

Fakultät für Mathematik und Informatik

Lehrgebiet Datenbanksysteme für neue Anwendungen

Prof. Dr. Ralf Hartmut Güting

Masterarbeit

im Studiengang

Master of Computer Science

Hashbasiertes Gruppieren für allgemeine Aggregatfunktionen

vorgelegt von

Dieter Capek

Matrikelnummer 7852630

Lenzstraße 41a, 82140 Olching

betreut von

Prof. Dr. Ralf Hartmut Güting

Abgabetermin 30.04.2012

Inhaltsverzeichnis

[1 Einleitung 6](#_Toc314651771)

[1.1 Hinführung. Datenmodelle und Aggregation 6](#_Toc314651772)

[1.2 Hashbasiertes Gruppieren für allgemeine Aggregatfunktionen. 9](#_Toc314651773)

[1.3 Ausblick auf die Arbeit 9](#_Toc314651774)

[2 Review 11](#_Toc314651775)

[2.1 Literatur Überblick zur Gruppierung 11](#_Toc314651776)

[2.2 Aggregation in marktüblichen Datenbank Systemen 14](#_Toc314651777)

[2.3 Aggregation bei YouTube Videos 14](#_Toc314651778)

[2.4 Das Datenbank System Secondo 15](#_Toc314651779)

[2.4.1 Überblick 15](#_Toc314651780)

[2.4.2 Secondo Architektur 16](#_Toc314651781)

[2.4.3 Abfragen in Secondo 18](#_Toc314651782)

[3 Eigene Arbeit 22](#_Toc314651783)

[3.1 Formale Beschreibung der Fragestellung 22](#_Toc314651784)

[3.2 Spezifikation 23](#_Toc314651785)

[3.3 Entwurf 25](#_Toc314651786)

[3.3.1 Type Mapping 25](#_Toc314651787)

[3.3.2 Normales und Symmetrisches Verschmelzen 27](#_Toc314651788)

[3.3.3 Value Mapping 28](#_Toc314651789)

[3.3.4 Pseudocode für die Value Mapping Funktion 29](#_Toc314651790)

[3.4 Umgang mit Hauptspeicher, Phasen 32](#_Toc314651791)

[3.5 Entwicklungsumgebung 34](#_Toc314651792)

[3.6 Implementierung 34](#_Toc314651793)

[3.7 Experimentelle Auswertungen 35](#_Toc314651794)

[4 Schluss 36](#_Toc314651795)

[5 Literaturverzeichnis 37](#_Toc314651796)

**Abbildungsverzeichnis**

[Abbildung 1: Konzeptuelles Datenmodell für die Abverkäufe eines Handelsunternehmens 7](#_Toc312347045)

[Abbildung 2: Aggregation der Aufrufe eines YouTube Videos nach Zeit 15](#_Toc312347046)

[Abbildung 3: Aggregation der Aufrufe eines YouTube Videos nach Herkunftsland des Betrachters 15](#_Toc312347047)

[Abbildung 4: Aggregation der Aufrufe eines YouTube Videos nach Personenmerkmalen 15](#_Toc312347048)

[Abbildung 5: Zusammenspiel der Secondo Hauptkomponenten 17](#_Toc312347049)

[Abbildung 6: Secondo Architektur 17](#_Toc312347050)

[Abbildung 7: Grafische Darstellung des Ergebnisses einer Abfrage im Secondo Hoese Viewer. 19](#_Toc312347051)

[Abbildung 8: Datenstruktur des groupby2 Operators. Ein Array von STL Vektoren. 29](#_Toc312347052)

**Verzeichnis der Listings und Tabellen**

[Listing 1: SQL Statement zur Aggregation von Lagerdaten eines Handelsunternehmens 8](#_Toc312347053)

[Listing 2: Secondo Abfrage in einer SQL ähnlichen Syntax 18](#_Toc312347054)

[Listing 3: Textdarstellung des Ergebnisses einer Abfrage in Secondo 19](#_Toc312347055)

[Listing 4: Secondo Abfrage in „User Level Syntax“ 20](#_Toc312347056)

[Listing 5: Secondo Abfrage in „nested list Syntax“ 21](#_Toc312347057)

[Tabelle 6: Spezifikation des groupby2 Operators 25](#_Toc312347058)

[Listing 7: Pseudo Code für die Value Mapping Funktion des groupby2 Operators 32](#_Toc312347059)

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht.

Ort und Datum Dieter Capek

Gesamtumfang: 60 bis 70 Seiten

|  |  |  |
| --- | --- | --- |
| **Kapitel** | **Umfang min** | **Umfang max** |
| Einleitung | 2 | 5 |
| Review | 15 | 20 |
| Eigene Arbeit | 30 | 40 |
| Schluss | 2 | 5 |
| Gesamtumfang | 60 | 70 |

# Einleitung

## Hinführung. Datenmodelle und Aggregation

Datenbanken modellieren einen Ausschnitt der Welt in einem Computersystem. Ziel ist es, aus diesem System Aussagen über die modellierte Welt zu gewinnen. Die modellierten Fakten liegen in verschiedener Granularitäten vor, die von der Beschreibung einer einzelnen Geschäftstransaktion bis zu Aussagen über Personengruppen oder Zeiträume reichen.

Beispiele für die Beschreibung einzelner Geschäftstransaktionen sind im Handel der Kauf eines einzelnen Artikels, der an einer Scanner Kasse erfasst wird. Bei einem Telekommunikation Unternehmen beschreib der sogenannte Call Detail Record eine einzelne Telefonverbindung im Detail. Für typischerweise verwendete Aggregate können im Handel Verdichtungen der Abverkäufe nach Markt, Artikel und Zeit (Tag, Woche und Monat) gelten. In der Telekommunikation sind z.B. Verdichtungen nach Kunde, verwendetem Produkt und Zeit üblich.

Historisch gesehen wurden Datenbanksysteme zunächst vor allem für sogenannte Online Transaction Processing (OLTP) Systeme eingesetzt, die Daten auf dem Niveau der einzelnen Geschäftstransaktionen verarbeiten.

Erst später, mit stark gestiegenen Rechnerleistungen, konnten entscheidungsunterstützende Systeme entwickelt werden. Enterprise Data Warehouse Systeme versuchen, alle Daten eines Unternehmens in den entscheidungsrelevanten Ausschnitten zur Verfügung zu stellen. Heute spricht man von Business Intelligence (BI) Systemen. Diese beinhalten sowohl Transaktionsdaten als auch aggregierte Daten.

Frühe Versuche, die Aggregation bereits beim Laden vorzunehmen und dann nur noch die aggregierten Daten zu analysieren, dürfen als überholt gelten. Es gilt als einer der Hauptvorteile dieser Systeme, bei Bedarf in Echtzeit die Analyse verfeinern und auf detailliertere Daten verwenden zu können.

Geschäftsentscheidungen werden meist auf der Ebene der Aggregate getroffen. Diese liegen meist in mehreren Ausprägungen vor. Im der Abbildung enthält die Faktentabelle Abverkäufe Informationen zum einzelnen Artikel (z.B. Verkauf eines Bügeleisens). Die Aggregation erfolgt im vorliegenden konzeptuellen Datenmodell hinsichtlich der Dimensionen Zeit (einzelner Warenkorb, Tag und Monat), Artikel (Artikel, Warengruppe und Abteilung) und Verkaufsorganisation (Markt, Verbund, Land).

Beispiele für Geschäftsfragen können hier z.B. sein

* Welchen Preis hatte ein durchschnittlicher Warenkorb?  
  Vergleich von verschiedenen Regionen.
* Wie hat sich der Umsatz mit Unterhaltungselektronik in der Vorweihnachtszeit 2011 entwickelt? Vergleich zum Vorjahr 2010.
* Wie lange benötigt in Italien ein neu eröffneter Markt, um in der Abteilung Tonträger ähnlichen Umsatz zu erzielen wie ein etablierter Markt?

Abbildung : Konzeptuelles Datenmodell für die Abverkäufe eines Handelsunternehmens

Es ist ein wesentliches Leistungsmerkmal eines BI Systems, wie schnell die verschiedenen Aggregate berechnet werden können. Es ist durchaus üblich, dass die Aggregate insgesamt mehr Plattenplatz in Anspruch nehmen als die Detaildaten.

