Hashbasiertes Gruppieren für allgemeine Aggregatfunktionen

Inhalt

[1 Mein Thema für die Informatik Masterarbeit 2](#_Toc311295439)

[2 Spezifikation des groupby2 Operator 3](#_Toc311295440)

[2.1 Name 3](#_Toc311295441)

[2.2 Specification 3](#_Toc311295442)

[2.3 Offene Fragen 4](#_Toc311295443)

[2.4 Type Mapping 4](#_Toc311295444)

[2.5 Value Mapping Function 5](#_Toc311295445)

[2.5.1 Lösungsansatz und Datenstruktur 6](#_Toc311295446)

[2.5.2 Pseudocode für die Value Mapping Funktion 6](#_Toc311295447)

[2.5.3 Number of Value Mappings, Value Mappings and Selection Function 9](#_Toc311295448)

[2.6 Example 9](#_Toc311295449)

[3 Formale Beschreibung der Fragestellung 10](#_Toc311295450)

[3.1 Genauere Darstellung 11](#_Toc311295451)

# Mein Thema für die Informatik Masterarbeit

Am 18.08.2011 haben Prof. Güting und ich über meine Master Arbeit gesprochen. Ich habe dieses Thema erhalten. Diese Themenbeschreibung habe ich aus dem PDF Dokument kopiert:

**Thema Nr. 22. Hashbasiertes Gruppieren für allgemeine Aggregatfunktionen.**

– Operatorsyntax wie vorher. Beim Eintreffen eines Tupels wird dieses unmittelbar im

Tupelstrom 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 Stackwerte.

– Sobald der Speicher erschöpft ist,

a. wird bei Werten fester Größe für jede neu auftretende Gruppe das Tupel nicht verarbeitet, sondern in den TupleBuffer 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 Stackwerte 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

# Spezifikation des groupby2 Operator

Die Spezifikation erfolgt entsprechend der Kapitelüberschriften in Kapitel „5.3 Operators“ des Secondo Programmes Guide. Die kursiven Abschnitte sind kopierte Texte.

## Name

Der Name des Operators ist: groupby2

## Specification

|  |  |
| --- | --- |
| **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 (Name:Funktion::Initialwert) durch Komma getrennt  **]** |
| Signature | stream(Tuple) x AttrList x  (NewAttr-1 x (Tuple x Data -> Data) x Data) ..  (NewAttr-j x (Tuple x Data -> Data) x Data)  ->  stream(Tuple(Attrlist)o Tuple([NewAttr-1: Data]..[NewAttr-j: Data]) |
| Meaning | Groups the tuples from the input stream according to the attributes of ‘AttrList’ and computes one or more aggregate values NewAttr-1 .. NewAttr-j for each group.  Function values are computed from the attributes of the input tuple, a given function parameter and an initial value. Functions here have the same capabilities as in the aggregateC operator.  The grouping attributes followed by NewAttr-1 .. NewAttr-j form the result tuples and are streamed to other operators.  The empty list is allowed for the grouping attributes. This results in a single result tuple of the form NewAttr-1 .. NewAttr-j representing the result for the complete input stream. |
| Example | query plz feed filter[.Ort = "Dresden"]  groupby2[Ort; AnzOrte: fun(t:TUPLE, i:int) i + 1::0] consume; |
| Result | Ort : Dresden  AnzOrte : 146 |

Anmerkungen:

* Neue Funktionen werden in der Syntax des aggregateC Operators angegeben. Dadurch kann innerhalb der Aggregatfunktion auf mehrere Attribute des 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.
* Ein vorhergehender Sort nach Gruppierungsattributen ist nicht erforderlich.
* Für jede Aggregatfunktion muss ein Initialwert angegeben werden.
* Sind für eine Gruppe alle Attribute von Function undefiniert, tritt die Gruppe im Ergebnis nicht auf.
* Es wird symmetrisches Verschmelzen wie in aggregateB unterstützt

## Offene Fragen

1. Wie werden Tupel behandelt, die in einigen Gruppierungsattributen undefiniert sind?  
   Vorschlag: diese Tupel werden nicht berücksichtigt und einfach übergangen.

