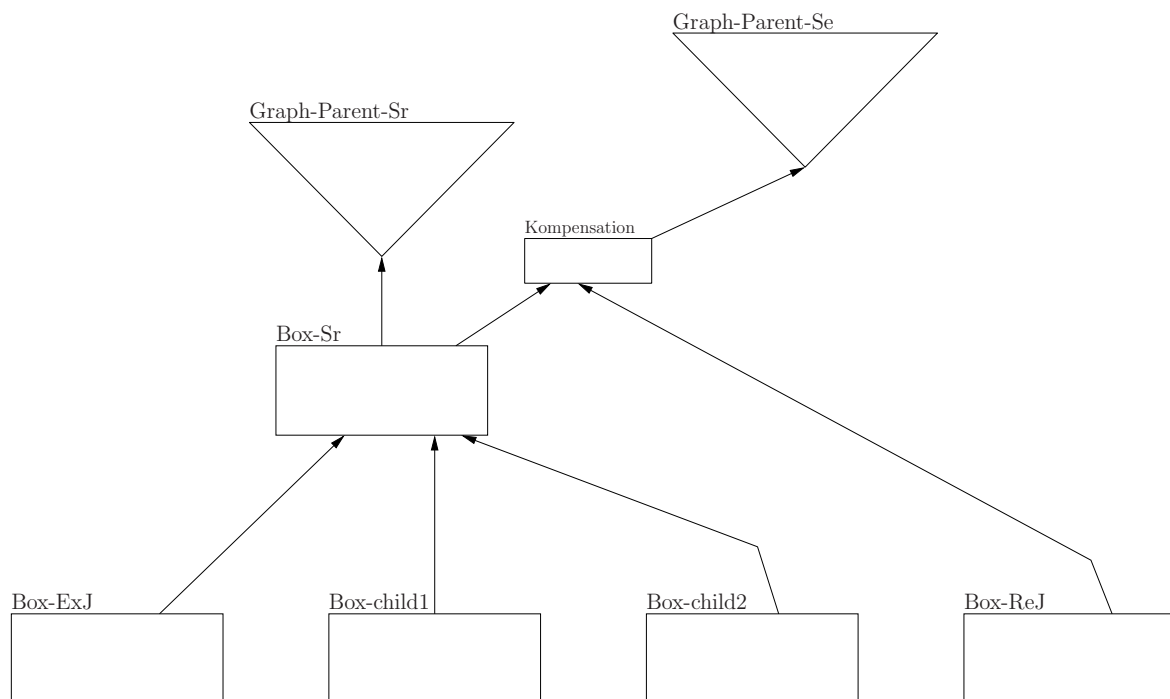


Abbildung 3.12: Matchsituation nach dem Restrukturieren

wird, kann dieser Aufwand vernachlässigt werden.

Insgesamt werden durch die Restrukturierung Anfragen zusammengefasst, die damit auf eine gemeinsame Box zugreifen können. Im besten Fall werden Boxen ohne Kompensation gematcht, so dass sogar Boxen eingespart werden können. Die Kompensation ersetzt den Subsumee und wird eine Ebene weiter darüber angeordnet. Dadurch wurden mehr Anfragen auf eine Teilanfrage abgebildet, wodurch die Wahrscheinlichkeit einer Materialisierung der Teilanfrage steigt.

3.5.3 Merging

Die dynamische Anfragebeantwortung nutzt Muster, um Boxen vollständig durch ein Äquivalent zu ersetzen. Dabei wird die ersetzte Box durch eine Kompensation einer ähnlichen Box repräsentiert. Die Kompensation liefert das gleiche Ergebnis, wie die ersetzte Box.

Bei der Abbildung der Anfragen ist dieses Ziel nicht immer erwünscht. Boxen sollen nicht unbedingt nur ersetzt, sondern es sollen auch gleiche Bestandteile einer Box gemeinsam genutzt werden. Dazu ist es nötig, die Gemeinsamkeiten herauszufiltern und in einer neuen Box zu kapseln. Das Merging wird nach der Restrukturierung auf den gesamten Mapgraph angewendet. Erst danach steht der Mapgraph für eine neue Einfügeoperation zur Verfügung.

Der Mapgraph wird, wie bei der Restrukturierung, rekursiv ausgehend von den Tabellenboxen durchlaufen. Aus den Elternboxen einer Mapgraphbox werden Paare permutiert, die sukzessiv zur Matchfunktion geleitet werden. Findet ein Merging zwischen zwei Boxen statt, wird das Durchlaufen des Mapgraphs an dieser Stelle unterbrochen und eine Restrukturierung ab dieser Box vorgenommen. Diese Maßnahme ist nötig, da die anderen Elternboxen Teilmengen der Mergebox sein könnten. Ist die Restrukturierung des Teilastes beendet, wird der Durchlauf des Mapgraphs mit einer möglicherweise durch die Restrukturierung veränderten Elternboxmenge fortgesetzt. Der Berechnungsaufwand ist der Gleiche wie bei der Restrukturierung (Abschnitt 3.5.2). Die Muster, die diese Vermischung der Boxen übernehmen, werden im Musterkapitel (Abschnitt 3.6) vorgestellt.

Weiterhin denkbar wäre das Merging von Boxen, bei dem Prädikate verwendet werden, die nicht einer gemeinsamen Prädikatschnittmenge angehören. Dies würde eine Vereinigung von Relationen bedeuten, die zur Anfragezeit wieder aufgelöst werden müsste. Dieser Mehraufwand sollte sich in Bezug zur Einsparung von Speicherplatz für das System lohnen. Dazu wären Verfahren nötig, die Mergeboxen aus unterschiedlichen Prädikatmengen mit Herkunft aus den zu mergenden Boxen erstellen und bewerten. Eine Mergebox wird nur erstellt, falls der Profit der Mergebox größer als der zu mergenden Boxen ist. Dabei muss der Mehraufwand für die Berechnung der ursprünglichen Relationen der zu mergenden Boxen vom Profit der Mergebox abgezogen werden. Dies ist nötig, um den Mehraufwand der Berechnung der Anfragen zur Laufzeit gerecht zu werden. Der Mapgraph sollte von dieser Vereinigung nicht betroffen sein, da sonst ein weiterer Boxtyp zur Repräsentation der Vereinigung erstellt werden müsste. Bei Änderung des Workloads wären dann weitere Reduzierungsverfahren nötig, die die Vereinigungsboxen wieder teilen. Daher sollten erst bei der Profitberechnung diese Mergeboxen unabhängig vom Mapgraph erstellt und im Zwischenspeicher verwaltet werden. Die Mergebox in der

Kandidatenliste wäre damit ein Repräsentant der gemergten Boxen, der nicht im Mapgraph existiert. Dieses Verfahren wäre allerdings sehr aufwendig und ein Nutzen müsste erst untersucht werden. Dies soll hier nur am Rande diskutiert werden und für zukünftige Arbeiten einen Aspekt darstellen.

3.6 Muster der Matchfunktion

Die im Grundlagenkapitel verwendeten Muster (Abschnitt 2.3.6) können für das Matching der Boxen vollständig übernommen werden. Des Weiteren müssen für die Mergingoperation zusätzliche Muster bereitgestellt werden. Diese werden in den folgenden Abschnitten behandelt.

Dazu werden zwei Boxen anhand von Bedingungen miteinander verglichen. Matcht der Vergleich positiv, so wird eine Mergebox erzeugt, die Gemeinsamkeiten der Boxen abbildet. Die verglichenen Boxen werden als Subsumees bezeichnet und müssen ihrerseits noch behandelt werden. Um festzustellen, ob sich das Merging von Boxen lohnt, muss eine Bewertung stattfinden. Boxen werden nur dann gemergt, wenn die erzeugte Mergebox profitablere Ergebnisse liefert, als jede der zu mergenden Boxen. Somit ist sichergestellt, dass kein Effizienzverlust durch das Merging entsteht. Dazu wird die Mergebox nach den folgenden Mustern erzeugt und zur Profitfunktion 3.8 geleitet. Danach wird der Profit der Mergebox mit den Profiten der Subsumees verglichen. Ist der Profit der Mergebox größer als der der Mergeboxen, so wird das Merging zugelassen, ansonsten findet kein Merging statt.

Weiterhin wird das Matching von Intervalprädikaten und der Umgang mit Disjunktionen diskutiert.

3.6.1 Merging von Select-Boxen

Muster

Zwei Subsumee-Boxen $b1$, $b2$ vom Typ Select-Box.

Match-Bedingung

1. Boxen $b1$, $b2$ haben mindestens ein gemeinsames Prädikat.

Merge-Box

Die Mergebox entspricht dem Subsumer. Sie enthält die gemeinsamen Prädikate der Subsumees und alle aus den Kindboxen ableitbaren Attribute. Als Kindboxen erhält sie alle Kindboxen der Subsumees, von denen die Prädikate abgeleitet werden können. Als Elternboxen werden die Subsumees hinzugefügt. Die Referenzrate der Box wird aus der Summe der verschmolzenen Boxen ermittelt.

Subsumees

Alle gemeinsamen Prädikate, sowie die Verlinkung zu deren abstammenden Kindboxen, werden gelöscht. Beide Boxen erhalten als neue Kindbox den Subsumer.

Beispiel

In Abbildung 3.13 ist ein Mapgraph während des Mergevorgangs dargestellt. Die Subsumees *Box-MS4* und *Box-MS5* werden miteinander verglichen.

Beide sind vom Typ Select-Box und haben ein gemeinsames Prädikat „ $x = a$ “. Damit ist die Bedingung des Musters erfüllt.

Eine Merge-Box wird erstellt, die das Prädikat „ $x = a$ “ und die Attribute (a, b, c, x, y) der abhängigen Kindboxen erhält. Alle Kindboxen der Subsumees (*Box-MS1*, *Box-MS2*), von denen das Mergeboxprädikat „ $x = a$ “ abgeleitet werden kann, werden dem Subsumer als Kindboxen zugewiesen. Außerdem werden die Subsumees dem Subsumer als Elternboxen zugewiesen.

Den Subsumee-Boxen wird das gemeinsame Prädikat „ $x = a$ “ und die Verlinkung zu den Kindboxen *Box-MS1*, *Box-MS2* entfernt. Der Subsumer wird von beiden als Kindbox hinzugefügt.

Der Mapgraph nach der Optimierung ist in Abbildung 3.14 dargestellt. Die Mergebox vereinigt die Selektion des gemeinsamen Prädikates „ $x = a$ “. Alle nötigen Attribute werden für die Elternboxen bereitgestellt. Die Referenzanzahl der Mergebox entspricht der Summation der Referenzwerte der Subsumees. Insgesamt wurden durch das Merging eine Box eingespart und vielzählige Anfragen auf einen Teilgraph abgebildet.

3.6.2 Merging von Group-By-Boxen

Muster

Zwei Subsumee-Boxen *b1*, *b2* vom Typ Group-By-Box.

Match-Bedingung

1. Boxen *b1*, *b2* haben mindestens ein gemeinsames Prädikat.
2. Die Kindbox der Subsumees ist nicht vom Typ Group-By-Box.

Merge-Box

Der Subsumer erhält die Menge der Prädikate und Attribute der Subsumees. Die gemeinsame Kindbox der Subsumees bekommt der Subsumer als Kindbox zugewiesen. Außerdem werden die Subsumees als Elternboxen dem Subsumer hinzugefügt. Die Referenzrate der Box wird aus der Summe der verschmolzenen Boxen ermittelt.

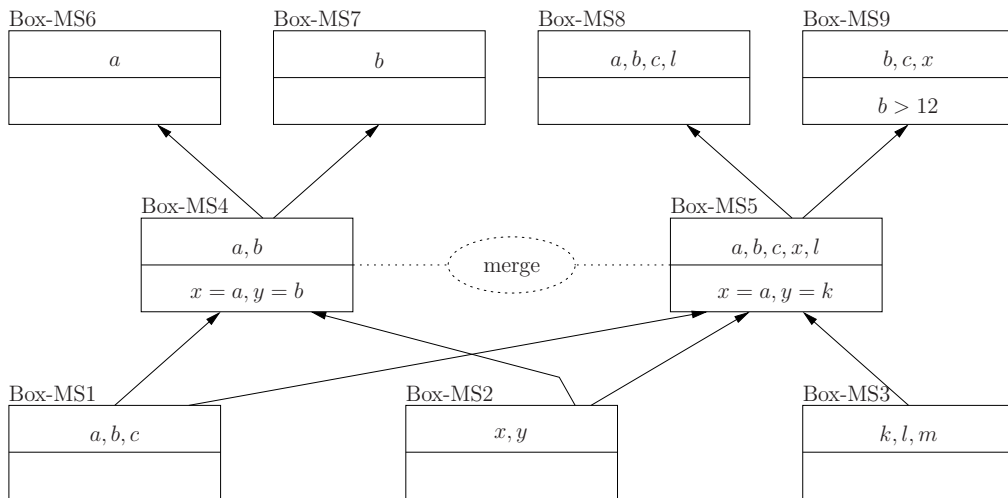


Abbildung 3.13: Merging von Selektionsboxen

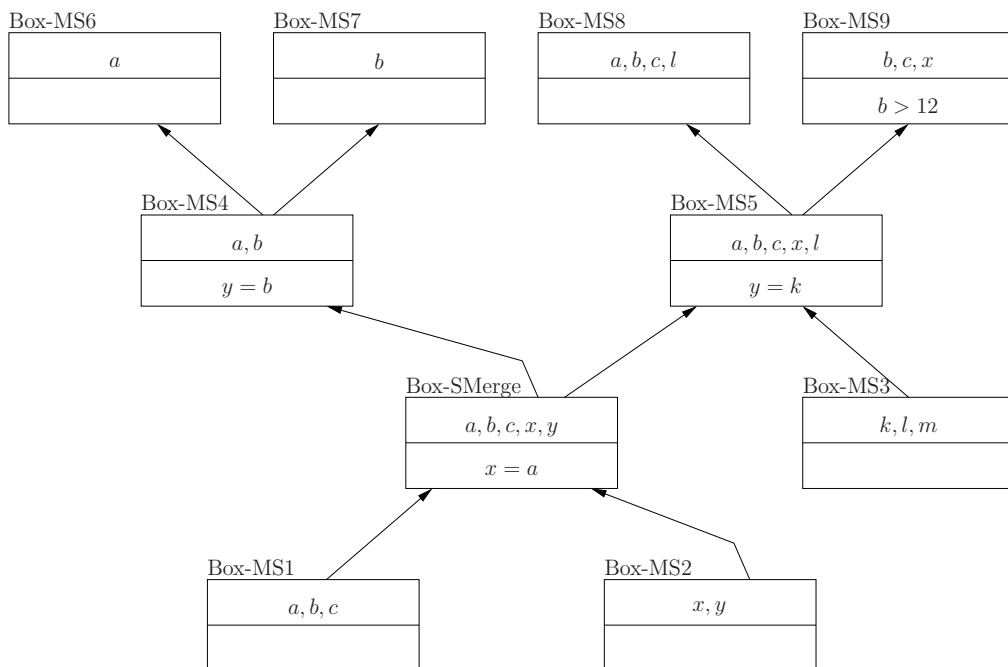


Abbildung 3.14: Mapgraph nach Merging von Select-Boxen

Subsumees

Die Subsumees leiten ihre Aggregatfunktionen von den Subsumer-Aggregatfunktionen ab. Dabei gelten die Ableitungsregeln aus Abschnitt 2.3.7. Die Aggregatfunktionen erhalten im Subsumer einen bestimmten Alias, der der Subsumeebox zeigt, welches Attribut sie ableiten muss. Optimierend könnten bei der *COUNT*-Funktion *NON NULL*-Attribute auf *COUNT(*)* abgebildet werden. Die Verbindung zu der Kindbox wird gelöscht und stattdessen der Subsumer als neue Kindbox genutzt.