Dieser Trend wird unterstützt durch die starke Leistungserhöhung bei Datenbank Rechnern, die durch den schnellen Fortschritt von Hardware und Software Technologien ermöglicht wird. Leistungsfähigere Systeme verschieben die Machbarkeitsgrenze und führen zur Haltung größerer Datenmengen. Solche System halten auch mehr Aggregate sind auf effiziente Algorithmen für Joins und Aggregation angewiesen.

In Structured Query Language (SQL) werden Gruppierungen durch das Group By Statement beschrieben. Das folgende Bespiel verdichtet Lagerbestände nach Markt, Tag und Produkt. Nach den Gruppierungsmerkmalen werden die Menge des Lagerbestands, sowie eine Bewertung nach Verkaufspreisen vorgenommen.

insert into EP.age\_stock\_temp

select

OUTLET\_ID,

date\_id,

PROD\_ID,

sum( stock\_qty ),

sum( STOCK\_SALES\_VAL ),

sum( STOCK\_SALES\_VAL\_NET ),

sum( STOCK\_LAST\_ROI\_VAL ),

sum( STOCK\_ROI\_VAL )

from ep.stock

where

stock\_qty > 0 and

date\_id = 980825 and -- insert stock date here

depot\_id in (select depot\_id from depot where depot\_cat\_id in (0,1))

group by 1, 2, 3;

Listing : SQL Statement zur Aggregation von Lagerdaten eines Handelsunternehmens

Im Focus dieser Arbeit sind erweiterte Datenmodelle von räumlichen und sich bewegenden Objekten.

Von Interesse sind die räumlichen Objekte Punkt, Linie und Region, sowie Mengen dieser Objekte, z.B. ein Netzwerk von Straßen oder Eisenbahnverbindungen und Partitionen, z.B. eine Länderkarte von Europa. Es geht um die Darstellung bewegter Objekte jeglicher Art in Datenbanken. Bewegliche Objekte können hier z.B. Fahrzeuge, Wale, Grundstücke oder Wirbelstürme sein. Grundsätzlich sind bewegte Objekte geometrische Komplexe, die ihre Form und Position stetig in der Zeit verändern.

Erweitert man die Datenmodelle um räumliche und sich bewegende Objekte, stellt sich auch Aggregation sehr viel umfangreicher dar. Anfragen mit Aggregation an ein solches Datenbank System sind nun z.B.

* Vereinigung mehrerer räumlicher Gebiete.  
  Gruppieren der Landkreise nach Bundesland: Jeder der Landkreise ist durch eine Region und diese durch etwa 100 Punkte dargestellt, die die äußeren Grenzen des Landkreises modellieren. Die Vereinigung der Landreise eines Bundeslandes ergibt die Region des Bundeslandes.
* Anzahl von beweglichen Objekten in einer Region zu einem bestimmten Zeitpunkt, etwa zum Zweck der Verkehrskontrolle und für Stadtplanung.  
    
  Wie viele Flugzeuge werden den Frankfurter Flughafen innerhalb des Zeitintervalls zwischen einer und zwei Stunden von jetzt gerechnet erreichen?  
    
  Wie viele Schiffe werden den Eingang des Nord-Ostsee-Kanals im Laufe des morgigen Tages erreichen?  
    
  Wie viele Personenwagen werden am Montagmorgen zwischen 6:30 und 8:00 Uhr das Münchener Stadtgebiet anfahren?

Theoretisch ist interessant, dass die beiden formalen theoretischen Ansätze für relationale Systeme, Relationenalgebra und Relationenkalkül, keine Operatoren für Aggregation vorsehen. Diese Erweiterungen entstanden im Rahmen der Entwicklung der SQL Abfragesprache und ihrer Vorgänger.

*Aggregation has been added to both relational calculus and algebra and adds the same expressive power to each of them [Klug 1982]* (Graefe, 1993)*.*

Zielgruppe dieser Master Arbeit sind Leser, die einen Abschluss in Informatik haben. Im Text wird davon ausgegangen, dass die Leser ein Verständnis relationaler Datenbanken besitzen. Genaue Kenntnisse zu relationalen Datenbanken werden nicht vorausgesetzt.

## Hashbasiertes Gruppieren für allgemeine Aggregatfunktionen.

Thema dieser Master Arbeit stellt das effiziente Hash basierte Gruppieren dar.

Das genaue Thema dieser Master Arbeit ist in folgender Aufgabenstellung beschrieben

* Operatorsyntax wie vorher. Beim Eintreffen eines Tupel wird dieses unmittelbar im Tupel Strom für seine Gruppe verarbeitet. Dabei wird für jede Gruppe entweder ein aggregierter Wert aufbewahrt (für Werte fester Größe) oder ein Stack von Werten, um symmetrisches Verschmelzen zu ermöglichen.
* Der Speicherplatz ist proportional zur Anzahl der Gruppen, falls Werte konstanter Größe verarbeitet werden. Ansonsten ist der Platzbedarf gegeben durch die Summe der Größen bereits angelegter Stack Werte.
* Sobald der Speicher erschöpft ist,

a) Wird bei Werten fester Größe für jede neu auftretende Gruppe das Tupel nicht verarbeitet, sondern in den Tuple Buffer geschrieben.

b) Wird bei Werten variabler Größe für ein neu zu verarbeitendes Tupel die Verknüpfungsoperation durchgeführt. Danach wird die Gruppe ausgelagert (in ein

spezielles File), d.h. die Stack Werte werden herausgeschrieben. Dadurch sinkt

der verwendete Speicherplatz. Von nun an werden Tupel für nicht vorhandene

Gruppen sofort herausgeschrieben für den nächsten Durchlauf. Für vorhandene

Gruppen wird jeweils die Gruppe ausgelagert, wenn der Speicherplatz überschritten

wird.

* Modellierung der Kosten in Abhängigkeit vom verwendeten Speicher
* Kostenschätzung auf der Grundlage von Samples
* Experimente zur Übereinstimmung von Modell und Realität
* Fortschrittsschätzung

## Ausblick auf die Arbeit

Ausblick auf die Arbeit

*Welches Problem wird da gelöst?*

Hash basiertes Gruppieren in Secondo. Der heute vorhandene groupby Operator setzt eine Sortierung der Input Tupel nach den Gruppierungsattributen voraus.

*Performance: ich will einen schnellen Algorithmus schreiben, da der Sort sehr teuer werden kann.Welche Laufzeiten werden heute erreicht? Bei welchen Problemen?*

Gibt es Messungen dazu? Kann ich etwas referieren? Wo sind aktuell Schwierigkeiten?

# Review

## Literatur Überblick zur Gruppierung

*Grundlagen darstellen, die für das Verständnis der Arbeit nötig sind.*

*Literatur Überblick: „DB Literatur zur Gruppierung darstellen. Umsehen in der Literatur“.*

\*\* Gegenüberstellung Sortierung und Hashing als grundlegende Verfahren

The purpose of many query-processing algorithms is to perform some kind of matching, i.e., bringing items that are “alike” together and performing some operation on them. There are two basic approaches used for this purpose, sorting and hashing. This pair permeates many aspects of query processing, from indexing and clustering over aggregation and join algorithms to

methods for parallelizing database operations (Graefe, 1993).

\*\* Vergleich und Analogie zu Join Algorithmen

Grouping algorithms can be derived from the join algorithm by letting the candidate output records play the role of records from the small join input (Goetz, 2011).

Grundlegende Ansätze zur Berechnung eines Gruppierung Operators sind Sortierung und Hashing. Beides sind blockierende Operatoren. Klassisch wurde Gruppierung durch eine Sortierung hinsichtlich der Gruppierungsmerkmale realisiert.

\*\* duality of sort and hash based algorithms (chapter 7)

Both approaches permit in-memory versions for small data sets and disk-based versions for larger data sets. If a data set fits into memory, quicksort is the sort based method to manage data sets while classic (in-memory) hashing can be used as a hashing technique. It is interesting to note that both quicksort and classic hashing are also used in memory to operate on subsets after “cutting” an entire large data set into pieces. The cutting process is part of the divide-and-conquer paradigm employed for both sort- and hash-based query-processing algorithms.

..

There exists, however, an important difference. In the sort-based algorithms, a large data set is divided into subsets using a physical rule, namely into chunks as large as memory. These chunks are later combined using a logical step, merging. In the hash-based algorithms, large inputs are cut into subsets using a logical rule, by hash values. The resulting partitions are later combined using a physical step, i.e., by simply concatenating the subsets or result subsets (Graefe, 1993).