## Type Mapping

*This function checks whether an operator can handle a certain combination of types as parameters. If so, the resulting type is returned, the symbol typeerror otherwise.*

Hier beschreibe ich die Prüfungen. Diese sollten sich aus den Prüfungen in den Operatoren groupby und aggregateC ergeben. Die Prüfungen werden inhaltlich beschrieben.

* Prüfung der Listen Struktur  
  Die Anzahl der Eingabewerte muss gleich drei sein:
  + Input Stream,
  + Liste von Gruppen Attributen
  + Liste der Aggregat Funktionen (Name:Funktion::Initialwert)

Fehlen darf nur die Liste von Gruppen Attributen.

* Test von Argument 1 (wie groupby): Input Stream  
  Test auf Tuple Stream.
* Test von Argument 2 (wie groupby): 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. Diese Prüfung erfolgt wie beim groupby Type Mapping.
* Test von Argument 3: Liste der Aggregatfunktionen (wie aggregateC Operator)  
  Die Elemente der Liste sind durch Komma getrennt.  
  Syntax der einzelnen Elemente: Funktionsname:Funktion::Initialwert  
  Es muss mindestens eine Funktion angegeben sein.   
  Jeder Funktionsname muss ein gültiger Attributname sein.  
  Kein Funktionsname darf bereits als Attribut vorkommen.  
  Jede Funktion muss genau zwei Argumente haben (eine isMap<2> sein).  
  Das erste Argument jeder Funktion muss das Tupel des Stream sein.

Der Typ jedes Initialwerts muss isDATA sein.  
Für jede Funktion muss genau ein Initialwert angegeben sein.   
Das zweite Argument jeder Funktion muss vom gleichen Typ wie der zugehörige Initialwert sein.  
Das Ergebnis jeder Funktion muss vom gleichen Typ wie der zugehörige Initialwert 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.

## Value Mapping Function

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

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

Funktionsmerkmale

* 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

Dieser Abschnitt schildert formlos die Grundidee der Implementierung im Value Mapping.

Für jede Gruppe von Tupeln wird ein Ergebnis Tupel, das aus den Gruppen Attributen und den Zwischenergebnissen der Aggregation besteht, in einem Tuple Buffer 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

TupleBuffer groupby2Bucket[99997];

gespeichert.

Anmerkung: keine Tupel Buffer verwenden, siehe Email Prof. Güting vom 25.11. Welche Objekte der Secondo Welt eignen sich als Hash Bucket?

### 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;

### Number of Value Mappings, Value Mappings and Selection Function

*If the operator has more than a single signature, a set of different value mappings are used to handle them. The count of such overloaded value mapping functions corresponds to this number.*

*This is an array containing the value mappings for all signatures covered by this operator.*

*This function decides, which of the value mappings within the value mapping array is to be used to handle a certain combination of parameter types (a certain signature). The input of this function is the same as for the type mapping. The result is the index within the value mappings array.*

Der groupby2 Operator hat genau eine Signatur.

## Example

*Within another file, <AlgebraName>.example, for each operator an example application must be given together with the result of this example. This is used in daily tests of SECONDO. Fourthermore, the example query is used to build the system relation SEC2OPERATORINFO. Note that the operator is disabled in SECONDO, if no example is present. A notice on excluded operators is prompted to the user during the startup of SECONDO. So, if you cannot use your newly implemented operator, check for that notice!*

Stand der Implementierung vom 25.11.:

Secondo => query plz feed filter[.Ort = "Dresden"] groupby2[Ort; AnzOrte: fun(t:TUPLE, i:int) i + 1::0] consume;

Ort : Dresden

AnzOrte : 146

Secondo => query plz feed groupby2[PLZ; AnzOrte: fun(t:TUPLE, i:int) i + 1::0] filter[.PLZ = 99976] consume

PLZ : 99976

AnzOrte : 15

# 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?

## Genauere Darstellung

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);