Beispiel

In Abbildung 3.15 ist ein Mapgraph abgebildet, in dem zwei Gruppierungsboxen, *Box-MG2* und *Box-MG3*, gemergt werden sollen. Das vorgestellte Muster erfüllt die Voraussetzung, die zum Merging notwendig ist. Beide Boxen haben die gemeinsamen Prädikate *a, b* und die Kindbox der Subsumees ist keine Gruppierungsbox, wodurch auch die Match-Bedingung erfüllt ist.

Das Muster erstellt daraufhin eine Merge-Box, die alle Prädikate sowie Attribute der Subsumees vereinigt. Da Attribut *a* auch Nullwerte enthalten kann, darf die Aggregatfunktion *count(a)* nicht auf die Aggregatfunktion *count(*)* des anderen Subsumees abgebildet werden. Die Aggregatfunktion *count(a)* erhält in der Mergebox einen speziellen Aliasnamen *cntA*, der von *Box-MG3* konsumiert werden muss. Die gemeinsame Kindbox der Subsumees werden dem Subsumer zugeordnet. Als Elternboxen werden der Mergebox die Subsumees hinzugefügt. Die Referenzanzahl der Mergebox entspricht der Summation der Referenzwerte der Subsumees.

Die Subsumees ersetzen ihre Kindbox durch die erstellte Mergebox und leiten die Aggregatfunktion *count* nach *sum* ab. Dabei verwenden sie das durch den Aliasnamen gekennzeichnete Aggregatattribut der Mergebox. *Box-MG2* konsumiert das Attribut *cnt* und *Box-MG3* das Attribut *cntA* der Mergebox. Durch die Ableitung ist sichergestellt, dass die Boxen die gleichen Tupel repräsentieren, wie vor dem Merging. Abbildung 3.16 zeigt den Mapgraph nach der Mergingoperation.

Aufgrund der zweiten Bedingung des Musters ist ein erneuter Match zwischen den Subsumees unmöglich. Diese Bedingung wird nötig, damit keine Endlosschleife beim Merging von Gruppierungsboxen entsteht. Die gemergten Boxen behalten beim Merging ihre Prädikate und würden ohne diese Bedingung beim nächsten Durchlauf des Mergingalgorithmus wieder vermischt. Die Mergebox würde erneut erstellt und an die vorhergehende Mergebox angefügt. Die Subsumees würden eine Ebene nach oben verschoben. Dieser Prozess wäre unendlich bzw. dann vorbei, wenn nicht mehr ausreichend Speicher zur Verfügung steht.

3.6.3 Intervalprädikate

Beim Merging von Boxen, die Intervalle von Tupeln selektieren, muss auf die Teilmengenbedingung geachtet werden. Während beim Einfügen und Restrukturieren von Boxen die Teilmengenbedingung gelten muss (das heißt, der Subsumer muss mindestens alle Tupel des Subsumee enthalten), gilt diese Bedingung beim Merging nur begrenzt. Gefordert wird nur eine minimale Überlappung der Relationen. Die Bedingung aus Unterabschnitt 3.6.1, dass die Boxen mindestens ein gemeinsames Prädikat aufweisen müssen,

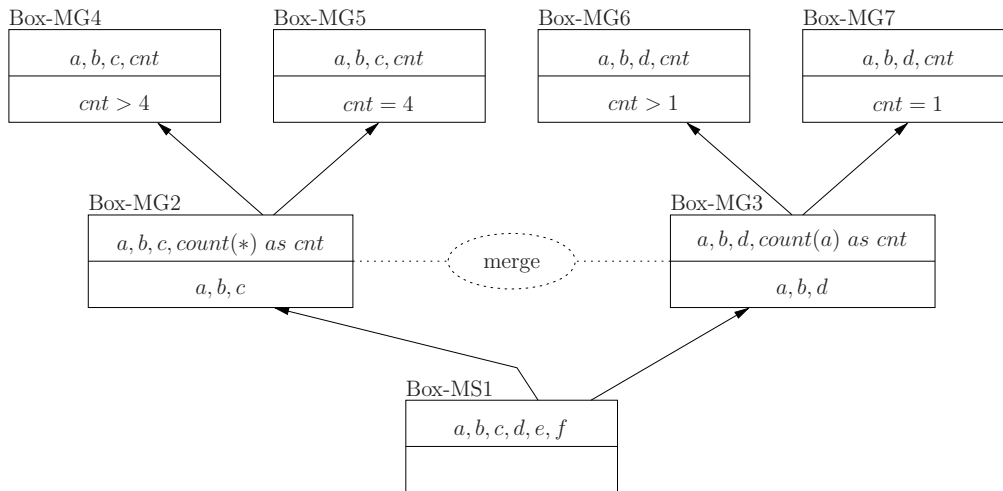


Abbildung 3.15: Merging von Gruppierungsboxen

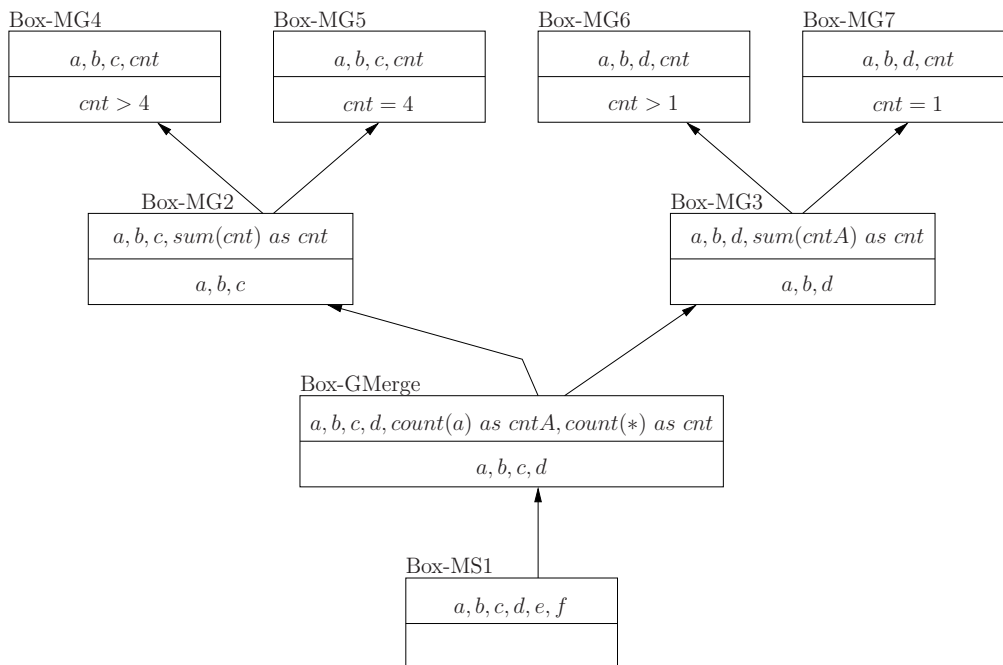


Abbildung 3.16: Mapgraph nach Merging von Group-By-Boxen

gilt für Intervallprädikate nur insofern, dass die Boxen gemeinsame Teilrelationen enthalten. Also die Subsumees eine gemeinsame Schnittmengenrelation haben. Die Erfüllung dieser Bedingung führt nicht zwingend zum Merging. Auch hierfür müssen die Boxen bewertet werden, so dass anhand des sich ergebenden Profits entschieden werden kann, ob ein Merging realisiert wird oder nicht.

3.6.4 Konjunktionen und Disjunktionen

Die bisherigen Muster der dynamischen Anfragebeantwortung haben ausschließlich die Konjunktion von Boxprädikaten berücksichtigt. Es muss daher geklärt werden, wie mit disjunktiven Ausdrücken umgegangen wird.

Jede Anfrage kann mittels algebraischer Regeln von KNF in DNF umgewandelt werden [15]. Die disjunktive Form entspricht der Vereinigung der durch die konjunktiven Teile repräsentierten Relationen. Um die Strategie beim Matching nicht grundlegend verändern zu müssen, werden disjunktive Ausdrücke als ein spezielles Prädikat gekapselt. Beim Matching der Box muss nur die Prädikatmatchfunktion angepasst werden.

Die Prädikatmatchfunktion wandelt dazu die Prädikate von Subsumee und Subsumer von KNF in DNF um und matcht jede Konjunktion der Subsumeeprädikate mit allen Konjunktionen der Subsumerprädikate. Sollten alle Konjunktionen des Subsumees positiv matchen, so ist auch der Prädikatmatch positiv.

Die Anfrage in Abbildung 3.17 stellt eine SQL-Anfrage in KNF dar. Das Pendant in Abbildung 3.18 repräsentiert die gleiche Anfrage in disjunktiver Normalform. Beide Anfragen liefern die gleiche Ergebnisrelation.

```
SELECT name, vorname
FROM Person
WHERE name='Meier'
      and (vorname='Hans' or vorname='Heinz')
      and gebdatum>'1.1.1980'
```

Abbildung 3.17: Anfrage in konjunktiver Form

```
SELECT name, vorname
FROM Person
WHERE (
      name='Meier'
      and vorname='Hans'
      and gebdatum>'1.1.1980'
    )
OR
(
      name='Meier'
      and vorname='Heinz'
      and gebdatum>'1.1.1980'
    )
```

Abbildung 3.18: Anfrage in disjunktiver Form

Wird eine Anfrage, wie in Abbildung 3.19, mit Boxen des Mapgraphs, repräsentiert durch Abbildung 3.17, gematcht, so muss die Teilmengenbedingung geprüft werden. Dazu

werden die Konjunktionen der Subsumeeprädikate mit jeder Konjunktion der Subsumerprädikate gematcht. Die Subsumeeprädikate selektieren eine Teilmenge einer Konjunktion ($name='Meier'$ and $vorname='Hans'$ and $gebdatum>'1.1.1980'$) des Subsumers. Damit ist die Teilmengenbedingung und somit der Prädikatmatch positiv erfüllt.

```
SELECT name, vorname
FROM Person
WHERE name='Meier'
      and vorname='Hans'
      and gebdatum>'1.1.1989'
```

Abbildung 3.19: Anfrage, die gematcht werden soll

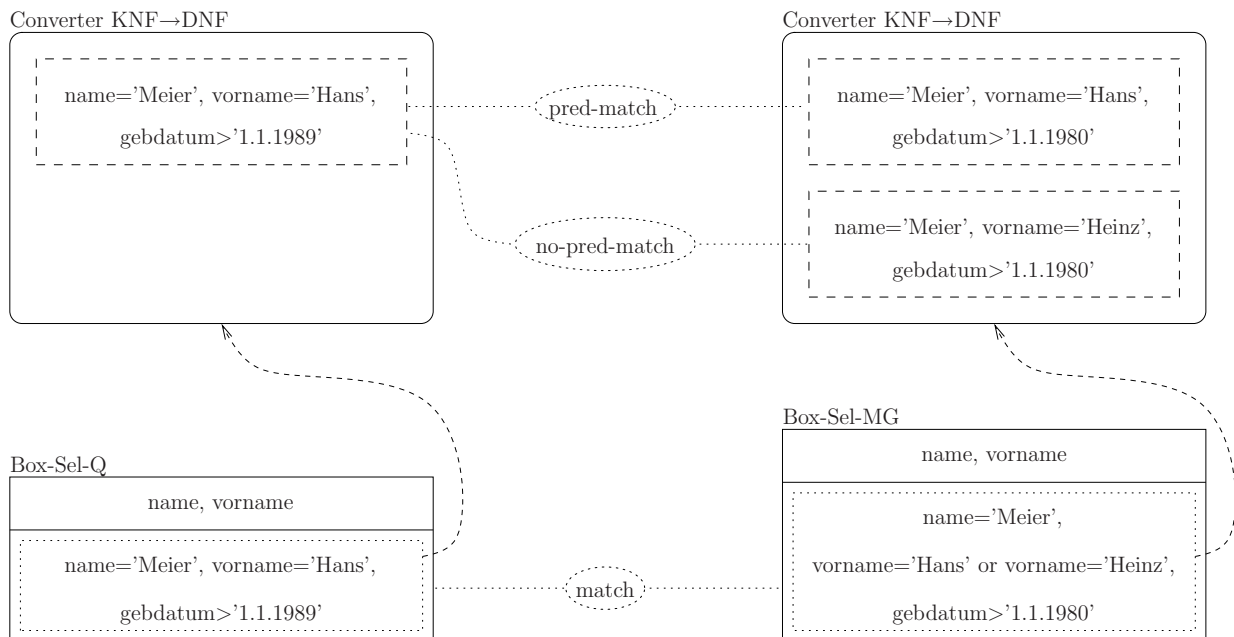


Abbildung 3.20: Match disjunktiver Prädikatmengen

Abbildung 3.20 zeigt diesen Sachverhalt als QGM. Es existiert ein Konverter, der die Prädikatmenge von KNF in DNF überführt. Dabei stellt *Box-Sel-Q* die Prädikate der Anfrage aus Abbildung 3.19 und *Box-Sel-MG* die Prädikate der Anfrage aus Abbildung 3.17 dar. Um festzustellen, ob ein Match existiert, wird die Prädikatmenge der Box *Box-Sel-MG* dem Konverter übergeben, der die Prädikatmenge in DNF überführt. Die gebildeten Konjunktionen werden mit der einzigen Konjunktion aus *Box-Sel-Q* zur Prädikatmatchfunktion geleitet, die den Match mit einer der Konjunktionen aus *Box-Sel-MG* positiviert.

3.7 Aging

Bei der dynamischen Bestimmung von materialisierten Sichten besteht das Problem, dass sich ein ändernder Workload nicht optimal in Bezug auf die Repräsentation der Anfragen äußern kann. Die Referenzierung der Boxen bewirkt, dass sich häufiger gestellte

Anfragen positiver bei der Profitberechnung der Boxen herausstellen. Dieses Verhalten ist bei einem gleichbleibendem Workload auch erwünscht, jedoch führt es bei einem variablen Workload zu einer annähernd statischen Menge von materialisierten Sichten. Der Mapgraph würde Anfragen verstärkt profitieren lassen, die nicht mehr vom aktuellen Workload genutzt werden. Neuere Anfragen könnten erst über längere Zeit auf materialisierte Sichten abgebildet werden, wodurch das System statisch bzw. sehr träge beim Vorschlagen von materialisierten Sichten werden würde. Außerdem führt die Dekrementierung von Boxen zur Verringerung der Anzahl der Boxen im Mapgraph. Immer wenn der Workload sich ändert, würden Boxen im Mapgraph, die nicht mehr verwendete Anfragen des Workloads repräsentieren, ohne die Agingfunktion als ewige Relikte im Mapgraph verbleiben und diesen „aufblähen“. Dies hat zur Folge, dass möglicherweise ein erheblich höherer Verwaltungsaufwand des Mapgraphs entsteht.