\*\* Hashing

In general, when equality matching is required, hashing should be considered because the expected complexity of set algorithms based on hashing is O(N) rather than 0( N log N) as for sorting (Graefe, 1993).

Hash-based query processing algorithms use an in-memory hash table of database objects to perform their matching task. If the entire hash table (including all records or items) fits into memory, hash-based query processing algorithms are very easy to design, understand, and implement, and they outperform sortbased alternatives (Graefe, 1993).

However, if the required hash table is larger than memory, hash table overflow occurs and must be dealt with. There are basically two methods for managing hash table overflow, namely avoidance and resolution. In either case, the input is divided into multiple partition files such that partitions can be processed independently from one another, and the concatenation of the results of all partitions is the result of the entire operation (Graefe, 1993).

Hash-based algorithms tend to produce their outputs in a very unpredictable order, depending on the hash function and on overflow management (Graefe, 1993).

Beschreibung der drei Fälle (S 91 im 1993 Artikel. Auch gut beschrieben in Kap 2.1 ff im 2011 Artikel)

* Hash table overflow avoidance
* Hash table overflow resolution
* Hybrid hashing algorithms



(Graefe, 1993)

\*\* Grundlegende Hash Verfahren (S 119)

We call this direct recursive implementation of hash-based partitioning depth-first partitioning and consider its behavior as well as its resource sharing and performance effects a dual to eager merging in sorting. The alternative schedule is breadth-first partitioning, which completes each level of partitioning before starting the next one (Graefe, 1993).

\*\* Aggregation

Aggregation is typically supported in two forms, called scalar aggregates and aggregate functions [Epstein 1979]. Scalar aggregates calculate a single scalar value from a unary input relation, e.g., the sum of the salaries of all employees. Scalar aggregates can easily be determined using a single pass over a data set.

\*\* In Secondo sind das die heutigen Operatoren aggregate, aggregateB, aggregateC.

Aggregate functions, on the other hand, determine a set of values from a binary input relation, e.g., the sum of salaries

for each department. Aggregate functions are relational operators, i.e., they consume and produce relations (Graefe, 1993).

\*\* das ist mein Fall

There are three types of algorithms for aggregation and duplicate removal based on nested loops, sorting, and hashing (Graefe, 1993).

The first algorithm, which we call nested-loops aggregation, is the most simple-minded one. Using a temporary file to accumulate the output, it loops for each input item over the output file accumulated so far and either aggregates the input item into the appropriate output item or creates a new output item and appends it to the output file. Obviously, this algorithm is quite inefficient for large inputs, even if some performance enhancements can be applied (Graefe, 1993).

Early work on implementation techniques for query processing in relational databases assumed that aggregation requires an input sorted on the grouping columns. Once the input sorted, the cost of the actual aggregation is minimal or even trivial. Thus, sorting the input was usually the dominant cost of aggregation operations (Goetz, 2011).

For aggregation, items are sorted on their grouping attributes (Graefe, 1993).

Es gelten also die Überlegungen zum Sortieren und dessen Kosten. Secondo Groupby Operator mit Aggregate je Gruppe: einfach den Gruppenwechsel finden und alles bis zum nächsten Wechsel aggregieren.

\*\* Auf Aggregation beziehen. Wenn die Gruppierung die Datenmenge stark reduziert, ist hashing besser. Enthält jede Gruppe ein Element, ist der ganze Operator zweckfrei – egal, wie er implementiert ist.

Hash *join* has been considered the best join algorithm to exploit large memory and to process unsorted inputs (Goetz, 2011).

Only the smaller input determines the number of partitioning levels, independent of the size of the larger input. Thus, hash join exploits the difference in the sizes of its inputs (Goetz, 2011).

Hashing can also be used for aggregation by hashing on the grouping attributes. Items of the same group (or duplicate items in duplicate removal) can be found and aggregated when inserting them into the hash table. Since only output items, not input items, are kept in memory, hash table overflow occurs only if the output does not fit into memory (Graefe, 1993).

Evtl. Den Kostenvergleich in Kap 4.4 einbauen: S 102, Abb. 11 in .

\*\* gute Formulierung eines Fazit

Es gibt mehrere alternative Algorithmen zur Implementierung jedes rela­tionalen Operators, und für die meisten Operatoren kann keine universell überlegene Technik angegeben werden. Welcher Algorithmus der geeignetste ist, hängt von vielen Faktoren ab wie der Größe der beteiligten Relationen und dem Vorhandensein von Inde­xen und Sortierreihenfolgen.

The important conclusion from these dualities is that neither the absolute input sizes nor the absolute memory size nor the input sizes relative to the memory size determine the choice between sort- and hash-based query-processing algorithms. Instead, the choice should be governed by the sizes of the two inputs into binary operators relative to each other and by the danger of performance impairments due to skewed data or hash value distributions. Furthermore, because neither algorithm type outperforms the other in all situations, both should be available in a query execution engine for a choice to be made in each case by the query optimizer (Graefe, 1993).

?? Passt dieses Thema hier herein?  
Aggregation and Selectivity Estimation. Kap 8.5. Window queries, join queries and nearest neighbor queries (Güting & Schneider, Moving Objects Databases, 2005).

## Aggregation in marktüblichen Datenbank Systemen

-- Soll ich dieses Kapitel überhaupt reinnehmen?  
 „Wie macht es ein marktgängiges System“: Teradata Groupby – passt das hier herein?   
Bin aber unsicher, da

* TD ist MPP, zunächst muss also die physische Trennung überwunden werden
* Verwendung von Spool Files, kein Tuple Strom
* Wie genau weiß ich es? TD aggregiert nur einfache Datentypen (wahrscheinlich keine Attribute)

Gruppieren in spaltenbasierten Systemen wie Vertica und Sybase.

?? da gibt es wohl auch keine Tupel Ströme. Info liegt oft nur sehr allgemein vor.

## Aggregation bei YouTube Videos

Als Beispiel für eine innovative Anwendung von Aggregation kann die Darstellung von Daten zum Ansehen von Internet Videos gelten. Es geht hier um die Aggregation von Daten aus den Log Dateien von Webservern.

Als Beispiel werden die Daten für das YouTube Video „FIVB Heroes in Super Slow Motion - Ekaterina Gamova“ dargestellt (Benutzer FIVB Heroes, 2011).

Die Aggregationen auf der YouTube Webseite zeigt die Gesamtzahl der Aufrufe, ihre zeitliche und geografische Verteilung, sowie eine Beschreibung der Betrachter nach Geschlecht und Alter für die Zielgruppen Analyse.

Aufrufe insgesamt: 5.045

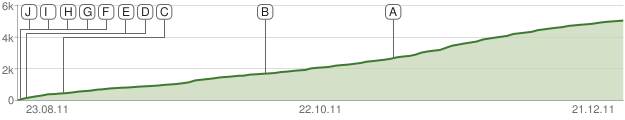
****

Abbildung : Aggregation der Aufrufe eines YouTube Videos nach Zeit

Dieses Video ist am beliebtesten in:

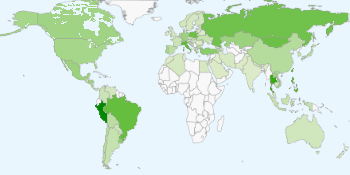
http://s.ytimg.com/yt/img/thematic_map_key-vflod8fQ3.png Mehr/Weniger

Abbildung : Aggregation der Aufrufe eines YouTube Videos nach Herkunftsland des Betrachters

Zielgruppen. Am liebsten sehen dieses Video:

|  |  |
| --- | --- |
| Geschlecht | Alter |
| Männlich | 45-54 |
| Männlich | 35-44 |
| Weiblich | 13-17 |

Abbildung : Aggregation der Aufrufe eines YouTube Videos nach Personenmerkmalen

Sicherlich handelt es sich hier um voraggregierte Werte, die nicht in Echtzeit berechnet werden. Allerdings wird die Aggregation kompliziert durch die Replikation der Video Server, die hohe Anzahl verfügbarer Filme und sehr große Datenmengen.

## Das Datenbank System Secondo

### Überblick

Secondo ist ein Datenbanksystem, das vor allem für Forschung und Lehre im Bereich Datenbanken eingesetzt wird. Es bietet einen Systemrahmen, in den beliebige Datenmodelle eingebettet werden können.

