Information Systems 36 (2011) 341–358

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

θ-Constrained multi-dimensional aggregation

Michael Akinde[a], Michael H. Böhlen[b], Damianos Chatziantoniou[c], Johann Gamper[d,*]

[a] IT Department, The Norwegian Meteorological Institute, Norway
[b] Department of Computer Science, University of Zürich, Switzerland
[c] Faculty of Management Science and Technology, Athens University of Economics and Business, Greece
[d] Faculty of Computer Science, Free University of Bolzano-Bozen, Dominikanerplatz 3, 39100 Bolzano, Italy

A R T I C L E I N F O

Article history:
Received 21 March 2009
Accepted 12 July 2010
Recommended by: P. O'Neil

Keywords:
OLAP
SQL/OLAP
Window functions
Multi-dimensional aggregation

A B S T R A C T

The SQL:2003 standard introduced window functions to enhance the analytical processing capabilities of SQL. The key concept of window functions is to sort the input relation and to compute the aggregate results during a scan of the sorted relation. For multi-dimensional OLAP queries with aggregation groups defined by a general θ condition an appropriate ordering does not exist, though, and hence expensive join-based solutions are required.

In this paper we introduce θ-constrained multi-dimensional aggregation (θ-MDA), which supports multi-dimensional OLAP queries with aggregation groups defined by inequalities. θ-MDA is not based on an ordering of the data relation. Instead, the tuples that shall be considered for computing an aggregate value can be determined by a general θ condition. This facilitates the formulation of complex queries, such as multi-dimensional cumulative aggregates, which are difficult to express in SQL because no appropriate ordering exists. We present algebraic transformation rules that demonstrate how the θ-MDA interacts with other operators of a multi-set algebra. Various techniques for achieving an efficient evaluation of the θ-MDA are investigated, and we integrate them into concrete evaluation algorithms and provide cost formulas. An empirical evaluation with data from the TPC-H benchmark confirms the scalability of the θ-MDA operator and shows performance improvements of up to one order of magnitude over equivalent SQL implementations.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Problem definition and running example

On-line analytical processing (OLAP) has become a mature field with an abundance of systems and methods. It has evolved from a niche area for highly sophisticated corporations to an essential component of any modern business entity or institution. A crucial element of OLAP systems is the succinct formulation of analytical queries and their efficient evaluation. Expressing analytical queries in SQL was for a long time difficult and often resulted in prohibitive running times. The SQL:2003 standard enhanced SQL with window functions, which provide support for advanced analytical functions, including moving and cumulative aggregations. The key concept of window functions is to sort the input relation and to compute the aggregate during a scan of the sorted relation. For each row in the result a sliding window determines a contiguous range of rows over which the aggregate value for that row is computed. By having a sliding window that starts at the first row (as defined by the ordering) and ends at the current row, cumulative aggregates can be computed. For cumulative aggregations over multiple dimensions, however, such an ordering does not exist, and the tuples that contribute to an aggregation result are formed by non-contiguous rows.

Example 1. Consider the *Lineitem* relation of the TPC-H[1] benchmark, which stores information about sales orders,

* Corresponding author.
E-mail addresses: michael.akinde@met.no (M. Akinde), boehlen@ifi.uzh.ch (M.H. Böhlen), damianos@aueb.gr (D. Chatziantoniou), gamper@inf.unibz.it (J. Gamper).

[1] http://www.tpc.org/tpch/

0306-4379/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2010.07.005

Lineitem

	Ordkey	*Partkey*	*Suppkey*	*Quant*	*Price*	*Disc*	*Shipdate*
r_1	O1	P1	S1	2	220	0.00	2008.01.23
r_2	O2	P1	S1	4	440	0.05	2008.01.23
r_3	O3	P2	S1	6	300	0.10	2008.01.23
r_4	O4	P2	S2	7	420	0.10	2008.01.23
r_5	O5	P2	S1	2	100	0.00	2008.01.24
r_6	O6	P1	S2	3	240	0.05	2008.01.24
r_7	O7	P2	S1	9	450	0.05	2008.01.24
r_8	O8	P1	S2	8	640	0.10	2008.01.24

Fig. 1. Instance of *Lineitem* relation.

X

	Shipdate	*Disc*	*CntDD*	*CumCntD*	*CumCntDD*
x_1	2008.01.23	0.00	1	4	1
x_2	2008.01.23	0.05	1	4	2
x_3	2008.01.23	0.10	2	4	4
x_4	2008.01.24	0.00	1	8	2
x_5	2008.01.24	0.05	2	8	5
x_6	2008.01.24	0.10	1	8	8

Fig. 2. Result of query Q1.

each one consisting of one or more items. A tuple in the relation represents an item of an order and records various pieces of information: the order key (*Ordkey*), the line number (*Linenum*), a part key (*Partkey*), a supplier key (*Suppkey*), the quantity (*Quant*), the applied discount (*Disc*), the shipping date (*Shipdate*), etc. Fig. 1 shows a simplified instance of the *Lineitem* relation with eight orders of one lineitem each that were shipped at two consecutive days. We use this relation as a running example throughout the paper.