Aus diesem Grund ist es nötig, die Boxenreferenzen in bestimmten Zeit- oder Anfrageabständen zu dekrementieren, damit neuere Anfragen schneller auf die Menge der materialisierten Sichten abgebildet werden können. Dabei muss beachtet werden, dass zu langsames Dekrementieren statisch wirken und zu schnelles ein zu häufiges sprunghaftes Ändern der ASTs bewirken würde. Zu schnelles Dekrementieren äußert sich besonders negativ bei häufigen Workload-drifts, also der sprunghaften Änderung des Workloads über einen gewissen Zeitraum. Die vorgeschlagenen ASTs würden im schlechtesten Fall mit anderen ASTs in der Materialisierung alternieren. Der dadurch entstehende Mehraufwand könnte sämtlichen Effizienzgewinn, der aus den ASTs resultiert, wettmachen. Aus diesem Grund ist ein „intelligentes“ Dekrementieren nötig, dem im Gesamtrahmen eine gesonderte Arbeit gewidmet sein sollte.

Das hier verwendete Dekrementierungsverfahren basiert auf dem Inkrementieren des Referenzzählers bis zu einer definierten Obergrenze. Erreicht eine beliebige Box im Mapgraph MG diese Obergrenze, werden die Referenzzähler $ref(b)$, mit $b \in MG$, aller Boxen im Mapgraph um einen Faktor F ganzzahlig geteilt.

$$ref_{new}(b) = \frac{ref_{cur}(b)}{F}$$

Unterläuft der Referenzwert einer Box einen unteren Schwellenwert, so wird sie aus dem Mapgraph entfernt. Dieses Vorgehen vermeidet Ineffizienzen durch einen zu großen Mapgraph, der Anfragen abbildet, die einem nicht mehr verwendeten Workload angehören. Durch die Wahl eines geeigneten Maximalreferenzwertes und Divisors kann der Mapgraph auf einen sich gleichmäßig ändernden Workload eingestellt werden und Workload-drifts zu einem gewissen Grad entgegenwirken. Die Einstellung dieser Werte kann entweder manuell vom Anwender vorgenommen oder automatisch durch die Auswertung von hierfür in Frage kommender Heuristiken ermittelt werden.

Beispiel

Abbildung 3.21 stellt einen Mapgraph dar, dessen Boxen bereits eine Referenzierung aufweisen. Diese ist schematisch mit jeder Box verknüpft. So wurde z. B. *Box-MG1* genau 999 Mal referenziert.

Nebenbei bemerkt ist die Summe der Referenzanzahl der Elternboxen von *Box-MG1* auch 999. Demzufolge gab es keine Anfrage im Workload, die ausschließlich dem Teil-

graph, ausgehend von *Box-MG1*, entsprach. Betrachtet man die Summe der Referenzanzahl der Boxen *Box-MG4* und *Box-MG5*, so stellt man fest, dass sie unterhalb der Referenzanzahl der einzigen gemeinsamen Kindbox *Box-MG2* liegt. Daraus folgt, dass es Anfragen im Workload geben muss, die ausschließlich auf dem Teilgraph von *Box-MG2* abgebildet wurden. Die Differenz der Referenzanzahl zwischen dieser Box und der Summe der Referenzanzahl der Elternboxen entspricht 25 Anfragen.

Die maximale Referenzanzahl soll mit $ref_{max} = 1000$, der Divisor mit $F = 3$ und der untere Schwellenwert mit 0 angenommen werden. Wird beim nächsten Einfügen einer Anfrage *Box-MG1* positiv gematcht, so erhöht sich der Referenzwert dieser Box auf die maximale Referenzanzahl ref_{max} . Daraufhin wird auf alle Boxen $b \in MG$ eine Neureferenzierung mittels der Berechnungsvorschrift für $ref_{new}(b)$ vorgenommen. Der resultierende Mapgraph mit den dekrementierten Referenzen ist in Abbildung 3.22 dargestellt.

Es soll ein Workload simuliert werden, der ab einem Zeitpunkt keine Anfragen mehr auf dem Teilgraph oberhalb von *Box-MG3* abbildet. Es werden nur noch die Boxen, die nicht diesem Teilgraph entsprechen, inkrementiert. Durch die folgenden Dekrementierungen aller Mapgraphboxen werden *Box-MG6* und *Box-MG7* um den Faktor F bei jeder Rereferenzierung kleiner, bis schließlich die Referenzanzahl 0 erreicht ist. Tabelle 3.2 stellt den Verlauf der Dekrementierung dar. Dabei entspricht der Level der Dekrementierungsstufe. Die *Box-MG6* entsprach vor der Entfernung aus dem Workload ca. jeder

Level	Box-MG6	Box-MG7
0	511	89
1	170	29
2	56	9
3	18	3
4	6	1
5	2	0
6	0	-
7	-	-

Tabelle 3.2: Aging von Boxen

zweiten gestellten Anfrage. Bei der nächsten Dekrementierungsstufe entspricht sie nur noch jeder 5,8 Anfrage. Dieser Trend setzt sich fort, so dass die Box nach der 7. Dekrementierungsstufe aus dem Mapgraph gelöscht wird. Man kann also mittels geeigneter Dekrementierungswerte (ref_{max}, F) eine materialisierte Sicht schneller bzw. langsamer vom einem sich ändernden Workload beeinflussbarer machen.

Bei diesem Aging-Verfahren kommt es nicht so sehr auf das Löschen der Boxen an. Vielmehr sollen die Referenzen anschaulich den Workload bei der Profitberechnung repräsentieren, ohne dabei anfällig für Workloaddrifts zu sein. Dabei sollte der maximale Referenzierungsgrad abhängig von der „Breite“ des Mapgraphs gewählt werden, da sonst Boxen entfernt werden könnten, die im Workloadzyklus regelmäßig auftreten, aber durch

eine schnelle Dekrementierung nicht ausreichend inkrementiert werden. So kann eine Box, die mit wenig Referenzen aber mit hohem Berechnungsaufwand für eine Materialisierung in Frage kommt, aus dem Mapgraph entfernt werden, bevor die Profitberechnung stattfindet. Aus diesem Grund ist eine dynamische Anpassung vorteilhafter.

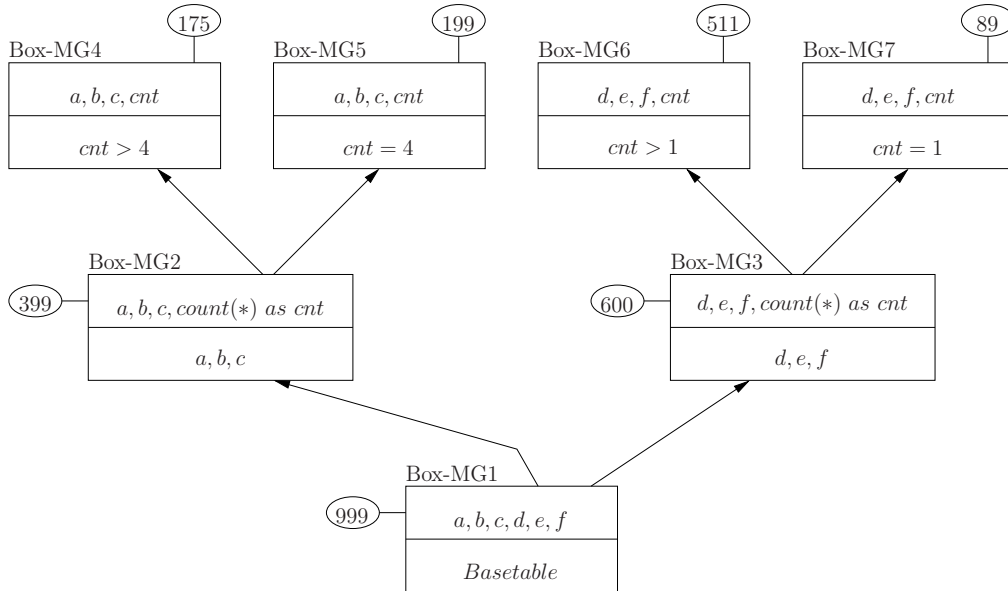


Abbildung 3.21: Mapgraph vor der Aging-Operation

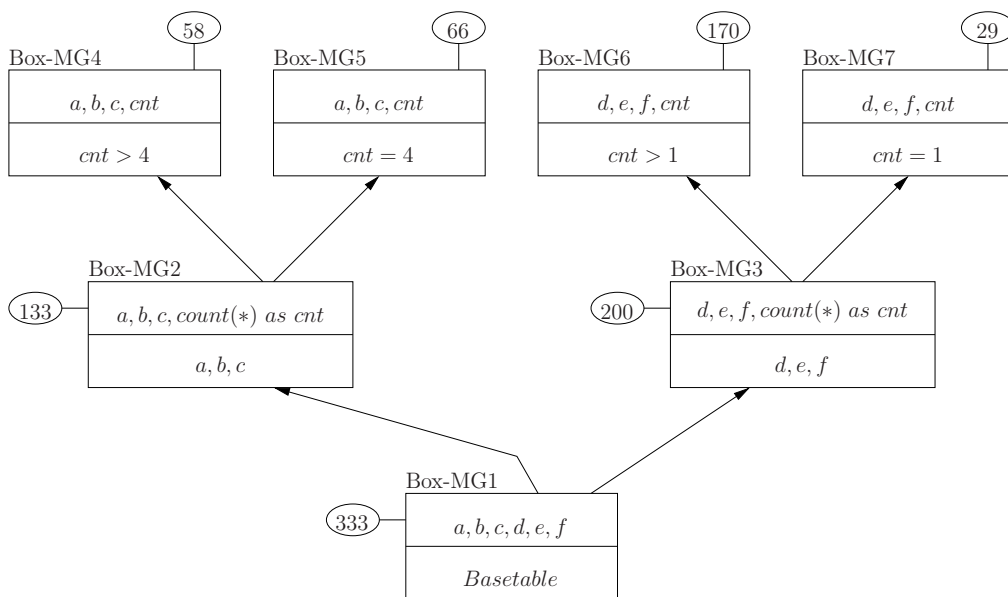


Abbildung 3.22: Mapgraph mit dekrementierten Boxreferenzen

3.8 Profitberechnung

Um eine bestmögliche Repräsentation der Anfragen des Workloads durch einzelne Boxen zu erhalten, muss jede Box im Mapgraph bewertet werden. Diese Aufgabe übernimmt die

Profitfunktion, die aus den Eigenschaften einer Box den relativen Repräsentationswert berechnet. Je höher dieser Wert, desto repräsentativer vertritt diese Box den Workload. Dabei positivieren die Operationskosten und Referenzanzahl den Profit, während die durch die Box vertretende Relationsgröße, bestehend aus Tupelgröße und Tupelanzahl, negativ wirkt. Die Referenzanzahl wird bei Boxen, die mehrere Anfragen abdecken, stärker zur Geltung kommen, da die Box mehrfach gematcht und damit auch mehrfach die Referenzanzahl inkrementiert wird. Dieses Verhalten ist zur Profitberechnung dringend notwendig, da materialisierte Sichten auch eine Obermenge vieler Anfragen darstellen. Jedoch muss ebenso die Relationsgröße der Box berücksichtigt werden, da dem System nur eine begrenzte Speichergröße zum Materialisieren zur Verfügung steht.

Gegeben ist eine Box b , die Operationskostenfunktion $operationCost(Box)$, Tupelanzahlfunktion $tupleCount(Box)$, Tupelgrößefunktion $tupleSize(Box)$ und Referenzanzahlfunktion $referenceCount(Box)$. Die Funktionen ermitteln jeweils die Eigenschaftswerte des zugehörigen Eigenschaftsnamen. Die Profitfunktion $profit(Box)$ ist wie folgt definiert:

$$profit(b) = \frac{operationCost(b)}{tupleCount(b) * tupleSize(b)} * referenceCount(b)$$

Ein wichtiges Kriterium bei der Profitberechnung betrifft die Aktualisierungshäufigkeit der abhängigen Basistabellen einer Mapgraphbox. Normalerweise müsste dieser Wert mit in die Profitberechnung einfließen, um materialisierte Sichten mit häufigerem Updatecharakter negativer zu bewerten. Änderungen der Basistabellen haben zur Folge, dass die Integrität einer abhängigen AST nicht mehr gewährleistet ist und diese nach jeder Änderung der abhängigen Basistabelle neu aufgebaut werden muss. Die Aktualisierung führt dazu, dass das System möglicherweise mehr damit, als mit der Beantwortung der Anfragen beschäftigt ist. Auf der anderen Seite wird eine Materialisierung mit häufigem Updatecharakter unter Umständen vom Benutzer gewünscht. Daher wird dieser Sachverhalt offen gehalten. Die Implementation kann dann nach den Wünschen des Benutzers gestaltet werden. Denkbar wäre eine Kombination aus beiden. Der Benutzer kann selbst materialisierte Sichten erstellen und gleichzeitig ASTs vom System verwalten lassen. Leider widerspricht dieser Grundsatz der vollständigen Automatisierung bei der Erstellung von ASTs, sollte aber trotzdem für anspruchsvollere Benutzer optional zur Verfügung stehen.

Es gäbe eine weitere Funktion $updateBehavior(Box)$, die einen Wert für die Aktualisierung der abhängigen Basistabellen liefert. Je höher dieser Wert, desto mehr Aufwand entsteht bei Aktualisierung der zu erstellenden AST. Damit ergibt sich eine alternative Profitfunktion $profit_{update}(Box)$, die das Updateverhalten der Basistabellen mit einberechnet.

$$profit_{update}(b) = \frac{operationCost(b) * referenceCount(b)}{tupleCount(b) * tupleSize(b) * updateBehavior(b)}$$

Diese Profitfunktion würde den Berechnungsaufwand für die Aktualisierung der materialisierten Sichten mitberücksichtigen. Auf den ersten Blick erscheint diese Methodik besser zu sein, jedoch müsste zusätzlich die Aktualisierungszeit mitbetrachtet werden. Realistisch wäre ein Szenario, in dem das Updateverhalten der Basistabellen einer materialisierten Sicht sehr hochfrequent wäre. Somit müsste die materialisierte Sicht sehr

oft aktualisiert werden. Stellt man sich vor, dass nur nachts bzw. außerhalb der Hauptworkloadzeit Änderungsoperationen auf den Basistabellen ausgeführt werden, so wäre zwar das Updateverhalten stark ausgeprägt, jedoch würde es den Nutzen der AST zur Hauptworkloadzeit keinesfalls einschränken. Um diesen Fakt ausreichend zu berücksichtigen, wären weitere Informationen im Bezug auf die Aktualisierungszeiten der Basistabellen nötig. Dies macht die Berechnung des Nutzens einer Box zu einer weiteren anspruchsvollen Aufgabe. Um diesen Sachverhalt auszugrenzen, wird in dieser Arbeit die erste Profitfunktion ohne die Berücksichtigung des Aktualisierungscharakters genutzt. Da die Profitfunktion modular implementiert wird, sollte ein transparenter Austausch durch eine geeignete Profitfunktion kein Problem darstellen.