Neben dem relationalen Datenmodell können objektorientierte, temporale, räumliche oder XML Datenmodelle eingesetzt werden. So können z.B. Daten wie Graphen, Netzwerke oder chemische Formeln gespeichert werden. Datenmodelle werden durch Algebra Module in das Secondo System eingebunden. Der Schwerpunkt der vorliegenden Algebra Implementierungen liegt hierbei auf räumlichen und zeitlich-räumlichen Datenmodellen. Dies ergibt sich durch den Schwerpunkt des Lehrgebiets auf „Moving Objects Databases“.

Secondo wurde seit der Mitte der 90er Jahre vor Forschern und Studenten in der Forschungsgruppe von Prof. Güting an der Fernuniversität in Hagen, am Lehrgebiet „Datenbanksysteme für neue Anwendungen“, entwickelt.

Bei Secondo handelt es sich um frei verfügbare Software, die der GNU General Public License der Free Software Foundation unterliegt. Der Quellcode ist frei verfügbar auf der Secondo Webseite.

Es liegen Secondo Versionen für die Plattformen Windows, Linux und Mac OS X vor. Secondo verwendet die Berkeley Datenbank für persistentes Speichern. Auf diese Weise sichert sich Secondo Funktionalitäten wie Transaktion Management, die Verwaltung von Datenbanken und eine Schnittstelle zu Dateisystemen.

Für den Aufbau einer Secondo Umgebung sind eine Reihe von Software Paketen erforderlich, z.B. der C++ Compiler der GNU Compiler Collection, die Berkely Datenbank für persistentes Speichern und den Java Software Development Kit für die Endbenutzer Schnittstellen.

### Secondo Architektur

Muss hier als theoretische Grundlage„second order signature“ rein? Falls ja, ist es ein zusätzliches Kapitel, z.B. vor diesem Kapitel

Das Secondo System besteht aus drei Hauptkomponenten

* Der Secondo Kern implementiert die einzelnen Datenmodelle, ist erweiterbar durch Algebra Module und führt die Verarbeitung von Abfragen durch. Der Secondo Kern ist in C++ implementiert und verwendet die Berkeley Datenbank.
* Der Optimizer stellt für das relationale Datenmodell eine SQL ähnliche Abfragesprache und eine Abfrageoptimierung zur Verfügung. Der Optimizer ist in Prolog realisiert.
* Als grafische Schnittstelle für Endbenutzer steht eine Reihe von „Viewern“ zur Verfügung, die auf die Anzeige von räumlichen Datentypen und von sich bewegenden Objekten spezialisiert sind. Diese „Viewer“ sind in Java entwickelt.

Diese Komponenten können im Zusammenspiel miteinander oder unabhängig voneinander verwendet werden (Abbildung 5).

Die Verwendung des Secondo Optimizer ist optional: Abfragen können in einer SQL ähnlichen Abfragesprache an den Optimizer gerichtet werden oder in „User Level Syntax“ direkt an den Secondo Kern zur Verarbeitung übergeben werden (siehe Kapitel 2.4.3).



Abbildung : Zusammenspiel der Secondo Hauptkomponenten

Abbildung 6 gibt einen Überblick zur Architektur des Secondo Systems.



Abbildung : Secondo Architektur

Ober in der Abbildung und in Schicht (6) sind die verschiedenen Endbenutzer Schnittstellen dargestellt. Es ist Einbenutzer Betrieb (einfache Kommando Shell) oder Client-Server Betrieb für eine Vielzahl von Benutzern möglich (grafische Werkzeuge).

Der Command Manager (Schicht 5) leitet Abfragen je nach Art der verwendeten Sprache an den Optimizer oder den Query Prozessor weiter (siehe Kapitel 2.4.3 für eine genaue Beschreibung).

Der Optimizer (Schicht 4) stellt für das relationale Datenmodell eine SQL ähnliche Abfragesprache und eine Abfrageoptimierung zur Verfügung. Seine Verwendung ist optional.

Der Query Prozessor (Schicht 3) führt Abfragen aus. Hierzu wird ein Operatorbaum erstellt und dann jeder Knoten dieses Baums bewertet, indem Operatorfunktionen der Algebren aufgerufen werden. Operatorfunktionen können Parameterfunktionen sein oder Tupel Ströme verarbeiten.

Algebra Module (Schicht 2) implementieren Datentypen und die zugehörigen Operatoren. Algebra Module stellen alle Unterstützungsfunktionen für die Integration zu den anderen Teilen des Secondo Systems zur Verfügung. Zurzeit sind etwa 30 Algebren in Secondo realisiert. Einige Beispiele für Algebren sind

* StandardAlegebra: Stellt die Datentypen int, real, bool und string bereit
* RelationAlgebra: Stellt den Datentyp „Relation“ bereit und enthält alle Operatoren, die für die Implementierung einer relationalen Sprache nach Art von SQL erforderlich sind.
* BTreeAlgebra: implementiert B-Bäume
* RTreeAlgebra: implementiert R-Bäume (mehrdimensionale räumliche Indexstrukturen)
* SpatialAlgebra: Stellt die räumlichen Datentypen point, points, line, region bereit

Die Secondo Tools (Schicht 1) sind eine Menge von Werkzeugen für den Secondo Betrieb, z.B.

* Werkzeuge für den Umgang mit geschachtelten Listen
* Ein Storage Manager
* Ein Tupel Manager, der auch eingebettete große Objekte handhaben kann
* Ein Parser für die „User Level Syntax“ Sprache

Der Secondo Kern verwendet Datenbanken, die aus Secondo Objekten bestehen. Datenbanken können erstellt, gelöscht, geöffnet und geschlossen, sowie in Dateien exportiert und von dort zurückgelesen werden. In den Dateien sind die Secondo Objekte als geschachtelte Listen in Textform repräsentiert .

### Abfragen in Secondo

Secondo stellt drei Abfragesprachen zur Verfügung. Die erste Möglichkeit zur Formulierung von Abfragen in Secondo ist die Verwendung einer SQL ähnlichen Sprache, die von einem in Prolog geschriebenen Optimizer weiterverarbeitet wird. Dieser stellt ein SQL Prädikat in einer Prolog Umgebung zur Verfügung.

Zur Verdeutlichung dient die folgende Beispielabfrage, die im sogenannten Hoese Viewer, einer grafischen Endbenutzer Schnittstelle, eingegeben wurde.

Beispielabfrage: Finde die Namen und Adressen aller italienischen Restaurants in Berlin, die weniger als zwei Kilometer vom Brandenburger Tor entfernt sind. (obige Webseite):

select [r:name, r:strasse, r:geoData]

from [restaurants as r, sehenswuerdpoi as s]

where [ s:name="Brandenburger Tor",

r:art="Italienisch",

distance(r:geoData, s:geodata) < 2000]

Listing : Secondo Abfrage in einer SQL ähnlichen Syntax

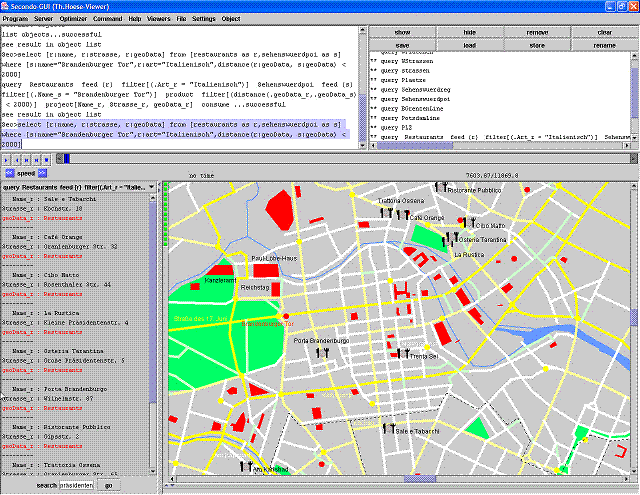


Abbildung : Grafische Darstellung des Ergebnisses einer Abfrage im Secondo Hoese Viewer.

Die Abfrage ist im Fenster oben links markiert sichtbar. Das Ergebnis, eine Liste von Restaurants, wird im linken unteren Teil der Grafik als Liste angezeigt. Zusätzlich zum Namen und der Straße des Restaurants wird die geografische Position als Punkt ausgegeben:

Name\_r : Sale e Tabacchi

Strasse\_r : Kochstr. 18

geoData\_r : point: (9566,11178)

...

Name\_r : Trenta Sei

Strasse\_r : Markgrafenstr. 36

geoData\_r : point: (9699,11887)

Listing : Textdarstellung des Ergebnisses einer Abfrage in Secondo

Rechts unten ist das Ergebnis als Grafik sichtbar: in einem Stadtplan von Berlin sind die Restaurants mit „Messer und Gabel Symbolen“ markiert.

Die zweite Möglichkeit, Abfragen in Secondo zu formulieren, ist die „User Level Syntax“. Die obige Beispielabfrage lautet in dieser Syntax wie folgt:

Secondo => query

Restaurants feedproject[Art, geoData, Name, Strasse]{r}

Sehenswuerdpoi feedproject[geoData, Name]{s}

filter[(.Name\_s = "Brandenburger Tor")]

symmjoin[(distance(.geoData\_r, ..geoData\_s) < 2000)]

filter[(.Art\_r = "Italienisch")]

project[Name\_r, Strasse\_r, geoData\_r] consume;

Listing 4: Secondo Abfrage in „User Level Syntax“

Man erkennt das Wort query zur Kennzeichnung einer Abfrage, die Namen der Relationen Restaurants und Sehenswuerdpoi, den Join Operator „symmjoin“, sowie die Filter Operator „filter“.

Durch den Secondo Parser werden Abfragen der „User Level Syntax“ umgesetzt in „nested list syntax“. Dieses Format verwendet Secondo System intern. Unsere Beispiel Abfrage lautet in „nested list syntax“:

(query

(consume

(project

(filter

(symmjoin

(rename

(feedproject Restaurants

(Art geoData Name Strasse))

r)

(filter

(rename

(feedproject Sehenswuerdpoi

(geoData Name))

s)

(fun

(streamelem1 STREAMELEM)

(=

(attr streamelem1 Name\_s)

"Brandenburger Tor")))

(fun

(lefttuple2 TUPLE)

(righttuple3 TUPLE2)

(<

(distance

(attr lefttuple2 geoData\_r)

(attr righttuple3 geoData\_s))

2000)))

(fun

(streamelem4 STREAMELEM)

(=

(attr streamelem4 Art\_r)

"Italienisch")))

(Name\_r Strasse\_r geoData\_r))))

Listing 5: Secondo Abfrage in „nested list Syntax“

*Einführung in Secondo: wie geht Query Prozessing? – soll ich das zusätzlich beschreiben? Soll ich die Auswertung der Queries beschreiben, also den Query Prozessor, den Operator Baum?  
Irgendwo sollte etwas über Tuple Strom Verarbeitung gesagt werden?*

# Eigene Arbeit

Beschreibung der eigenen Arbeit

Reihenfolge Spezifikation, Entwurf, Implementierung, Messergebnisse (notiert: Exprimentelle Auswertungen). Darstellung etwa wie bei einem Wasserfall Vorgehen

## Formale Beschreibung der Fragestellung

Ich will eine Notation finden, um die groupby2 Fragestellung zu beschreiben. Welches Modell soll ich verwenden? Ich versuche mal Matrizen.

Elemente aij beschreiben Attribute aus dem Eingabestrom, gij sind Gruppierungsattribute des groupby, fij sind Ergebnis Attribute der Aggregatunktionen.

Die Zeilen kann man als Tupel verstehen. Der Zeilenindex stellt die Reihenfolge des Tupels im Stream dar. Die Ausgangsrelation besteht also aus m horizontalen n-Tupeln (ai1 .. ain).

Durch Anwendung der groupby Operators g mit l <= n Gruppierungsattributen gi und Aggregatfunktionen fj entsteht als Ergebnis:

Das Ergebnis hat meist deutlich weniger Zeilen k <= m. Im linken Teil sieht man die Gruppen und im rechte Teil der Matrix die Werte der Aggregatfunktionen. Die Gruppierungsattribute sind eine Teilmenge der Ausgangsattribute, die Reihenfolge kann vertauscht sein.

Im heutigen Secondo groupby Operator sind beliebige Aggregat Funktionen zulässig. Dazu speichert der Operator alle Tupel der Gruppe und berechnet dann die Funktionen. Alle Attribute der Gruppe liegen vor.

Der neue groupby2 Operator setzt keine Sortierung der Tupel voraus. Groupby2 speichert nur das bisherige Gruppenergebnis und verarbeitet das aktuell hinzukommende Tupel „dazu“. Im Moment ist offen, auf welche Aggregatfunktionen wir einschränken müssen. Wahrscheinlich muss Zusatzinformation im Ergebnistupel gehalten werden, etwa die Anzahl der verarbeiteten (und definierten) Attributwerte für Durchschnittsbildung.

Der Folge der verarbeiteten Tupel ti je Gruppe entspricht für jede Aggregatfunktionen eine Folge von „Zwischenergebnissen“. Das letzte Ergebnis ist das Endergebnis.

Schränken wir zunächst die Aggregatfunktionen ein auf die Verwendung von Attributen in der eigenen Relation und Gruppe (es sind keine Joins oder sonstige Information „von außen“ zulässig). Eine Aggregatfunktion für eine Gruppe entsteht dann als Folge von Werten mit

f1 = AggFkt (Tupel 1 der Gruppe);

fn = AggFkt( fn-1, Tupel n der Gruppe);

Das klappt z.B. für max, min und count. Beispiel für die bei max erzeugte Folge:

f1 = max (a1j) = a1j;

fn = max( fn-1, anj);

Hierbei ist zu sichern, dass die verarbeiteten Attribute definierte Werte haben. Z.B. für avg muss für jede Ergebnisfunktion und jede Gruppe die Anzahl der definierten Werte mitgezählt werden.

Frage: welche Funktionen lassen sich in der Form

f1 = AggFkt (Tupel 1 der Gruppe);

fn = AggFkt( fn-1, Tupel n der Gruppe);

errechnen?

Man kann den Operator groupby2 so verstehen, dass für jede der in den Daten vorhandenen Tupel Gruppen eine oder mehrere Folgen berechnet werden. Diese Folgen sind definiert durch die angegebenen Aggregatfunktionen. Diese Funktionen müssen von der Art sein, dass die Reihenfolge der Tupel den Endwert der Folge (das Ergebnis der Aggregatfunktion) nicht beeinflusst.

Die Eingabedaten lassen sich verstehen als m unabhängige Mengen M1 .. Mm von Tupeln. Diese Mengen unterhalten unterschiedlich viele Elemente n1 bis nm.

Für jede dieser Mengen Mi werden j Folgen von Funktionswerten Fj berechnet (j konstant). Die Funktionswerte werden errechnet aus den Tupeln der Gruppe und Initialwerten Ij , die für alle Gruppen gleich sind.

Das Ergebnis des groupby2 Operators sind also m\*j Funktionswerte Fj(Mi). Jeder dieser Funktionswerte entsteht als Folge

Fj(Mi) 1 = Fj (Initialwert Ij, Tupel 1 der Gruppe Mi);

Fj(Mi) ni = Fj ( fni-1, Tupel ni der Gruppe Mi);

## Spezifikation

Die Spezifikation des groupby2 Operators folgt dem Format des Kapitel „5.3 Operators“ des Secondo Programmes Guide.