3.9 Auswahl der materialisierten Sichten

Die Auswahl der materialisierten Sichten erfolgt in regelmäßigen Anfrageabständen. Dazu wird auf allen Mapgraphboxen, außer den Tabellenboxen, die Profitfunktion angewendet. Das Ergebnis ist eine Liste von Box-Profitwerte-Paaren. Diese wird absteigend nach den Profitwerten sortiert. Die Relationsgrößen der Boxen werden absteigend sukzessiv summiert und falls möglich, von der Cachegröße abgezogen. Ist die Größe einer Boxrelation größer als die restliche Cachegröße, wird sie nicht von der restlichen Cachegröße abgezogen, sondern das Box-Profitwerte-Paar aus der Liste entfernt. Nach dem Durchlauf besteht die Liste nur noch aus Boxen, die nach der Materialisierung vollständig in den Cache passen. Dann werden alle ASTs die nicht in der Materialisierungsliste vorkommen, aus dem Cache gelöscht. Umgekehrt werden alle Boxen, die nicht als AST im Cache repräsentiert werden, materialisiert.

Tabelle 3.3 zeigt ein Beispiel, bei dem nach Profit sortierte Boxen mit zugehörigem Profit und Relationsgröße dargestellt werden. Zusätzlich gibt es eine Spalte *Cachegröße*, die den freien Cachespeicher nach Abarbeitung der Box anzeigt. Dabei hat die Box mit ID 5 den höchsten (1000) und die Box mit ID 2 den niedrigsten Profit(430). In diesem Beispiel soll die maximale Cachegröße mit 1000 angenommen werden. Dabei handelt es sich in allen Spalten um Beispielwerte, die die Erklärung des Verfahrens vereinfachen sollen, jedoch kein Zusammenhang mit der Realität herstellen. Die erste Box (ID 5) der Liste passt in den Cache, da ihre Größe (300) kleiner als die restliche Cachegröße ist. Daher bleibt sie in der Liste enthalten und es wird mit der nächsten Box (ID 1) der Liste fortgefahren. Diese und die beiden nächsten Boxen (ID 7,3) passen ebenso in den Cache. Eng wird es erst für die darauf folgende Box (ID 6). Ihre Materialisierung (Größe 320) würde die restliche Cachegröße von 240 überlaufen. Aus diesem Grund kann die Box nicht materialisiert werden, worauf sie aus der Liste entfernt wird. Die darauffolgenden Boxen (ID 0,4) kommen aufgrund ihrer in den Cache passenden Relationsgrößen wieder zur Materialisierung in Frage und verbleiben in der Liste. Alle weitere Boxen können nicht materialisiert werden, da ihre Relationsgröße über der restlichen Cachegröße von 10 hinausragt.

Damit verbleiben die Boxen 5,1,7,3,0,4 in der Liste, die materialisiert werden müssen. Nehmen wir an, dass der bisherige Cache aus den Boxen mit ID 5,1,7,6,2 besteht. Das bedeutet, dass die ASTs mit ID 6,2 aus dem Cache gelöscht und die Boxen 3,0,4 materialisiert werden müssen. Man könnte daraus schließen, dass sich der vor kurzem ereignete Workload vermehrt auf Anfragen der Box 3 bezogen haben muss, da sie bei der vorherigen

Box-ID	Profit	Rel-Gr.	Cache-Gr.
5	1000	300	700
1	900	250	450
7	850	80	370
3	760	130	240
6	740	320	DEL
0	710	110	130
4	600	120	10
2	430	45	DEL
*	...	> 10	DEL

Tabelle 3.3: Auswahl von materialisierten Sichten

Materialisierung anscheinend schlechter bewertet wurde. Dies ist aber nur eine Annahme. Genauso gut kann eine andere Workloadänderung aufgetreten sein. Zum Beispiel könnte Box 6 weniger häufig angefragt worden sein.

Der hier verwendete Algorithmus arbeitet nach dem Greedy-Verfahren. Dabei werden materialisierte Sichten mit Indexen ausgewählt, die nicht notwendigerweise das optimale Ergebnis darstellen. Um ein optimales Ergebnis zu erhalten, müssten alle Kombinationen von materialisierten Sichten mit Indexen permutiert und bewertet werden (Rucksackproblem [11]). Anfragen sind oft so gestaltet, dass sie eventuell mehrere Teilanfragen enthalten, die unterschiedliche Sichten nutzen. Aus diesem Grund wäre die Materialisierung von Boxen, die beim Greedy-Verfahren nicht zum Tragen kommen, aber für eine optimale Lösung nötig sind, wünschenswert. Dieser Tatsache spricht entgegen, dass die Permutation und Bewertung aller Hüllen von AST-Mengen mit Indexen exponentiellen Berechnungsaufwand bedeuten würde, was unter realen Umständen nicht realisierbar wäre. Daher wird das Greedy-Verfahren verwendet, das zwar nicht unbedingt ein optimales, dafür aber ein in polynomieller Zeit zu berechnendes und an das Optimum angenähertes Ergebnis liefert.

3.10 Indexe

Eine weitere wichtige Fähigkeit bei der Optimierung ist die automatische Erzeugung von Indexen [13] auf materialisierte Sichten. Indexe sollten auf Prädikate, die häufig angefragt und aufwendig zu berechnen sind, abgebildet werden. Beim Mapgraph beinhalten die Elternboxen der AST-Box die Prädikate, die bei Anfragen mit der materialisierten Sicht als Teilanfrage berechnet werden müssen. Sicherlich ist es nicht nötig, alle Elternboxprädikate als Indexe zu übersetzen, da sich die Ersparnis des Berechnungsaufwands im Verhältnis zum Speicherverbrauch kaum rentieren dürfte. Daher wird eine Bewertung der Elternboxen benötigt, so dass möglichst eine optimale Menge von Indexen daraus erzeugt werden kann.

Man könnte ähnlich zum Abschnitt 3.9 rekursiv die Elternboxen mittels der Profitfunktion bewerten und die Prädikate der profitabelsten Boxen auf Indexe abbilden. Dabei darf die Relationsgrößenberechnung aufgrund der eingeschränkten Dimensionalität der Indexe nur auf Attribute angewendet werden, die von den Prädikaten der Box abgeleitet werden können.

Dazu wird die Elternbox geklont, wobei die Klonoperation nur Attribute kopiert, die von den Prädikaten abgeleitet werden können. Die Prädikate müssen von den Attributen der AST ableitbar sein. Ist ein Prädikat nicht ableitbar, wird dieses Prädikat, als auch das Attribut, dass von diesem Prädikat ableitbar ist, entfernt. Dieser Sachverhalt tritt ein, wenn es sich bei der Elternbox um eine Verbundbox handelt und die Prädikate von einer anderen Kindbox als der AST abgeleitet wurden. Zusätzlich werden Attribute hinzugefügt, die nicht in der Attributmenge vorkommen, die aber von Prädikaten abgeleitet werden können, welche wiederum von der AST abstammen.

Die geklonte Box wird dann zur Profitfunktion geleitet, die den Profit der Box berechnet. Kommt es bei den Boxen zur Überlappung der Indexabbildungen durch ähnliche Prädikatdefinitionen, werden die Profite für die Erstellung dieses Indexes aufsummiert. Damit werden Muster eingespart, welche die Boxen mit ähnlichen Prädikaten vorher hätten zusammenfassen müssen. Durch die bessere Bewertung des Indexes steigt die Chance auf eine Materialisierung, da dieser in der Liste der Kandidaten möglicherweise aufsteigen könnte.

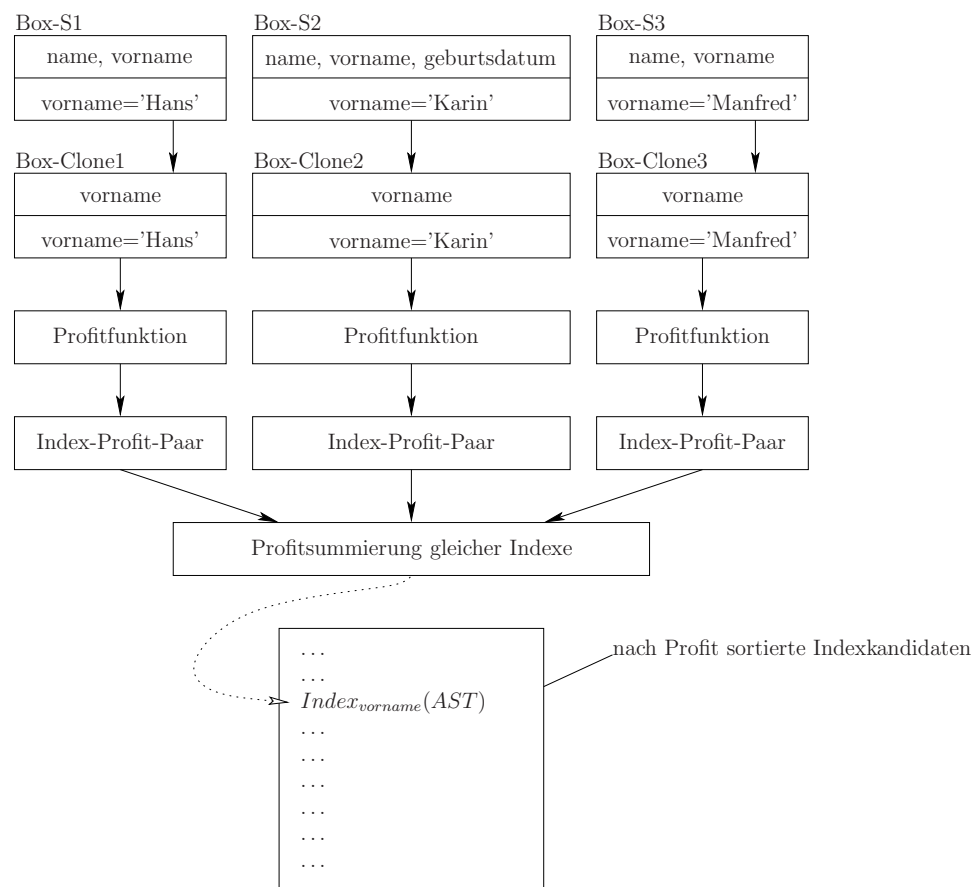


Abbildung 3.23: Ermittlung der Indexkandidaten

Abbildung 3.23 zeigt die Ermittlung von Indexkandidaten anhand eines Beispiels. Es existieren drei Boxen *Box-S1*, *Box-S2*, *Box-S3*, die zur Indexauswahl in Frage kommen. Diese werden geklont, worauf sie nur noch das gemeinsame Attribut *vorname* besitzen. Die Profitfunktion bewertet die geklonten Boxen mittels der Profitfunktion. Anschließend werden die Profite gleicher Boxen aufsummiert und die Indexkandidaten anhand ihrer Profite in die Kandidatenliste einsortiert. In diesem Beispiel wurde der Index auf den dritten Rang einsortiert.

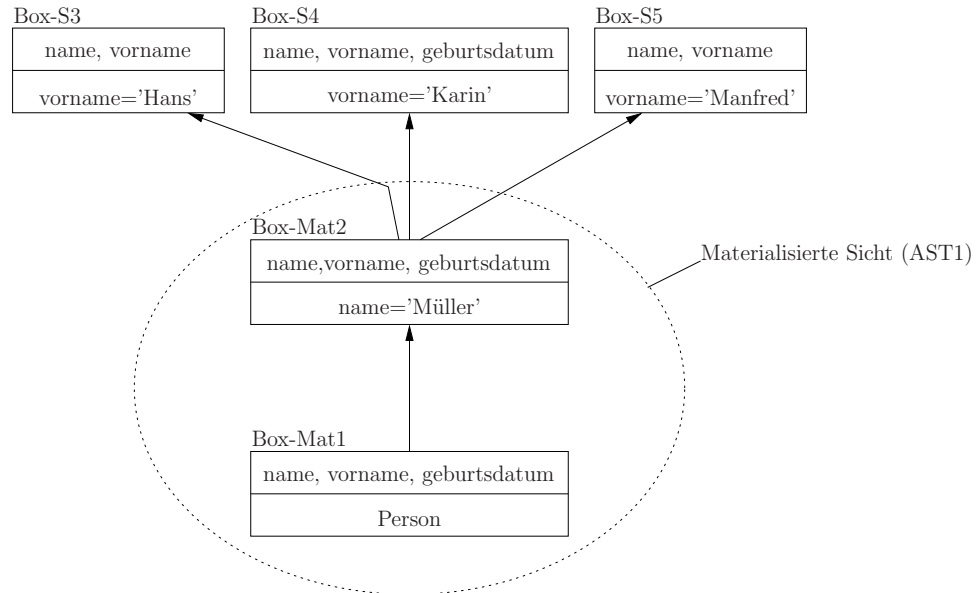


Abbildung 3.24: Materialisierte Sicht mit Elternboxen

Abbildung 3.24 verdeutlicht den Sachverhalt beim Auswählen von Indexen. Dargestellt ist die Projektion eines Mapgraphs, in dem ein Teilgraph ab *Box-Mat2* materialisiert worden ist. Die Elternboxen von *Box-Mat2* stellen Anfragen des Workloads dar, die diese AST nutzen. In allen dreien wird das gleiche Attribut *vorname* verglichen, so dass auch der gleiche Index vorgeschlagen wird. Laut Definition werden die Profite der Boxen summiert und ein Indexkandidat wird erstellt, der auf die materialisierte Sicht *AST1* zeigt. Der Index repräsentiert die drei Elternboxen *Box-S3*, *Box-S4*, *Box-S5*, wobei die Position in der Kandidatenliste dem Profit des Index's, errechnet aus den Profiten der Elternboxen, entspricht. Dieser stellt einen eindimensionalen Sekundärindex mit dem Attribut *vorname* dar. Die Auswahl der Indexe verläuft ähnlich wie die Auswahl der materialisierten Sicht (Abschnitt 3.9), nur dass die Summierung der Boxprofite für gleiche Indexe hinzukommt.

Bei der Profitberechnung spielen zusätzliche Faktoren, wie dünn- oder dichtbesetzte Indexstruktur eine Rolle. Geclusterte Indexe könnten als dünnbesetzte Indexe abgebildet werden. Dieser Fakt müsste bei der Indexgrößenberechnung mit einfließen. Dazu sind allerdings Bewertungsstrategien nötig, die ermitteln, ob sich ein dünnbesetzter Index lohnt oder nicht. Dies hängt wiederum davon ab, wie mit verfügbarem Speicher umgegangen wird. Falls vom Nutzer die schnelle Anfragebearbeitung im Vordergrund steht, ist von einem dünnbesetzten Index abzusehen, da dieser einen höheren Aufwand beim Finden der Datensätze hat, als ein dichtbesetzter Index. Umgekehrt gilt das ebenso. Soll nur wenig Speicher für Indexstrukturen verbraucht werden, ist die Verwendung und bessere Bewer-

tung von dünnbesetzten Indexen vorzuziehen. Dieser Sachverhalt kann also nur geklärt werden, indem der Nutzer seine Konfiguration der Indexauswahlstrategie dem System vorher mitteilt. Sollten sich die Tatsachen im Laufe der Zeit ändern, da z. B. plötzlich der Speicherplatz erweitert wurde und dadurch auch mehr Speicher zum Verwalten der Indexe zur Verfügung steht, kann ohne Weiteres die Konfiguration geändert werden. Das System sollte daraufhin dichtbesetzte Indexe den dünnbesetzten vorziehen und damit bei der nächsten Indexberechnung Indexe auswählen, die die Berechnungszeit einiger Anfragen verkürzen könnten.

3.11 Zusammenfassung

Die dynamische Ermittlung und Verwaltung von materialisierten Sichten basiert auf einem azyklischen Graphen, dessen Knoten verschiedene Boxtypen repräsentieren. Jede Box hat zugewiesene Eigenschaften, wie Tupelanzahl, Tupelgröße, Berechnungsaufwand und Referenzanzahl. Diese Eigenschaften werden zur Profitberechnung genutzt, um Boxen vergleichbar zu machen.

Die Zerlegung der Anfragen erfolgt ähnlich der Zerlegung der dynamischen Anfragebearbeitung (Abschnitt 2.3.2), nur dass zusätzlich Tabellenboxen eingeführt werden, die statistische Daten der Basistabellen repräsentieren.

Es existiert ein Mapgraph, auf dem der Workload abgebildet wird. Die zerlegten Anfragen werden über den Navigator in den Mapgraph eingefügt und müssen eventuell restrukturiert und gemergt werden. Das Matching der Anfragen mit dem Mapgraph entspricht dem Matching des Grundlagenkapitels (Abschnitt 2.3.3) plus zusätzlicher Muster für das Mergingverfahren.

Zur Auswahl geeigneter Boxen aus dem Mapgraph zur Erstellung von materialisierten Sichten ist eine Bewertung aller Boxen, außer den Tabellenboxen des Mapgraphs, mittels einer Profitfunktion nötig. Die Boxen werden nach dem Profit sortiert und, solange genug Speicher zur Verfügung steht, materialisiert.

Zum schnelleren Zugriff auf die materialisierten Sichten werden Indexe benötigt. Diese werden aus den Elternboxen der AST-Boxen gewonnen. Dazu wird der Profit der Boxen gemessen und ähnlich wie die Auswahl der AST sortiert und materialisiert.

Insgesamt wird der Workload komplett auf dem Mapgraph abgebildet, wobei Überlappungen der Anfragen zusammengefasst werden. Jede Box im Graph wird dabei referenziert, so dass die Nachfrage der Boxen bei der Profitberechnung berücksichtigt werden kann. Mergingoperationen, die den Mapgraph wesentlich optimieren, aber auch verschlechtern können, werden durch Berechnung der Profitabilität des Mergings kontrolliert und die Entscheidung eines Mergings von Boxen somit optimiert.

Kapitel 4

Implementation und Evaluation

4.1 Einleitung

Dieses Kapitel beschäftigt sich mit der Umsetzung des Konzeptes in eine Applikation, die zum Evaluieren benötigt wird. Als Vergleichsapplikation soll der *Design Advisor* von IBM DB2 (Version 9.1) genutzt werden.

Der Aufbau und Ablauf der Applikation, sowie die Nutzung von Hilfsprogrammen wird in den folgenden Abschnitten erklärt. Es werden Hilfsprogramme benötigt, die Schemainformationen und Anfragen dem Programm zur weiteren Auswertung mitteilen.

Der Mehraufwand, der beim Bearbeiten und Auswerten des Workloads entsteht, muss bei der Bewertung des Verfahrens berücksichtigt werden. Die Anfragen müssen in den Mapgraph eingefügt werden, wobei Reorganisationsmaßnahmen, wie Restrukturierung oder Merging von Boxen, nötig werden. Dieser Aufwand muss bei Verwendung der Optimierung berücksichtigt werden, da der Profit der Optimierung nicht durch den Organisationsaufwand wieder wett gemacht werden darf.

Es wird ein Workload erstellt, der durch den Design Advisor und die Konzeptapplikation behandelt wird. Die Ergebnisse werden verglichen und bewertet.

4.2 Programmablauf

Das Konzept dieser Arbeit wurde als Java-Applikation umgesetzt. Dabei wird die Datenbankfunktion komplett emuliert. Die Emulation beschränkt sich nur auf Teilstrukturen des DBMS, die für die Umsetzung des Konzeptes nötig sind. Dabei wurden statistische Zusammenhänge, die den Boxeigenschaften entsprechen und zur Profitberechnung benötigt werden, in Form von Annahmen vereinfacht.

Das Programm erhält als Eingabe eine Anfragemenge, die in das Query Graph Model umgewandelt werden. Es können einzelne Anfragen als Eingabeparameter, sowie auch mehrere Anfragen in einer Datei behandelt werden. Dazu stehen Verwaltungswerkzeuge zur Verfügung, die in Abschnitt 4.3 näher beschrieben werden.

Die Vorschläge zur Materialisierung werden der Standardausgabe, nach Beendigung des *Workload-Emulator* (Abschnitt 4.3.4), als QGM übergeben und können dann im DBMS übernommen werden. Unter realen Umständen wäre das Tool im Optimierer integriert und könnte durch Setzen des zuständigen Registers aktiviert werden.

4.2.1 Kompilierung von SQL in QGM

Da das Programm ein DBMS nur simuliert, muss ein Compiler existieren, der die Anfrage von der SQL-Notation in die QGM-Notation überführt. Normalerweise würde es genügen direkt den Anfrageplan zu überführen. Da die Emulation jedoch keine Anfragepläne verwendet, ist dies nicht möglich.

Der Compiler arbeitet mit regulären Ausdrücken, die einen Teil des SQL-Standards[5] abdecken. Dabei unterstützt er folgende Bereiche des SQL-Standards:

- Projektion
 - mit Aggregatfunktionen SUM, COUNT, MAX, MIN
 - ohne Skalarfunktionen
 - ohne Subselects
- Verbund
 - mit Subselects in beliebiger Verschachtelungstiefe
 - ohne reflexiven Verbund
 - ohne Verbundselektion (Schlüsselwort 'ON')
- Selektion
 - mit Operatoren <, <=, !=, =, >=, >
 - ohne Like, Similar To, Between, ...
- Gruppierung
- Aggregatwerteselektion
 - mit Operatoren <, <=, !=, =, >=, >
 - ohne Subselects
 - Gruppierungsklausel muss vorkommen
 - nur Definition des Aliasnamen der Aggregatfunktion

Nach der Kompilierung der Anfrage wird ihre QGM-Variante dem Mapgraph zum Einfügen übergeben.

4.2.2 Matching der Anfrage

Nachdem die Anfrage von SQL in QGM überführt wurde, wird sie über die *insertQuery*-Funktion des Mapgraphs diesem hinzugefügt. Navigator und Mapgraph wurden zu einer Klasse verschmolzen, da die Programmlogik sehr eng verzahnt ist. Die Einfügefunktion ruft eine Hilfsfunktion zum Navigieren durch den Mapgraph auf und übergibt dabei sukzessiv alle Tabellenboxkombinationen aus Mapgraph und Anfrage. Die Navigationsfunktion matcht daraufhin alle Boxpaare nach dem im Konzept beschriebenen

Ablauf. Dazu werden eventuell Tabellenboxen, die sich noch nicht im Mapgraph befinden, eingefügt. Zum Matching existieren Muster, die nach den Vorlagen der dynamischen Anfragebearbeitung erstellt wurden. Die Boxen des QGM wurden als eigenständige Klasse implementiert, die weitere Hilfsklassen nutzt. Eine Hilfsklasse ist die Statistik der Box, die deren Eigenschaftswerte kapselt. Die Statistik wird nach jeder Änderung von abhängigen Boxen oder der Box selber aktualisiert. Ist der Einfügeprozess einer Anfrage abgeschlossen, folgt die Restrukturierung des Mapgraphs, wobei in dieser Zeit das System die Einfügung weiterer Anfragen blockiert. Nach der Restrukturierung wird die Blockade aufgehoben und die nächste Anfrage kann in den Mapgraph eingefügt werden.

4.2.3 Profitberechnung und Ersetzungsverfahren

Nachdem der Workload abgearbeitet wurde, wird der Mapgraph der Klasse *MapGraphAnalyzer* übergeben. Die Klasse bewertet alle Boxen ohne Tabellenboxen mittels der Profitfunktion. Die Boxen werden in einer Liste nach ihrem Profit sortiert und aus der Liste aussortiert, falls sie nicht in den vorher definierten Cache passen. Beim Aussortieren wird absteigend vorgegangen, so dass profitablere Boxen eher in der Liste verbleiben.

Sollen die materialisierten Sichten vollautomatisch verwaltet werden, so werden die bisher im Cache erhaltenen ASTs durch die in der Profitliste existierenden Sichtrepräsentanten ersetzt. Meistens wird dabei keine vollständige Ersetzung, sondern nur ein teilweiser Austausch bzw. gar keine Änderung des Caches vorgenommen. Dies hängt jedoch von der Dynamik des Workloads ab, da häufige Änderung dessen auch eine häufige Änderung des Caches zur Folge hat.

4.3 Verwaltungstools

Die Programme befinden sich im Verzeichnis */bin* und werden von der Kommandozeile im Textmodus ausgeführt.

4.3.1 Tabellenverwaltung

Es existiert ein Hilfsprogramm *tablemanager*, mit dem sich die Tabellendefinition der Tabellenboxen des Mapgraphs festlegen lassen. Da jede Tabellenbox der Anfrage vorher dem Data-Dictionary[13] bekannt gemacht werden muss, da sonst keine statistischen Berechnungen möglich sind, existiert dieses Tool zum Setzen der Statistikwerte.

Es existieren zwei Methoden eine Tabellenbox zu definieren. Der Aufruf des Befehls ohne Parameter führt dazu, dass eine Liste von möglichen Optionen genannt wird.

```
$ ./bin/tablemanager
Usage: TableCreator <option>
Options :
    create [table name/schema/tupleCount]    Create a new table.
    delete                                     Delete a existing table.
    show                                       Show all tables in Database.
```

Zum Erstellen von Tabellenboxen im Data-Dictionary ist es am einfachsten, den Befehl mit der Option *create* ohne weitere Parameter aufzurufen.

```

$ ./bin/tablemanager create
Press Ctrl+C to exit program.

Please enter the name of the table: Person

Schema [name (size in byte; cardinal number of parameter), vorname ...]:
  name (100;5000), vorname (75;2000)

Amount of tuple (rows): 100000
Table created!

```

Bei diesem Beispiel wurde eine Tabellenbox *Person* erstellt, die als Spaltendefinition die Attribute *name* mit Größe 100 / Kardinalität 5000 und *vorname* mit Größe 75 / Kardinalität 2000 hat. Insgesamt existieren 100000 Tupel in dieser Relation.

Weiterhin ist es möglich Tabellen durch einen Parameter direkt zu spezifizieren. Dies erleichtert die Definition von Tabellenboxen bei einer großen Anzahl von Tabellen. Das Trennzeichen zwischen dem Namen der Tabelle, dem Schema und der Tupelanzahl ist ein `/`.

```

$ ./bin/tablemanager create "Person/name(100;5000),vorname(75;2000)/100000"

```

Hier wurde die gleiche Tabelle erzeugt, wie beim vorherigen Beispiel. Falls eine Tabelle bereits existiert, wird eine Exception geworfen.

```

$ ./bin/tablemanager create "Person/name(100;5000),vorname(75;2000)/100000"
db.TableAlreadyExistsException: Relation with name "Person" already exists!
  at db.DataDictionary.setTableBox(DataDictionary.java:82)
  at shell.TableManager.createTable(TableManager.java:74)
  at shell.TableManager.main(TableManager.java:58)

```

Mit der Option *show* werden alle definierten Tabellen angezeigt. So kann eine bereits getroffene Definition einer Tabellenbox analysiert werden.

```

$ ./bin/tablemanager show
T::Person
-> name (100;5000)
-> vorname (75;2000)
TupSize=175
TupCount=100000
RelSize=17.5M

```

4.3.2 Darstellung des MapGraphs

Zum Kontrollieren der korrekten Abbildung des Mapgraphs existiert ein Verwaltungstool *viewmapgraph*, welches die Boxen mit Eigenschaftswerte verknüpft darstellt. Dieses Programm soll prinzipiell nur dem Debugging von Fehlern dienen und spielt daher für den praktischen Einsatz keine Rolle.

4.3.3 Absetzen von Anfragen

Das Einfügen von Anfragen in den Mapgraph erfolgt mit dem Werkzeug *insertquery*. Das Programm erwartet als Parameter eine SQL-Anfrage in Anführungszeichen. Das folgende Beispiel fragt alle Namen der Tabelle *Person* ab.

```
$ ./bin/insertquery "select name from Person"
```

Existiert die Tabelle *Person* nicht oder ist die Syntax nicht konform mit dem Gültigkeitsbereich des Emulators, wird eine Exception geworfen, die den Fehler beschreibt.

Die Anfrage wird, wie in Abschnitt 4.2.1 beschrieben, kompiliert und dem Mapgraph zum Matching übergeben. Damit ist sie im Mapgraph abgebildet.

4.3.4 Workload-Emulator

Um nicht alle Anfragen des Workloads einzeln mit dem *insertQuery*-Tool einfügen zu müssen, existiert ein weiteres Werkzeug *workloademulator*. Dieses nutzt die Textdatei *datadictionary.workload* im Verzeichnis *data*, um dort die vorher definierten Anfragen auszulesen und zum *insertQuery*-Tool umzuleiten. Die Anfragen werden durch das Zeichen *;* voneinander getrennt, so dass das Tool die Anfragen eindeutig voneinander abgrenzen kann.

Im folgenden Beispiel werden zwei Anfragen dargestellt, die nacheinander mit dem *insertQuery*-Tool aufgerufen werden.

```
SELECT
    SUM(L_EXTENDEDPRICE) AS REVENUE
FROM LINEITEM
WHERE L_DISCOUNT > 0.06
    AND L_QUANTITY < 24;

SELECT
    SUPP_NATION,
    SUM(VOLUME) AS REVENUE
FROM
    (
        SELECT
            N_NAME AS SUPP_NATION,
            L_EXTENDEDPRICE AS VOLUME
        FROM SUPPLIER,
            LINEITEM,
            ORDERS,
            CUSTOMER,
            NATION
        WHERE S_SUPPKEY = L_SUPPKEY
            AND O_ORDERKEY = L_ORDERKEY
            AND C_CUSTKEY = O_CUSTKEY
            AND S_NATIONKEY = N_NATIONKEY
            AND N_NAME = 'FRANCE'
    ) AS SHIPPING
GROUP
    BY SUPP_NATION;
```

Es ist zusätzlich möglich verschiedene Anfragen auszukommentieren, so dass sie für den Einfügeprozess nicht in Frage kommen. Eine Anfrage wird mit der Zeichenkette *//* als Präfix auskommentiert.