|  |  |
| --- | --- |
| **Name** | **groupby2** |
| Algebra | ExtRelationAlgebra |
| Syntax | \_ groupby2 [AttrList; FunctionList]  .spec File Eintrag:  operator groupby2 alias GROUPBY2 pattern \_ op [list; funlist]  implicit parameter tuple type TUPLE  **groupby2 [**  Liste von Gruppen Attributen durch Komma getrennt **;**  Liste der Aggregat Funktionen durch Komma getrennt  **]**  Der Benutzer kann für jede Aggregat Funktion wählen, ob normales oder Verschmelzen als Algorithmus zum Einsatz kommen soll. Dies wird durch die Syntax ausgedrückt.  Syntax für normales Verschmelzen: (Name : Funktion :: Initialwert)  Syntax für symmetrisches Verschmelzen: (Name : Funktion1 :: Funktion2) |
| Signatur | stream(Tuple) x AttrList x  (NewAttr-1 x (Tuple x Data -> Data) x Data) ..  (NewAttr-j x (Data x Data -> Data) x (Tuple -> Data))  ->  stream(Tuple(Attrlist)o Tuple([NewAttr-1: Data]..[NewAttr-j: Data])  Hierbei können beliebig viele neue Attribute als Ergebnis der Aggregat Funktionen angegeben werden.  Für normales Verschmelzen verwendet der Benutzer die Signatur  (Name x (Tuple x Data -> Data) x Data)  wie oben für das Attribut NewAttr-1.  Für symmetrisches Verschmelzen verwendet der Benutzer die Signatur  (Name x (Data x Data -> Data) x (Tuple -> Data))  wie oben für das Attribut NewAttr-j. |
| Bedeutung | Gruppiert die Tupel des Eingabestroms entsprechend der Attribute aus  ‘AttrList’ und errechnet einen oder mehrere Aggregatwerte NewAttr-1 .. NewAttr-j für jede Gruppe.  Die Gruppen Attribute, gefolgt von den errechneten Aggregaten  NewAttr-1 .. NewAttr-j, sind die Ergebnistupel und bilden den Ausgabestrom für den Folgeoperator.  Werden keine Gruppen Attribute angegeben, ist das Ergebnis ein einziges Tupel der Form NewAttr-1.. NewAttr-j. Dieses Tupel ist das Gesamtergebnis der Aggregation. |
| Beispiel | query plz feed filter[.Ort = "Dresden"]  groupby2[Ort; AnzOrte: fun(t:TUPLE, i:int) i + 1::0] consume; |
| Ergebnis | Ort : Dresden  AnzOrte : 146 |

Tabelle : Spezifikation des groupby2 Operators

Anmerkungen:

* Neue Funktionen werden in der Syntax des aggregateC Operators angegeben:  
  (Name : Argument2 :: Argument3)   
  Dadurch kann innerhalb der Aggregatfunktion auf mehrere Attribute des Input Tupel zugegriffen werden.
* Für die Gruppierung sind kein, ein oder mehrere Attribute möglich. Sie sind durch Komma getrennt (wie groupby). Ist keine Gruppierung angegeben, wird ein Gesamtergebnis für den Input Stream errechnet.
* Eine vorhergehende Sortierung nach Gruppierungsattributen ist nicht erforderlich.
* Für jede Aggregatfunktion, für die normales Verschmelzen gewünscht ist, muss ein Initialwert angegeben werden.
* Es wird symmetrisches Verschmelzen wie in aggregateB unterstützt. Der Benutzer wählt für jede Aggregatfunktion den von ihm gewünschten Algorithmus durch das Format der Spezifikation aus.

## Entwurf

### Type Mapping

Die Type Mapping Funktion überprüft, ob ein Operator eine vorgegebene Kombination von Datentypen als Parameter verarbeiten kann. Falls dies der Fall ist, wird der Datentyp des Operator Ergebnisses an die aufrufende Funktion zurückgegeben.

Ist eine Verarbeitung nicht möglich, wird das Symbol *typeerror* an die aufrufende Funktion zurückgegeben.

Dieses Kapitel beschreibt die Prüfungen, die die Type Mapping Funktion vornimmt. Diese Prüfungen werden inhaltlich beschrieben.

* Prüfung der Listen Struktur  
  Die Anzahl der übergebenen Argumente muss gleich drei sein:
  1. Input Stream,
  2. Liste von Gruppen Attributen
  3. Liste der Aggregat Funktionen

Fehlen darf nur die Liste von Gruppen Attributen, die anderen Parameter müssen angegeben sein.

* Test von Argument 1: Input Stream  
  Test auf Tuple Stream.
* Test von Argument 2: Liste der Gruppierungsattribute  
  Die Elemente der Liste sind durch Komma getrennt.  
  Für jedes Gruppierungsattribut wird geprüft, ob es sich um einen Attributnamen handelt und ob dieser im Input Tuple vorkommt.
* Test von Argument 3: Liste der Aggregatfunktionen.  
  Die Funktionen der Liste sind durch Komma getrennt.  
    
  Es muss mindestens eine Funktion angegeben sein.   
  Jeder Funktionsname muss ein gültiger Attributname sein.  
  Kein Funktionsname darf bereits als Attribut vorkommen.  
    
  Jede Aggregat Funktion kann in einem von zwei Formaten angegeben werden.  
  Für normales Verschmelzen: (Name : Funktion :: Initialwert)  
  Für symmetrisches Verschmelzen: (Name : Funktion1 :: Funktion2)  
    
  Für Aggregat Funktionen mit normalem Verschmelzen wird geprüft:
  + Die Funktion muss genau zwei Argumente haben (eine isMap<2> sein).
  + Das erste Argument der Funktion muss das Tupel des Stream sein.
  + Es muss genau ein Initialwert angegeben sein.
  + Der Initialwert muss vom Typ isDATA sein.
  + Das zweite Argument der Funktion muss vom gleichen Typ wie der zugehörige Initialwert sein.
  + Das Ergebnis jeder Funktion muss vom gleichen Typ wie der zugehörige Initialwert sein.

Für Aggregat Funktionen mit normalem Verschmelzen wird geprüft:

* + Funktion1 muss genau zwei Argumente haben (eine isMap<2> sein).
  + Funktion2 muss genau ein Argument haben (eine isMap<1> sein).
  + Das Argument der Funktion2 muss das Tupel des Stream sein.
  + Die beiden Argumente von Funktion1, das Ergebnis von Funktion1 und das Ergebnis von Funktion2 müssen alle vom gleichen Typ sein.
  + Die beiden Argumente von Funktion1, das Ergebnis von Funktion1 und das Ergebnis von Funktion2 müssen alle vom Typ isDATA sein.

Beim ersten bemerkten Fehler wird die Funktion mit der Rückgabe von Symbol::TYPEERROR() beendet.

Falls keine Fehler auftreten, wird der List Expression für das Operatorergebnis aufgebaut:   
Der Ergebnistyp besteht aus den Gruppierungsattributen, gefolgt von den neu errechneten Funktionswerten. Sind keine Gruppierungsattribute angegeben, kommen sie auch im Ergebnistyp nicht vor.

Mit Append werden zusätzlich die Anzahl der Gruppierungsattribute und ihre Position im Input Tuple an die Value Mapping Funktion weiter gegeben.

Dieser List Expression wird als Funktionsergebnis des Type Mapping zurückgegeben.

Beispiel 1: Secondo Abfrage und resultierende Ein- und Ausgabe der Type Mapping Funktion bei einem Gruppierungsattribut und einer Aggregatfunktion

Secondo => query plz feed groupby2[PLZ; a1:fun(t:TUPLE, i:int) 1 + i ::0] count;

Eingabe Type Mapping:

((stream (tuple ((PLZ int) (Ort string)))) (PLZ) ((a1 (map (tuple ((PLZ int) (Ort string))) int int) int)))

Ergebnis Type Mapping:

(APPEND (1 1) (stream (tuple ((PLZ int) (a1 int)))))

Beispiel 2: Secondo Abfrage und resultierende Ein- und Ausgabe der Type Mapping Funktion bei zwei Gruppierungsattributen und vier Aggregatfunktionen

Secondo => query plz feed groupby2[PLZ,Ort;

Secondo -> F1:fun(s1:TUPLE, i1:int) i1 + 1::1,

Secondo -> F2:fun(s2:TUPLE, i2:int) i2 + 1::2,

Secondo -> F3:fun(s3:TUPLE, i3:int) i3 + 1::3,

Secondo -> F4:fun(s4:TUPLE, i4:int) i4 + 1::4

Secondo -> ] consume;

Eingabe Type Mapping:

((stream (tuple ((PLZ int) (Ort string))))   
(PLZ Ort)   
((F1 (map (tuple ((PLZ int) (Ort string))) int int) int)   
(F2 (map (tuple ((PLZ int) (Ort string))) int int) int)   
(F3 (map (tuple ((PLZ int) (Ort string))) int int) int)   
(F4 (map (tuple ((PLZ int) (Ort string))) int int) int)))

Ergebnis Type Mapping:

(APPEND (2 1 2) (stream (tuple ((PLZ int) (Ort string) (F1 int) (F2 int) (F3 int) (F4 int)))))

### Normales und Symmetrisches Verschmelzen

Hier die beiden Vorgehensweisen und die Motivation beschreiben.

### Value Mapping

*This function computes the result of the operator applied to its arguments.*

#### Funktionsmerkmale

Der groupby2 ist ein stromverarbeitender Operator. Hier wird die Funktion des Operators in Pseudocode beschrieben.