Das folgende Listing zeigt die beiden Anfragen aus dem vorherigen Beispiel, wobei die zweite Anfrage auskommentiert wurde und somit nicht dem Einfügeprozess unterliegt.

```
SELECT
    SUM(L_EXTENDEDPRICE) AS REVENUE
FROM LINEITEM
WHERE L_DISCOUNT > 0.06
    AND L_QUANTITY < 24;

SELECT
```

```

SUPP_NATION ,
SUM(VOLUME) AS REVENUE
//FROM
(
SELECT
  N_NAME AS SUPP_NATION ,
  L_EXTENDEDPRICE AS VOLUME
FROM SUPPLIER ,
  LINEITEM ,
  ORDERS ,
  CUSTOMER ,
  NATION
WHERE S_SUPPKEY = L_SUPPKEY
  AND O_ORDERKEY = L_ORDERKEY
  AND C_CUSTKEY = O_CUSTKEY
  AND S_NATIONKEY = N_NATIONKEY
  AND N_NAME = 'FRANCE'
) AS SHIPPING
GROUP
BY SUPP_NATION ;

```

4.4 Overhead des Verfahrens

In diesem Abschnitt soll diskutiert werden, wie groß der Mehraufwand durch das Navigieren und Matchen im Mapgraph ist. Die Optimierung ist nur dann erfolgreich, falls der Profit ihres Ergebnisses größer ist, als der Aufwand, der durch sie entsteht. Dabei müssen die Größe, der Traversieraufwand beim Einfügen und der Aufwand beim Matching in einer Boxebene des Mapgraphs berücksichtigt werden.

4.4.1 Größe des Mapgraphs

Die Größe des Mapgraphs hängt stark vom verwendeten Workload ab. Dabei spielt weniger die Anzahl, als mehr die Ähnlichkeiten der Anfragen eine Rolle. Wie im Konzept beschrieben, werden zueinander ähnliche Anfragen auf die selben Boxen abgebildet. Die Unähnlichkeit wird sich dann nur in den oberen Ebenen des Mapgraphs, also den Elternboxen der gemeinsamen Boxen, bemerkbar machen. Somit wird für eine Anfrage, die aus z. B. vier Boxen besteht, im Mapgraph nur eine Box, nämlich die Kompensation, erstellt.

Die Größe des Mapgraphs spielt beim Aufwand eine entscheidene Rolle. Der Aufwand der Navigation hängt davon ab, wieviele Boxen zum Matching besucht werden müssen. Daher ist der Aufwand des Matchings und der Optimierungen zur Größe des Mapgraph proportional.

4.4.2 Traversierung des Mapgraphs

Bei der Navigation durch den Mapgraph wird immer ein Teilgraph durchlaufen. Die Navigation beginnt bei den Tabellenboxen und verzweigt sich dann in die Richtung, in der die zur Anfrage passenden Boxen gematcht wurden. Der Aufwand entspricht somit der eines Baumes, da jede Tabellenbox als Wurzel angenommen werden kann, von der aus Kanten zu weiteren Teilästen verlaufen. Der sich daraus ergebene Aufwand für das Traversieren, ausgehend von einer Tabellenbox, wäre $\log_{m_{MG}}(n_{MG})$, wobei m_{MG} der Ordnung des Baumes und n_{MG} der Boxen im Baum des Mapgraphs entspricht.

Der Aufwand summiert sich für jede im Mapgraph befindliche Tabellenbox. Damit ergibt sich aus der Anzahl der Tabellenboxen des Mapgraphs $\#_{TB}(MG)$ und dem Aufwand des Traversierens jedes Baumes der maximale Gesamtaufwand $O(\#_{TB}(MG) \times \log_{m_{MG}}(n_{MG}))$.

4.4.3 Matching

Beim Traversieren der Anfrage und des Mapgraphs bei Einfüge-, Restrukturierungs- oder Mergingoperationen werden Paare aus allen Elternboxen einer Box gebildet und zur Matchfunktion geleitet. Der Aufwand, der dabei entsteht, ist höchstens quadratisch, also $O(n^2)$, wobei n der maximalen Anzahl an Elternboxen entspricht.

4.5 Stresstest (TPC-H)

TPC-H [17] (Transaction Processing Performance Council) ist ein entscheidungsunterstützender Benchmark zum Testen von relationalen Datenbankverwaltungssystemen. Die Ergebnisse des Benchmarks können für verschiedene Hardware-Datenbankkompositionen käuflich erworben werden.

Die enthaltene Datenbank besteht aus verschiedenen Tabellen, die eine Abbildung einer Geschäftsdatenbank widerspiegeln. So kann anhand von realen Bedingungen die Leistung des DBMS bzw. der genutzten Hardware getestet werden.

Es werden zwei Werkzeuge mitgeliefert, die zur Generierung genutzt werden können. Das Programm *dbgen* dient zum Erzeugen eines Datenbestandes, der in die Datenbank durch einen Load importiert werden kann. Dabei können verschiedene Parameter, z. B. zur Angabe der Größe der Datenmenge angegeben werden. Der folgende Befehl erzeugt eine Datenmenge der Größe von einem Gigabyte.

```
$ dbgen -s 1
```

Es existieren Anfragevorlagen, die durch das zweite Tool *qgen* in Anfragen, welche Teilrelation der durch *dbgen* erzeugten Datenmenge selektieren, umgewandelt werden. Die Operationszeit wird vom Benchmark-Tool gemessen und dient als Vergleich für verschiedene Systeme. Zum Erzeugen einer Anfrage wird *qgen* mit einem Parameter aufgerufen, der dem Namen der Anfragevorlage entspricht.

```
$ qgen 1
```

Bei diesem Beispiel wird die Vorlage aus Datei *1.sql* des *queries*-Verzeichnisses in die entsprechende Anfrage mit korrekten Prädikatwerten umgewandelt und an die Standardausgabe geleitet. Es können zusätzlich noch weitere Parameteroptionen gesetzt werden, die die Ausgabe von *qgen* modifizieren.

4.5.1 Workloadauswahl

Der von *qgen* generierte Workload ist dafür bestimmt, das DBMS auf ihre Antwortzeit zu untersuchen. Es werden einzelne Anfragen generiert, deren Laufzeit gemessen wird. Dies dient als Vergleich für andere DBMS, deren Geschwindigkeit getestet werden soll.

Die 22 Anfragen, die von TPC-H zum Testen mitgeliefert werden, reichen jedoch nicht aus, um ein realistisches Szenario unter Verwendung von materialisierten Sichten zu testen. Hält man sich in einem Wirtschaftsunternehmen allein nur den Kundenstamm vor Augen, besteht dieser schon aus mehreren tausend Kunden. Auf jeden Kunden werden Anfragen gestellt, die das System verarbeiten soll. Daraus ergeben sich sehr viele verschiedene Punkt- und Bereichsanfragen, die das System beantworten muss.

Um die automatische Erstellung von ASTs zu testen, ist es somit notwendig, eine Menge von Punkt- und Bereichsanfragen vom TPC-H-Workload abzuleiten. Es werden acht repräsentative Anfragen aus dem TPC-H Workload ausgewählt, wobei aus einer Anfrage mehrere Punkt- und Bereichsanfragen abgeleitet werden. Die Begrenzung der Anzahl von Repräsentanten ist darin begründet, dass die anderen Anfragen kaum andere Ergebnismengen darstellen, sondern nur skalare und algebraische Funktionen testen, die für den Test der AST-Auswahl keine Rolle spielen.

Die von TPC-H abgeleitete Anfrage wird so modifiziert, dass sie sich in einem oder mehreren Prädikaten unterscheiden, dabei jedoch noch eine gemeinsame Prädikatschnittmenge existiert. Damit wird sichergestellt, dass die Restrukturierungseigenschaft des Navigator's genutzt wird.

Die Anfragen der Abbildungen 4.1 bis 4.7 stellen die Anfragen von TPC-H dar. Die Anfrage aus Abbildung 4.8 ist eine modifizierte Form aus dem TPC-H Workload. Sie entspricht einer Obermengenrelation der Anfragen aus den Abbildungen 4.9, 4.10, 4.11, 4.12 und 4.13, die untereinander wiederum Teilrelationen bilden.

Verwendet man die Anfragen ohne die Nutzung von materialisierten Sichten, so beträgt die Laufzeit der Anfragebeantwortung insgesamt 437 Sekunden. Alle Messdaten wurden auf dem gleichen Rechner (AMD Athlon 1700XP, 1GB-RAM) ausgeführt. Eine detaillierte Darstellung der Ausführungszeiten der Anfragen ist in Tabelle 4.1 aufgelistet.

```
SELECT
  l_returnflag, l_linestatus, sum(l_quantity) AS sum_qty,
  sum(l_extendedprice) AS sum_base_price, max(l_discount) AS max_disc,
  count(*) AS count_order
FROM lineitem
WHERE l_shipdate < '1998-12-01'
GROUP
  BY l_returnflag, l_linestatus;
```

Abbildung 4.1: TPC-H Query 1

```
SELECT
  s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey
  AND s_suppkey = ps_suppkey
  AND p_size = 15
  AND s_nationkey = n_nationkey
  AND n_regionkey = r_regionkey
  AND r_name = 'EUROPE';
```

Abbildung 4.2: TPC-H Query 2

```

SELECT
  l_orderkey, sum(l_extendedprice) AS revenue, o_orderdate, o_shippriority
FROM customer, orders, lineitem
WHERE c_mktsegment = 'BUILDING'
  AND c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate < '1995-03-15'
  AND l_shipdate > '1995-03-15'
GROUP
  BY l_orderkey, o_orderdate, o_shippriority;

```

Abbildung 4.3: TPC-H Query 3

```

SELECT
  n_name, sum(l_extendedprice) AS revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND l_suppkey = s_suppkey
  AND c_nationkey = s_nationkey
  AND s_nationkey = n_nationkey
  AND n_regionkey = r_regionkey
  AND r_name = 'ASIA'
GROUP
  BY n_name;

```

Abbildung 4.4: TPC-H Query 4

```

SELECT
  sum(l_extendedprice) AS revenue
FROM lineitem
WHERE l_discount > 0.06
  AND l_quantity < 24;

```

Abbildung 4.5: TPC-H Query 5

```

SELECT
  supp_nation, sum(volume) AS revenue
FROM
  (
    SELECT
      n_name AS supp_nation, l_extendedprice AS volume
    FROM supplier, lineitem, orders, customer, nation
    WHERE s_suppkey = l_suppkey
      AND o_orderkey = l_orderkey
      AND c_custkey = o_custkey
      AND s_nationkey = n_nationkey
      AND n_name = 'FRANCE'
  ) AS shipping
GROUP
  BY supp_nation;

```

Abbildung 4.6: TPC-H Query 6

```

SELECT
  o_year, sum(volume) AS mkt_share
FROM
  (
    SELECT
      o_orderdate AS o_year, l_extendedprice AS volume, n_name AS nation
    FROM part, supplier, lineitem, orders, customer, nation, region
    WHERE p_partkey = l_partkey
      AND s_suppkey = l_suppkey
      AND l_orderkey = o_orderkey
      AND o_custkey = c_custkey
      AND c_nationkey = n_nationkey
      AND n_regionkey = r_regionkey
      AND p_type = 'ECONOMY ANODIZED STEEL'
    ) AS all_nations
GROUP
  BY o_year;

```

Abbildung 4.7: TPC-H Query 7

```

SELECT
  c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate BETWEEN '1995-01-01' AND '1995-03-31'
  AND l_returnflag = 'R'
  AND c_nationkey = n_nationkey
GROUP
  BY c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate;

```

Abbildung 4.8: TPC-H Query 8

```

SELECT
  c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate BETWEEN '1995-01-01' AND '1995-01-31'
  AND l_returnflag = 'R'
  AND c_nationkey = n_nationkey
GROUP
  BY c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate;

```

Abbildung 4.9: TPC-H Query 8-1

```

SELECT
  c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate BETWEEN '1995-01-01' AND '1995-02-31'
  AND l_returnflag = 'R'
  AND c_nationkey = n_nationkey
GROUP
  BY c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate;

```

Abbildung 4.10: TPC-H Query 8-2

```
SELECT
  c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
      AND l_orderkey = o_orderkey
      AND o_orderdate BETWEEN '1995-02-01' AND '1995-03-31'
      AND l_returnflag = 'R'
      AND c_nationkey = n_nationkey
GROUP
  BY c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate;
```

Abbildung 4.11: TPC-H Query 8-3

```
SELECT
  c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
      AND l_orderkey = o_orderkey
      AND o_orderdate BETWEEN '1995-03-01' AND '1995-03-31'
      AND l_returnflag = 'R'
      AND c_nationkey = n_nationkey
GROUP
  BY c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate;
```

Abbildung 4.12: TPC-H Query 8-4

```
SELECT
  c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey
      AND l_orderkey = o_orderkey
      AND o_orderdate BETWEEN '1995-02-01' AND '1995-02-31'
      AND l_returnflag = 'R'
      AND c_nationkey = n_nationkey
GROUP
  BY c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate;
```

Abbildung 4.13: TPC-H Query 8-5

Anfrage	Laufzeit in Sekunden
TPC-H Query 1 (Abb. 4.1)	62
TPC-H Query 2 (Abb. 4.2)	2
TPC-H Query 3 (Abb. 4.3)	39
TPC-H Query 4 (Abb. 4.4)	39
TPC-H Query 5 (Abb. 4.5)	32
TPC-H Query 6 (Abb. 4.6)	33
TPC-H Query 7 (Abb. 4.7)	58
TPC-H Query 8 (Abb. 4.8)	42
TPC-H Query 8-1 (Abb. 4.9)	24
TPC-H Query 8-2 (Abb. 4.10)	33
TPC-H Query 8-3 (Abb. 4.11)	34
TPC-H Query 8-4 (Abb. 4.12)	22
TPC-H Query 8-5 (Abb. 4.13)	17

Tabelle 4.1: Laufzeit des TPC-H Workloads ohne Verwendung von ASTs

4.5.2 Ergebnisse DB2 Design Advisor

Bei Verwendung der beschriebenen Anfragen als Workload schlägt der Design Advisor mehrere Definitionen zur Materialisierung vor. Die Abbildungen 4.14, 4.15, 4.16, 4.17, 4.18, 4.19 stellen die Erstellungsstatements dar. Alle Definitionen entsprechen Obermengen des Workloads.

Nach Erstellung der ASTs aus diesen Definitionen, wird die Ausführungszeit des Workloads erneut gemessen. Dazu ist es notwendig, vorher das *CURRENT REFRESH AGE*-Register des DBMS auf den Wert *ANY* zu setzen. Dadurch wird dem System erlaubt, entsprechende Anfragebereiche automatisch durch eine materialisierte Sicht zu ersetzen. Die Laufzeit des Workloads bei Benutzung der erstellten ASTs beträgt nur noch ca. 67 Sekunden. Damit zwar deutlich weniger, als keine Verwendung der ASTs, jedoch immer noch sehr aufwendig zu berechnen. Tabelle 4.2 gibt einen ausführlichen Überblick über die Laufzeiten der Workloadanfragen.

Anfrage	Laufzeit in Sekunden
TPC-H Query 1 (Abb. 4.1)	0.044
TPC-H Query 2 (Abb. 4.2)	0.374
TPC-H Query 3 (Abb. 4.3)	~ 10
TPC-H Query 4 (Abb. 4.4)	0.018
TPC-H Query 5 (Abb. 4.5)	0.019
TPC-H Query 6 (Abb. 4.6)	0.017
TPC-H Query 7 (Abb. 4.7)	0.907
TPC-H Query 8 (Abb. 4.8)	~ 10
TPC-H Query 8-1 (Abb. 4.9)	~ 9
TPC-H Query 8-2 (Abb. 4.10)	~ 9
TPC-H Query 8-3 (Abb. 4.11)	~ 9
TPC-H Query 8-4 (Abb. 4.12)	~ 9
TPC-H Query 8-5 (Abb. 4.13)	~ 10

Tabelle 4.2: Laufzeit des TPC-H Workloads bei Verwendung der DB2-ASTs

```

CREATE
  SUMMARY TABLE "DB2INST1"."MQT708211552390000" AS
  (
  SELECT
    Q8.CO AS "CO", Q8.C1 AS "C1", Q8.C2 AS "C2"
  FROM TABLE
    (
    SELECT
      Q7.CO AS "CO", SUM(Q7.C1) AS "C1", Q7.C2 AS "C2"
    FROM TABLE
      (
      SELECT
        Q2.N_NAME AS "CO", Q4.L_EXTENDEDPRICE AS "C1", Q1.R_NAME AS "C2"
      FROM DB2INST1.REGION AS Q1, DB2INST1.NATION AS Q2, DB2INST1.SUPPLIER AS Q3,
        DB2INST1.LINEITEM AS Q4, DB2INST1.ORDERS AS Q5, DB2INST1.CUSTOMER AS Q6
      WHERE
        (
          Q2.N_REGIONKEY = Q1.R_REGIONKEY
        )
        AND
        (
          Q3.S_NATIONKEY = Q2.N_NATIONKEY
        )
        AND
        (
          Q6.C_NATIONKEY = Q3.S_NATIONKEY
        )
        AND
        (
          Q4.L_SUPPKEY = Q3.S_SUPPKEY
        )
        AND
        (
          Q4.L_ORDERKEY = Q5.O_ORDERKEY
        )
        AND
        (
          Q6.C_CUSTKEY = Q5.O_CUSTKEY
        )
      ) AS Q7
    GROUP
      BY Q7.C2, Q7.CO
    ) AS Q8
  )
  DATA INITIALLY DEFERRED REFRESH DEFERRED IN USERSPACE1 ;

```

Abbildung 4.14: DB2 - AST-Vorschlag-1

```
CREATE
SUMMARY TABLE "DB2INST1"."MQT708211552400000" AS
(
SELECT
  Q3.C0 AS "C0", Q3.C2 AS "C1", Q3.C1 AS "C2"
FROM TABLE
(
  SELECT
    SUM(Q2.C0) AS "C0", Q2.C2 AS "C1", Q2.C1 AS "C2"
  FROM TABLE
  (
    SELECT
      Q1.L_EXTENDEDPRI AS "C0", Q1.L_QUANTITY AS "C1",
      Q1.L_DISCOUNT AS "C2"
    FROM DB2INST1.LINEITEM AS Q1
  ) AS Q2
  GROUP
    BY Q2.C1, Q2.C2
  ) AS Q3
)
DATA INITIALLY DEFERRED REFRESH DEFERRED IN USERSPACE1 ;
```

Abbildung 4.15: DB2 - AST-Vorschlag-2

```

CREATE
SUMMARY TABLE "DB2INST1"."MQT708211552410000" AS
(
SELECT
  Q8.CO AS "C0", Q8.C1 AS "C1"
FROM TABLE
  (
  SELECT
    Q7.CO AS "C0", SUM(Q7.C1) AS "C1"
  FROM TABLE
    (
    SELECT
      Q6.CO AS "C0", Q6.C1 AS "C1"
    FROM TABLE
      (
      SELECT
        Q1.N_NAME AS "C0", Q4.L_EXTENDEDPRICE AS "C1"
      FROM DB2INST1.NATION AS Q1, DB2INST1.CUSTOMER AS Q2,
        DB2INST1.ORDERS AS Q3, DB2INST1.LINEITEM AS Q4,
        DB2INST1.SUPPLIER AS Q5
      WHERE
        (
          Q5.S_NATIONKEY = Q1.N_NATIONKEY
        )
        AND
        (
          Q2.C_CUSTKEY = Q3.O_CUSTKEY
        )
        AND
        (
          Q3.O_ORDERKEY = Q4.L_ORDERKEY
        )
        AND
        (
          Q5.S_SUPPKEY = Q4.L_SUPPKEY
        )
      ) AS Q6
    ) AS Q7
  GROUP
    BY Q7.CO
  ) AS Q8
)
DATA INITIALLY DEFERRED REFRESH DEFERRED IN USERSPACE1 ;

```

Abbildung 4.16: DB2 - AST-Vorschlag-3

```

CREATE
SUMMARY TABLE "DB2INST1"."MQT708211552420000" AS
(
SELECT
  Q10.CO AS "C0", Q10.C1 AS "C1", Q10.C2 AS "C2"
FROM TABLE
(
  SELECT
    Q9.CO AS "C0", SUM(Q9.C1) AS "C1", Q9.C2 AS "C2"
  FROM TABLE
  (
    SELECT
      Q8.CO AS "C0", Q8.C1 AS "C1", Q8.C3 AS "C2"
    FROM TABLE
    (
      SELECT
        Q4.O_ORDERDATE AS "C0", Q5.L_EXTENDEDPRICE AS "C1",
        Q2.N_NAME AS "C2", Q7.P_TYPE AS "C3"
      FROM DB2INST1.REGION AS Q1, DB2INST1.NATION AS Q2,
        DB2INST1.CUSTOMER AS Q3, DB2INST1.ORDERS AS Q4,
        DB2INST1.LINEITEM AS Q5, DB2INST1.SUPPLIER AS Q6,
        DB2INST1.PART AS Q7
      WHERE
        (
          Q2.N_REGIONKEY = Q1.R_REGIONKEY
        )
        AND
        (
          Q3.C_NATIONKEY = Q2.N_NATIONKEY
        )
        AND
        (
          Q4.O_CUSTKEY = Q3.C_CUSTKEY
        )
        AND
        (
          Q5.L_ORDERKEY = Q4.O_ORDERKEY
        )
        AND
        (
          Q6.S_SUPPKEY = Q5.L_SUPPKEY
        )
        AND
        (
          Q7.P_PARTKEY = Q5.L_PARTKEY
        )
      ) AS Q8
    ) AS Q9
  ) GROUP
  BY Q9.C2, Q9.CO
  ) AS Q10
)
DATA INITIALLY DEFERRED REFRESH DEFERRED IN USERSPACE1 ;

```

Abbildung 4.17: DB2 - AST-Vorschlag-4

```

CREATE
SUMMARY TABLE "DB2INST1"."MQT708211553000000" AS
(
SELECT
  Q4.C0 AS "C0", Q4.C1 AS "C1", Q4.C3 AS "C2",
  Q4.C4 AS "C3", Q4.C6 AS "C4", Q4.C7 AS "C5",
  Q4.C8 AS "C6", Q4.C2 AS "C7", Q4.C5 AS "C8"
FROM TABLE
(
SELECT
  Q3.C2 AS "C0", Q3.C3 AS "C1", Q3.C0 AS "C2",
  SUM(Q3.C0) AS "C3", SUM(Q3.C5) AS "C4", Q3.C1 AS "C5",
  MAX(Q3.C1) AS "C6", COUNT(*) AS "C7", Q3.C4 AS "C8"
FROM TABLE
(
SELECT
  Q2.C3 AS "C0", Q2.C5 AS "C1", Q2.C2 AS "C2",
  Q2.C1 AS "C3", Q2.C0 AS "C4", Q2.C4 AS "C5"
FROM TABLE
(
SELECT
  Q1.L_SHIPDATE AS "C0", Q1.L_LINESTATUS AS "C1",
  Q1.L_RETURNFLAG AS "C2", Q1.L_QUANTITY AS "C3",
  Q1.L_EXTENDEDPRI AS "C4", Q1.L_DISCOUNT AS "C5"
FROM DB2INST1.LINEITEM AS Q1
) AS Q2
) AS Q3
GROUP
  BY GROUPING SETS((Q3.C0,Q3.C1),(Q3.C2,Q3.C3,Q3.C4))
) AS Q4
)
DATA INITIALLY DEFERRED REFRESH DEFERRED IN USERSPACE1 ;

```

Abbildung 4.18: DB2 - AST-Vorschlag-5

```

CREATE
SUMMARY TABLE "DB2INST1"."MQT708211553020000" AS
(
SELECT
  Q3.L_SHIPDATE AS "C0", Q1.C_MKTSEGMENT AS "C1",
  Q2.O_SHIPPRIORITY AS "C2", Q2.O_ORDERDATE AS "C3",
  Q3.L_ORDERKEY AS "C4", Q3.L_EXTENDEDPRI AS "C5",
  Q3.L_DISCOUNT AS "C6", Q3.L_QUANTITY AS "C7",
  Q3.L_RETURNFLAG AS "C8", Q3.L_LINESTATUS AS "C9",
  Q1.C_CUSTKEY AS "C10", Q2.O_CUSTKEY AS "C11",
  Q2.O_ORDERKEY AS "C12"
FROM DB2INST1.CUSTOMER AS Q1, DB2INST1.ORDERS AS Q2,
  DB2INST1.LINEITEM AS Q3
WHERE
(
  Q1.C_CUSTKEY = Q2.O_CUSTKEY
)
AND
(
  Q3.L_ORDERKEY = Q2.O_ORDERKEY
)
)
DATA INITIALLY DEFERRED REFRESH DEFERRED IN USERSPACE1 ;

```

Abbildung 4.19: DB2 - AST-Vorschlag-6

4.5.3 Ergebnisse Konzept-Applikation

Die Konzeptapplikation verarbeitet den gleichen Workload, wie der DB2 Design Advisor. Im Folgenden werden die Vorschläge für die Erstellung von materialisierten Sichten vorgestellt. Nach der Materialisierung der Vorschläge benötigt die Anfragebearbeitungszeit des Workloads nur noch ca. 5 Sekunden, bei gleicher Ausnutzung des zur Verfügung stehenden Caches wie der Design Advisor. Die Laufzeit des Workloads bei Ausnutzung der von der Konzeptapplikation vorgeschlagenen ASTs ist in Tabelle 4.3 dargestellt.

Anfrage	Laufzeit in Sekunden
TPC-H Query 1 (Abb. 4.1)	0.018
TPC-H Query 2 (Abb. 4.2)	0.022
TPC-H Query 3 (Abb. 4.3)	0.029
TPC-H Query 4 (Abb. 4.4)	0.017
TPC-H Query 5 (Abb. 4.5)	0.017
TPC-H Query 6 (Abb. 4.6)	0.017
TPC-H Query 7 (Abb. 4.7)	0.022
TPC-H Query 8 (Abb. 4.8)	1.125 (davon Gruppierungop. 0.981 Sekunden)
TPC-H Query 8-1 (Abb. 4.9)	0.541
TPC-H Query 8-2 (Abb. 4.10)	1.068
TPC-H Query 8-3 (Abb. 4.11)	0.749
TPC-H Query 8-4 (Abb. 4.12)	0.417
TPC-H Query 8-5 (Abb. 4.13)	0.476

Tabelle 4.3: Laufzeit des TPC-H Workloads bei Verwendung der Konzept-ASTs

```

CREATE
  summary table MQT_01 AS
  (
  SELECT
    o_year, sum(volume) AS mkt_share
  FROM
    (
    SELECT
      o_orderdate AS o_year, l_extendedprice AS volume, n_name AS nation
    FROM part, supplier, lineitem, orders, customer, nation, region
    WHERE p_partkey = l_partkey
      AND s_suppkey = l_suppkey
      AND l_orderkey = o_orderkey
      AND o_custkey = c_custkey
      AND c_nationkey = n_nationkey
      AND n_regionkey = r_regionkey
      AND p_type = 'ECONOMY ANODIZED STEEL'
    ) AS all_nations
  GROUP
    BY o_year
  )
  DATA INITIALLY DEFERRED REFRESH DEFERRED ;

```

Abbildung 4.20: Konzeptapplikation - AST-Vorschlag-1

```

CREATE
  summary table MQT_02 AS
  (
  SELECT
    s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
  FROM part, supplier, partsupp, nation, region
  WHERE p_partkey = ps_partkey
    AND s_suppkey = ps_suppkey
    AND p_size = 15
    AND s_nationkey = n_nationkey
    AND n_regionkey = r_regionkey
    AND r_name = 'EUROPE'
  )
  DATA INITIALLY DEFERRED REFRESH DEFERRED ;

```

Abbildung 4.21: Konzeptapplikation - AST-Vorschlag-2

```

CREATE
  summary table MQT_03 AS
  (
  SELECT
    l_returnflag, l_linestatus, sum(l_quantity) AS sum_qty,
    sum(l_extendedprice) AS sum_base_price, max(l_discount) AS max_disc,
    count(*) AS count_order
  FROM lineitem
  WHERE l_shipdate < '1998-12-01'
  GROUP
    BY l_returnflag, l_linestatus
  )
  DATA INITIALLY DEFERRED REFRESH DEFERRED ;

```

Abbildung 4.22: Konzeptapplikation - AST-Vorschlag-3

```
CREATE
summary table MQT_04 AS
(
SELECT
    sum(l_extendedprice) AS revenue
FROM lineitem
WHERE l_discount > 0.06
    AND l_quantity < 24
)
DATA INITIALLY DEFERRED REFRESH DEFERRED ;
```

Abbildung 4.23: Konzeptapplikation - AST-Vorschlag-4

```
CREATE
summary table MQT_05 AS
(
SELECT
    supp_nation, sum(volume) AS revenue
FROM
(
SELECT
    n_name AS supp_nation, l_extendedprice AS volume
FROM supplier, lineitem, orders, customer, nation
WHERE s_suppkey = l_suppkey
    AND o_orderkey = l_orderkey
    AND c_custkey = o_custkey
    AND s_nationkey = n_nationkey
    AND n_name = 'FRANCE'
) AS shipping
GROUP
    BY supp_nation
)
DATA INITIALLY DEFERRED REFRESH DEFERRED ;
```

Abbildung 4.24: Konzeptapplikation - AST-Vorschlag-5

```
CREATE
summary table MQT_06 AS
(
SELECT
    n_name, sum(l_extendedprice) AS revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND l_suppkey = s_suppkey
    AND c_nationkey = s_nationkey
    AND s_nationkey = n_nationkey
    AND n_regionkey = r_regionkey
    AND r_name = 'ASIA'
GROUP
    BY n_name
)
DATA INITIALLY DEFERRED REFRESH DEFERRED ;
```

Abbildung 4.25: Konzeptapplikation - AST-Vorschlag-6

```
CREATE
  summary table MQT_07 AS
  (
  SELECT
    l_orderkey, sum(l_extendedprice) AS revenue, o_orderdate, o_shippriority
  FROM customer, orders, lineitem
  WHERE c_mktsegment = 'BUILDING'
    AND c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND o_orderdate < '1995-03-15'
    AND l_shipdate > '1995-03-15'
  GROUP
    BY l_orderkey, o_orderdate, o_shippriority
  )
  DATA INITIALLY DEFERRED REFRESH DEFERRED ;
```

Abbildung 4.26: Konzeptapplikation - AST-Vorschlag-7

```
CREATE
  summary table MQT_08 AS
  (
  SELECT
    c_custkey, n_name, c_address, c_phone, c_comment, o_orderdate
  FROM customer, orders, lineitem, nation
  WHERE c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND o_orderdate >= '1995-01-01'
    AND o_orderdate <= '1995-03-31'
    AND l_returnflag = 'R'
    AND c_nationkey = n_nationkey
  )
  DATA INITIALLY DEFERRED REFRESH DEFERRED ;
```

Abbildung 4.27: Konzeptapplikation - AST-Vorschlag-8

4.6 Vorhersagequalität

Die Verbesserung der Anfrageantwortdauer bei Nutzung der von der Konzeptapplikation vorgeschlagenen ASTs erscheint auf den ersten Blick sehr radikal. Die Ursache liegt in der viel spezielleren Sichtdefinition, bei der die Kompensation der Anfragebeantwortung weniger aufwendig zu berechnen ist. Jedoch sei hier zu beachten, dass die von DB2 vorgeschlagene AST-Definition keinesfalls als schlecht zu bewerten ist. Die Definitionen werden vom Design Advisor sehr allgemein vorgeschlagen, was unter realen Umständen eine durchschnittliche Verbesserung liefern könnte. Kritisch dabei ist, dass die allgemein definierten Sichten für spezielle und stark frequentierte Workloadanfragen nur mittelmäßige Verbesserung bringen. Dieser Sachverhalt wird in den folgenden zwei Unterabschnitten näher betrachtet.