* Der Operator ist ein stromverarbeitender Operator
* Der Operator gruppiert durch Hashing der Gruppierungsattribute.  
  Eine Sortierung des Input Stream nach den Gruppierungsattributen ist nicht erforderlich.
* Falls Gruppenattribute definiert sind, werden Tupel ignoriert, für die ein oder mehrere Gruppierungsattribute nicht definiert sind. Diese Tupel gehen nicht in die Aggregation ein.
* Der Operator ist blockierend.   
  Die Hauptverarbeitung erfolgt beim REQUEST des ersten Tupel.
* Der Operator unterstützt Fortschrittsschätzung (Query Progress Estimation)
* Die Tupel des Input Stream werden nicht gespeichert. Gespeichert werden nur die Zwischenergebnisse der Gruppen.
* Die Funktionsauswertung erfolgt durch Aufruf des Query Prozessors wie beim aggregateC Operator.

#### Lösungsansatz und Datenstruktur

Für jede Gruppe von Tupeln wird ein Ergebnis Tupel, das aus den Gruppen Attributen und den Zwischenergebnissen der Aggregation besteht, in einem STL Vektor gespeichert. Diese Zwischenergebnisse ergeben sich als Folgen von Initialwerten und dem jeweils gelesenen Tupel. Die Funktionsauswertungen erfolgen durch den Secondo Query Prozessor.

Die Value Mapping Funktion verwendet Hashing für die Gruppierung der Tupel. Für jedes Input Tupel wird der Hash Wert über die Gruppierungsattribute berechnet. Das zugehörige Gruppen Tupel wird entsprechend des Hash Werts der Gruppe im jeweiligen Bucket von

Vector<Tuple\*> hBucket[99997];

gespeichert.

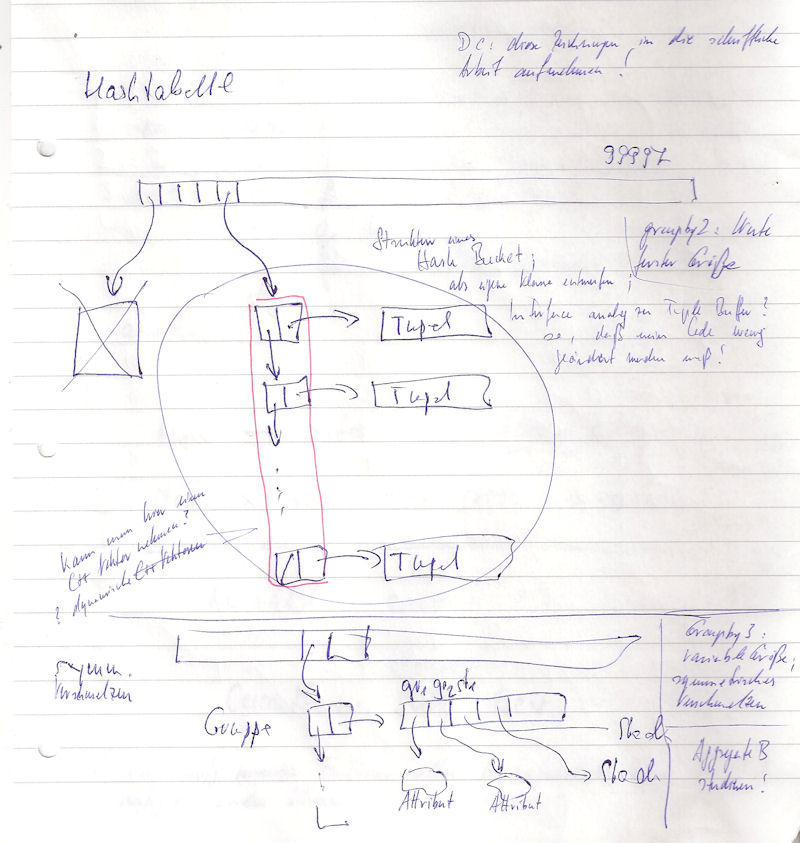


Abbildung : Datenstruktur des groupby2 Operators. Ein Array von STL Vektoren.

### Pseudocode für die Value Mapping Funktion

int GroupByValueMapping2 (Word\* args, Word& result, int message, Word& local, Supplier supplier) {

// The argument vector contains the following values:

// args[0] = input stream of tuples

// args[1] = list of grouping attributes

// args[2] = list of functions (with elements name, function, initial value)

// args[3] = number of grouping attributes (added by APPEND)

// args[4..] = positions of grouping attributes within input tuple (added by APPEND);

/\* Beispiel für drei Gruppierungsattribute: es wird APPEND (3 1 2 3) im Type Mapping erzeugt

Dies ergibt arg[3] = 3 Anzahl der Guppierungsattribute

arg[4] = 1 Index des ersten Gruppierungsattributs im Tupel

arg[5] = 2 Index des zweiten Gruppierungsattributs im Tupel

arg[6] = 3 Index des dritten Gruppierungsattributs im Tupel

\*/

switch(message) {

case OPEN:

Öffne den Input Stream und lies ein Tupel vom Stream;

If (Tupel erhalten)

Allokiere die Struktur für die lokalen Informationen;

Speichere die Tupel Adresse; erhöhe den Referenzzähler;

Erzeuge den Ergebnistyp dieses Operators;

Setzte die local Referenz auf die lokale Struktur;

Else

Setze die local Referenz Varaible auf NULL;

Return(0);

case REQUEST:

Pointer auf die lokale Struktur = local Referenz Variable;

if (erster REQUEST Aufruf)

if (keine Gruppierung angegeben)

// Berechne eine Gesamtsumme

Berechne die ersten Funktionswerte aus Tupel und Initialwerten;

Erzeuge das Ergebnis Tupel: Funktionswerte;

Lösche das Tupel des Input Stream;

Lies das nächste Tupel des Input Stream;

while (Tupel aus Input Stream erhalten)

Berechne Funktionswerte n+1 aus Ergebnis Tupel und neuem Tupel;

Schreibe die neuen Funktionswerte in das Ergebnis Tupel;

Lösche das Tupel des Input Stream;

Lies das nächste Tupel des Input Stream;

End-while

Result.addr = Adresse des Ergebnis Tupel;

Return(YIELD);

End-if // keine Gruppierung angegeben

// Führe die Aggregation mit Gruppierung durch und gib das erste Ergebnis Tupel zurück

while (Tupel aus Input Stream erhalten)

Berechne den Hash Wert des Tupel aus den Gruppenattributen;

Durchsuche das zugehörige Hash Bucket, ob es die Gruppe schon gibt;

if (Gruppe ist neu)

Berechne die ersten Funktionswerte aus Tupel und Initialwerten;

Bilde das Aggregat Tupel: Gruppierungsattribute o Funktionswerte;

Speichere das Aggregat Tupel im Hash Bucket;

Else // Gruppe kommt schon vor

Berechne die Funktionswerte n+1 aus Tupel und Funktionswerten n der  
 bestehenden Gruppe;

Schreibe die neuen Funktionswerte n+1 in das bestehende Gruppentupel;

End-if;

Lösche das Tupel des Input Stream;

Lies das nächste Tupel des Input Stream;

End-while;

// Finde das erste Ergebnis Tupel

Result.addr = NULL; // Es wurde noch kein Ergebnis Tupel gefunden

Suche das erste Hash Bucket, das ein Ergebnis Tupel enthält;

If (Hash Bucket mit Ergebnis Tupel gefunden)

Starte einen Scan dieses Hash Bucket;

Result.addr = Adresse des ersten Ergebnis Tupel;

Return(YIELD);

Else

Return(CANCEL);

End-if;

Else // folgende REQUEST Aufrufe: Ausgabe der Ergebnis Tupel

Result.addr = NULL; // zeigt an, dass noch kein Ergebnis Tupel gefunden wurde

if (keine Gruppierung angegeben)

// es gibt nur ein einziges Ergebnis Tupel, das wurde schon ausgegeben

Return(CANCEL);

End-if

If (es ist ein Hash Bucket Scan aktiv || es sind noch Hash Buckets zu verarbeiten)

// Finde das nächste Ergebnis Tupel

If (es ist ein Hash Bucket Scan aktiv)

GetNextTuple();

If (neues Tupel gefunden)

Result.addr = Adresse des neuen Tupel;