4.6.1 Workload mit vielen Punktanfragen

Wird eine Datenbank mit vielen Punktanfragen abgefragt, in denen Gruppen von Anfragen existieren, die eine gemeinsame Prädikatschnittmenge haben, so ist der Einsatz von allgemeinen Sichten, die genau diese Prädikate als Definition enthalten, sehr zu empfehlen. Existiert eine solche Sicht, können die vielen Punktanfragen von der allgemeinen Definition Gebrauch machen und somit ihren Operationsaufwand verbessern.

4.6.2 Workload mit wenigen Punktanfragen

Existiert ein Workload der kaum Gruppen von gemeinsamen Prädikatschnittmengen bildet, so kann eine allgemeine Sichtdefinition zwar Vorteile bei der Anfragebeantwortung haben, jedoch die speziellen Anforderungen der Anfragen nicht befriedigen. So müssen die Prädikate und Gruppierungen, die nicht durch die allgemeine Sicht abgedeckt wurden, durch die Kompensation sichergestellt werden, was insgesamt die Anfragedauer stärker belastet, als eine speziellere Sichtdefinition. Weiterhin verschlechternd ist die umfangreichere Größe der allgemeinen Definition. Eine speziellere Definition nutzt weitere Prädikate und eventuell Gruppierungen, die eine Ergebnissicht deutlich in der Größe reduzieren.

4.6.3 Vergleich der Verfahren

Der Design Advisor von DB2 nutzt beim vorliegenden Workload nur die Verbundprädikate beim Erstellen der Sichtdefinition. Dies ist ein geeignetes Mittel, um die Anfragedauer zu beschleunigen, jedoch nutzt er nicht den kompletten Rahmen an Möglichkeiten. Dies fällt besonders bei der Anfrage des Workloads in Abbildung 4.8 auf. Der Design Advisor verzichtet komplett auf die Verwendung des Datumbereiches beim Vorschlagen einer geeigneten AST, obwohl der Workload nur Teilmengen dieser Relation abfragt.

Die Anfragen aus den Abbildungen 4.10, 4.11, 4.12 und 4.13 matchen Teilmengen der Anfrage aus Abbildung 4.8 und positivieren somit die Materialisierungsentscheidung dieser allgemeinen Anfrage. Bei der Konzeptapplikation führt dies zum Vorschlag dieser

Anfrage, jedoch ohne Gruppierung¹, die eine weitere Verkleinerung der Ergebnissicht bewirken würde. Trotz dieser Einschränkung wird das Datumbereichsprädikat *o_orderdate BETWEEN '1995-01-01' AND '1995-03-31'* genutzt, das die Ergebnisrelation um 93 Prozent verkleinert und somit die Operationszeit und Sichtgröße entscheidend reduziert. Dies ist auch der Grund dafür, dass obwohl die Konzeptapplikation mehr Sichten als der Design Advisor vorschlägt, der Verbrauch an Speicher dennoch geringer ausfällt.

4.7 Zusammenfassung

Die Konzeptapplikation wurde mit Java aus Teilen eines DBMS emuliert. Die Anfragen werden in das Query Graph Model umgewandelt und mit Boxen des Mapgraphs gematcht. Nachdem der Workload abgearbeitet wurde, wird der Mapgraph der Klasse *MapGraphAnalyzer* übergeben, die mittels der Profitfunktion geeignete ASTs vorschlägt.

Zur Verwaltung der Konzeptapplikation existieren Werkzeuge, die die Definition von Tabellenboxen, das Einfügen von einzelnen Anfragen bzw. eines gesamten Workloads und das Betrachten des Mapgraphs vereinfachen.

Die Größe des Mapgraphs spielt beim Aufwand eine entscheidene Rolle. Der Aufwand der Navigation hängt davon ab, wieviele Boxen zum Matching besucht werden müssen. Daher ist der Aufwand des Matching und der Optimierungen proportional zur Größe des Mapgraphs. Der Aufwand für das Traversieren des Mapgraphs summiert sich für jede im Mapgraph befindliche Tabellenbox. Damit ergibt sich aus der Anzahl der Tabellenboxen des Mapgraphs $\#_{TB}(MG)$ und dem Aufwand des Traversierens jedes Baumes der maximale Gesamtaufwand $O(\#_{TB}(MG) \times \log_{m_{MG}}(n_{MG}))$. Bei Einfüge-, Restrukturierungs oder Mergingoperationen werden Paare aus allen Elternboxen einer Box gebildet und zur Matchfunktion geleitet. Der Aufwand, der dabei entsteht, ist höchstens quadratisch, also $O(n^2)$, wobei n der maximalen Anzahl an Elternboxen entspricht.

Zum Vergleich des Konzeptes wurde der Design Advisor von IBM und ein von TPC-H generierter Workload genutzt. Der Workload wurde so angepasst, dass Überlappungen von Ergebnisrelationen entstanden. Der Design Advisor schlägt bei Verwendung des beschriebenen Workload nur Sichtdefinitionen vor, die ausschließlich Verbundprädikate enthalten. Daraus ergibt sich eine große Obermenge des Workloads der die Anfragezeit um 84 Prozent verbessert. Die Konzeptapplikation schlug speziellere Sichten vor, die die Ergebnisgröße der Sicht und Kompensationsberechnungszeit weiter reduzierte. Die Anfragezeit wurde dadurch um 98 Prozent verbessert.

Insgesamt ist der Design Advisor beim Vorschlagen von ASTs sehr allgemein. Es werden ausschließlich Verbundprädikate zur Materialisierung ausgewählt, was speziellere Anfragen kaum Optimierungsmöglichkeiten lässt. Es muss allerdings berücksichtigt werden, dass die Konzeptapplikation bei einer großen Menge von Punktanfragen die gleichen Vorschläge zur Materialisierung machen würde, da die allgemeine Selektion bzw. Gruppierung positiviert werden würde.

¹Dieser Fakt ist darin begründet, dass die Kompensation bei der Konzeptapplikation nicht zur Elternbox weitergereicht wird, sondern sofort als Elternbox weitergematcht wird. Diese Vereinfachung ist zwar korrekt, jedoch implementiert sie die Muster des QGM nicht vollständig und führt somit nicht zum optimalen Ergebnis, welches das Konzept liefern würde. Tabelle 4.3 zeigt bei der Anfrage aus Abbildung 4.8 einen Anteil der Gruppierungsoperation an der Anfragebeantwortung von 87 Prozent.

Kapitel 5

Fazit und Ausblick

Das Problem der automatischen Erstellung von materialisierten Sichten unter Ausnutzung des Query Graph Models bei einem variablen Workload wurde mit dieser Arbeit weitestgehend gelöst. Es wurde eine algebraische Struktur zum Abbilden und Bewerten des Workloads gefunden. Dabei werden allgemeine, wie auch spezielle Sichten bewertet, ohne dabei auf allgemeine Annahmen wie Prädikatreduzierung zurückzugreifen. Die Prädikate werden nach ihrer Verwendung in Clustern von Boxen unterteilt und referenziert. Dadurch ist die Bewertung nicht nur auf Anfrageebene, sondern speziell auf den Prädikateilmengen möglich. Gemeinsame Bereiche des Workloads können somit auf eine Materialisierung abgebildet werden. Jeder Prädikatcluster kann anhand seines speziellen Profits (berechnet auf Grundlage seines Referenzwertes, Berechnungsaufwands und Relationsgröße) bewertet und daraufhin entweder materialisiert werden oder nicht.

Die Erweiterung der Boxtypen der dynamischen Anfragebeantwortung um den Typ Tabellenbox dient der statistischen Erweiterung der Abbildung von Basistabellen und kann vom abstrakten Typ Box abgeleitet werden. Die allgemeine Form der Box kann somit weiterhin verwendet werden.

Es wurden Strategien zur Verwaltung des Workloads entwickelt, die Restrukturierungen und Merging von Prädikatmengen und Attributen ermöglichen. Durch die Restrukturierung wird die Teilmengenbedingung sichergestellt. Das Merging optimiert die Sichtauswahl auf Grundlage der Verschmelzung von Boxen oder Teilen daraus. Jede Merge-Box wird mit Hilfe der Profitfunktion bewertet und auf Grundlage dessen möglicherweise zur Materialisierung ausgewählt. Resultierend existiert eine Liste mit allen Boxen des Mapgraphs und Mergeboxen, die nach den Profiten der Boxen sortiert ist und deren Boxen absteigend materialisiert werden.

Zugriffspfade lassen sich perfekt vom Mapgraph ableiten. Die Elternboxen einer materialisierten Box enthalten genau die Prädikate, die zur Erstellung von Indexen auf dieser Sicht notwendig sind. Leider konnte die Konzeptapplikation dieses Feature noch nicht implementieren, da für die Entwicklung zu wenig Zeit zur Verfügung stand. Aus diesem Grund kann noch kein Optimierungsgewinn ermittelt werden.

Um auf die Veränderung eines dynamischen Workloads bei der Materialisierung zu reagieren, werden die Referenzen der Boxen zyklisch um einen Faktor dekrementiert. Unterschreiten die Boxen eine untere Referenzschwelle, werden sie aus dem Mapgraph gelöscht. Dies vermindert den Aufwand der Verwaltung des Mapgraphs, da viele Boxen aus Punktanfragen nicht mehr behandelt werden müssen. In dieser Arbeit wurde ein

fester Wert für den oberen und unteren Referenzschwellenwert und den Dekrementierfaktor gesetzt. Bei dynamischen Workloads ändert sich ständig die Anzahl der Boxen im Mapgraph. Dies hat zur Folge, dass eine konstante Einstellung der Schwellenwerte einen zu großen oder zu kleinen Mapgraph zur Folge hätte. Daraufhin wären auch die Vorschläge der ASTs unoptimal. Dieser Fakt müsste noch verbessert werden und stellt daher eine Aufgabe für zukünftige Arbeiten dar.

In der Praxis wurde eine Konzeptapplikation entwickelt, die mit dem Design Advisor von IBM DB2 verglichen wurde. Dabei stellte sich heraus, dass bei einem Workload, der sowohl aus Bereich-, als auch Verbund- und Punktanfragen bestand, eine Verbesserung der AST-Vorschläge und somit einer Verminderung der Operationsanzahl zur Anfragezeit zu verzeichnen war. Allerdings wurde auch festgestellt, dass bei gewissen Workloadarten die Vorschläge gleichwertig sein können.

Insgesamt wurde mit dieser Arbeit ein Verfahren entwickelt, das der Verbesserung der Anfragezeit von Datenbankmanagementsystemen dient. Sie stellt die Grundlage für aufbauende Verbesserungen der einzelnen Module des Konzeptes dar. Diese umfassen vor allem die Muster des Mergings, die Profitfunktion und die Indexauswahl, welche die Schlüsselstellen für die Verbesserung der Sichtqualität darstellen und daher gesonderte Aufmerksamkeit erhalten sollten.

Literaturverzeichnis

- [1] R. G. Bello, K. Dias, A. Downing, J. James J. Feenan, J. L. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in oracle. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 659–664, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [2] D. Bonchev. *Chemical Graph Theory: Introduction and Fundamentals*. Taylor francis (UK), 1991.
- [3] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. *Proceedings of the Eleventh International Conference on Data Engineering*, pages 190–200, 1995.
- [4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *ICDE '95: Proceedings of the Eleventh International Conference on Data Engineering*, pages 190–200, Washington, DC, USA, 1995. IEEE Computer Society.
- [5] A. Eisenberg and J. Melton. Sql: 1999, formerly known as sql3. *SIGMOD Rec.*, 28(1):131–138, 1999.
- [6] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 152–159, Washington, DC, USA, 1996. IEEE Computer Society.
- [7] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 358–369, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
- [8] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 205–216, New York, NY, USA, 1996. ACM Press.
- [9] W. H. Inmon. *Building the Data Warehouse (3rd Edition)*. John Wiley & Sons, Inc., New York, 2002.
- [10] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys (CSUR)*, 16(2):111–152, 1984.

- [11] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer New York, 2004.
- [12] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 100–111, New York, NY, USA, 1997. ACM Press.
- [13] G. Saake and A. Heuer. *Datenbanken: Implementierungstechniken*. mitp-Verlag, Bonn, 1999.
- [14] P. Scheuermann, J. Shim, R. Vingralek, N. Aeronautics, S. Administration, and U. States. *WATCHMAN a Data Warehouse Intelligent Cache Manager*. National Aeronautics and Space Administration; National Technical Information Service, distributor, 1996.
- [15] U. Schoning. *Logik für Informatiker*. Spektrum Akademischer Verlag; Auflage: 5., 2000.
- [16] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 318–329, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.
- [17] T. P. P. C. (TPC). TPC-H Benchmark. <http://www.tpc.org>, 2006.
- [18] H. Wang and C. Zaniolo. User-Defined Aggregates for Datamining.
- [19] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex sql queries using automatic summary tables. *SIGMOD Rec.*, 29(2):105–116, 2000.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 6. September 2007

Ronny Bubke