Else

Beende diesen Hash Bucket Scan;

End-if

End-if

If (result.addr == NULL) // es wurde noch kein neues Tupel gefunden

Suche das nächste Hash Bucket, das Ergebnis Tupel enthält;

If (es gibt ein solche nächstes Bucket)

Starte einen Hash Bucket Scan;

Result.addr = Adresse des neuen Tupel;

End-if

End-if

End-if

If (result.addr)

Return (YIELD);

Else

Return (CANCEL);

End-if

End-if // die Verarbeitung von REQUEST ist hier beendet

case CLOSE:

Lösche die Struktur für die lokalen Informationen und setze die local Referenz auf NULL;

schließe den Input Stream;

Return(0);

case REQUESTPROGESS:

noch offen;

case CLOSEPROGRESS:

noch offen;

Listing : Pseudo Code für die Value Mapping Funktion des groupby2 Operators

## Umgang mit Hauptspeicher, Phasen

Secondo Umgang mit Hauptspeicher, PG email:

This strategy has now been changed as follows:

\* A global parameter ~GlobalMemory~ is introduced (in SecondoConfig.ini), by default 512 MB:

# Global memory available for all operators in MB # default is 512

GlobalMemory=1024

Further, each operator using memory registers with a function

<operator>.UsesMemory() (analogous to functions such as <operator>.EnableProgress()). Hence the query processor is aware of which operators need internal memory. It distributes the available global memory equally to such operators.

\* Memory can be assigned explicitly and individually in a query to a memory using operator, using a notation {memory <size>} where size is an int specifying the amount of memory for this operator in MB. For example:

query plz100 feed sortby[Ort asc] {memory 512} consume

Hence the ~sortby~ operator can use 512 MB.

The query processor subtracts the amount of explicitly specified memory from the global amount. It then distributes the rest to the remaining operators. If the remaining amount is less than 16 MB, the operator still receives 16 MB.

When you implement an operator using main memory buffers (e.g.

TupleBuffer), the following needs to be done:

\* The operator must be registered in the algebra as using memory, e.g.:

AddOperator(&extrelsortby);

extrelsortby.SetUsesMemory();

\* The operator asks the query processor how much memory is available for it using the new method qp->GetMemorySize(s) where s is the operator's node in the operator tree (the "Supplier")

long MAX\_MEMORY = qp->GetMemorySize(s) \* 1024 \* 1024;

--------------------------------------------- DC email

inzwischen habe ich die erste Version meiner „Hauptspeicher Kontingent“ Funktionen implementiert und möchte die Hauptideen von Ihnen akzeptiert erhalten.

Das Vorgehen ist realisiert entsprechend der Email von Prof. Güting vom 24.10.2011

Als Hauptspeicher Bedarf des groupby2 Operators zähle ich im Moment nur die Gruppentupel der aktuellen Phase.  
Für die Grüße eines Tupels rufe ich tuple.GetExtSize() auf:

            // add the memory used by this tuple

            gbli->Used\_Memory += tres->GetExtSize();

Sobald die Tupel an den Folgeoperator weitergegeben sind (am Ende einer Phase), gehören Sie meiner Ansicht dem Folgeoperator und ich kann das Operatorkontingent für die nächste Phase neu verbrauchen.

Ja.

Soll ich meine GroupByLocalInfo2 in das Operator Kontingent hineinrechnen?  
Ins Gewicht fallen vor allem die Hash Buckets:   
vector<Tuple\*> hBucket[99997];   // data structure for hash buckets

Ja.

*?? Aggregation verbessert die HSP Situation aus der Sicht des gesamten Operatorbaums, da viele Tupel zu einem Gruppentupel aggregiert werden.*

## Entwicklungsumgebung

Die Software Entwicklung wurde auf dem privaten PC des Autors dieser Master Arbeit durchgeführt.

Das Gerät ist ein Hewlett-Packard PC Pavilion a6664de. Das Gerät verwendet

* AMD Phenom 9750 Quad-Core Prozessor 2,4 GHz
* 8 GB Arbeitsspeicher
* Windows 7, 64 Bit Version

Die Secondo Entwicklung erfolgte mit der Secondo VM Appliance Version 3.2. Diese virtuelle Maschine wurde betrieben mit dem VMware Player Version 4.0.1 und den zugehörigen VMware Tools. Die virtuellen Maschinen wurden mit 2 GB Hauptspeicher und zwei CPUs konfiguriert.

Für die Software Entwicklung innerhalb der VM wurde verwendet

* Ubuntu 10.04 LTS mit GNOME Desktop 2.30.2
* C++ Compiler der GNU Compiler Collection, Version ??
* GNU GDB Debugger, Version 7.1

Der Quellcode wurde mit dem Tool Concurrent Version System (CVS) mit den Servern der Fernuniversität in Hagen ausgetauscht.

## Implementierung

Wird hier der Code aufgelistet?

Kommt die Code Liste in den Anhang? Nur als CD?

Eigene Algebra oder Groupby Algebra?

## Experimentelle Auswertungen

# Schluss

Kommt am Schluss

Die Forschungsthemen sind den kommerziellen Anwendungen in Unternehmen meist weit voraus.

Hoffnung, dass sich diese Lücke etwas schließt und „moving objects databases“ mehr Eingang in die Praxis finden.

Ausblick

* Erweiterung des Secondo Optimizer für die Verwendung des groupby2 Operators.  
  Kostenvergleich der Groupby und Groupby2 Operatoren.
* Verbesserungen des Groupby2 Operator durch die Verwendung von Statistiken.  
  Bei mehreren Phasen können die häufigsten Gruppen zuerst gerechnet werden.
* Parallelisierung des Groupby2 Operators auf mehrere Systeme (Parallel Secondo).
* Verbesserung des g2 Operators durch Maßnahmen gegen Data Skew.
* Verteiltes Group By in Rechnernetzen *– Kurs zu verteilten DBs?*
* Gruppieren in memory basierten Systemen *-- Hana?*

Skew is the major danger for inferior performance of hash-based query processing algorithms, There are several ways to deal with skew (Graefe, 1993) .



For example, for an intermediate query processing result for which no statistical parameters are known a priori, the first partitioning level might have to proceed naively pretending that the partitioning hash function is perfect, but the second and further recursion levels should be able to use statistics gathered in earlier levels to ensure that each partitioning step creates even partitions, i.e. that the data is partitioned with maximal effectiveness (Graefe, 1993).

Aggregation:

RFID: Verfolgung von Logistik Strömen – *da gabs viel TD Material*

Webdaten. Social Intelligence BI Initiative – *HP Material verwenden?*

# Literaturverzeichnis

Benutzer FIVB Heroes. (21. 12 2011). *YouTube*. Von FIVB Heroes in Super Slow Motion - Ekaterina Gamova: http://www.youtube.com/watch?v=QOSh4GQ6ub0&feature=related abgerufen

Dieker, S., & Güting, R. H. (2000). Plug and Play with Query Algebras: SECONDO. A Generic DBMS Development Environment. *Proc. Int. Database Engineering and Applications Symposium* (S. 380-392). Yokohama, Japan: IEEE Computer Society.

Goetz, G. (2011, 6 29). New algorithms for join and grouping operations. *Computer Science - Research and Development* , pp. 1-25.

Graefe, G. (Juni 1993). Query evaluation techniques for large databases. *ACM Computing Surveys* , S. 73-170.

Güting, R. H., & Schneider, M. (2005). *Moving Objects Databases.* San Francisco: Elsevier.

Güting, R. H., Ansorge, D., Behr, T., Düntgen, C., Jandt, S., & Spiekermann, M. (13. 7 2011). Secondo Version 3.1. User Manual. Hagen, Deutschland.

Güting, R. H., de Almeida, V. T., Ansorge, D., Behr, T., Düntgen, C., Jandt, S., et al. (14. 07 2011). Secondo Version 3.1. Programmers Guide. Hagen, Deutschland.

Schlageter, G., Becking, D., Rosenthal, P., & Wilkes, W. (2009). *Datenbanksysteme. Kurs 1665.* Hagen: Fernuniversität in Hagen.

Schneider, M. (2009). *Implementierungskonzepte für Datenbanksysteme. Kurs 1664.* Hagen: Fernuniversität in Hagen.

Secondo Team. (21. 12 2011). *Secondo*. Von http://dna.fernuni-hagen.de/Secondo.html/index.html abgerufen