Consider the following query to analyze the development of the number of sales orders:

Q1: Compute the number of sales orders per day and discount rate, the cumulative number of sales orders per day, and the cumulative number of sales orders per day and discount rate.

The result of this query is shown in Fig. 2. The first two columns, *Shipdate* and *Disc*, form the base table and represent the different combinations of dates and discount rates for which aggregate values shall be computed. The other three columns represent the aggregation results. *CntDD* reports the orders per day and discount rate, *CumCntD* the cumulative number of orders by day only, and *CumCntDD* reports the cumulative number of orders by day and discount rate, i.e., the number of orders with a shipping date that is smaller or equal to the value in column *Shipdate* and where a discount rate smaller or equal to the value in column *Disc* is applied. To facilitate reading, base table and aggregate values are separated by a vertical line in the result table.

The three aggregates in Query Q1 are of different nature, in particular with respect to the aggregation groups over which the aggregate functions are evaluated. For the first aggregate, *CntDD*, the aggregation groups are defined by identical values on the grouping attributes and can be expressed with the SQL `GROUP BY` clause. The second aggregate, *CumCntD*, is a one-dimensional cumulative aggregate that can be expressed with the SQL window functions and the `UNBOUNDED PRECEDING` clause. Thus, the aggregation groups can be specified by a particular sorting of the input relation in combination with a window to select a set of contiguous rows from the sorted relation.[2] The third aggregate, *CumCntDD*, is a two-dimensional cumulative aggregate that specifies the aggregation groups along two dimensions, for which SQL provides no adequate support. Window functions are not applicable, since the *Lineitem* relation cannot be ordered such that all aggregation groups are formed by contiguous rows.

Table 1
Illustration of aggregation groups.

Aggregate value	Aggregation group
x_1.*CntDD*	$\{r_1\}$
x_3.*CntDD*	$\{r_3,r_4\}$
x_1.*CumCntD*	$\{r_1,r_2,r_3,r_4\}$
x_4.*CumCntD*	$\{r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8\}$
x_3.*CumCntDD*	$\{r_1,r_2,r_3,r_4\}$
x_5.*CumCntDD*	$\{r_1,r_2,r_5,r_6,r_7\}$

Example 2. Table 1 shows selected aggregate values of the result in Fig. 2 together with the corresponding aggregation groups, i.e., the tuples needed to compute the aggregate values. Note that if the *Lineitem* relation is sorted first by shipping date and then by discount rate as shown in Fig. 1, x_5.*CumCntDD* is computed over the non-contiguous set of tuples $\{r_1, r_2, r_5, r_6, r_7\}$. It is impossible to order the *Lineitem* relation such that the aggregation group of x_5.*CumCntDD* consists of adjacent tuples in *Lineitem* and without introducing gaps in other aggregation groups of *CumCntDD*.

Query Q1 is an example of a θ-constrained multi-dimensional aggregation query. It is multi-dimensional since the grouping is done along more than one dimension. It is θ-constrained since the aggregation groups, over which the aggregates are computed, are determined by a general θ-condition composed of non-equality conditions (e.g., $\leq$). This is different from aggregation in SQL, where the aggregation groups are either determined by equality conditions on the grouping attributes or by a window in combination with a specific ordering of the input relation. A well-known class of θ-constrained multi-dimensional aggregation queries are multi-dimensional cumulative aggregates, where aggregation values are accumulated along two or more dimensions.

Expressing θ-constrained multi-dimensional aggregation queries in SQL is difficult and requires expensive operations such as join and Cartesian product, which yields prohibitive running times. We provide an easy-to-use aggregation operator which offers advanced grouping capabilities, supports a direct and straightforward specification of aggregation groups for complex OLAP queries, and allows an efficient query evaluation.

[2] The SQL window function yields two tuples only, and an additional join is required to expand the result to six result tuples as shown in Fig. 2.

1.2. Contribution

This paper proposes a new aggregation operator, termed θ-constrained *multi-dimensional aggregation* (θ-MDA), which allows a succinct, systematic, and intuitive formulation of complex OLAP queries, where grouping is done along more than one dimension and aggregation groups are determined by non-equality conditions. Fig. 3 illustrates the new operator, which requires four arguments: the *base table B*, the *detail table R*, a list of *aggregate functions* $\vec{l}$, and a list of *grouping conditions* $\vec{\theta}$. The θ-MDA computes the *result table*, *X*, by extending each tuple of the base table, *B*, with the *aggregation results* according to the aggregate functions in $\vec{l}$. The grouping conditions, $\vec{\theta}$, determine for each result tuple and aggregate function the assigned *aggregation group*, i.e., the set of tuples from the detail table, *R*, over which the aggregate function is computed. The hatched areas indicate different aggregation groups over which the corresponding aggregate values are computed.

The θ-MDA exhibits a number of salient features. First and most importantly, the aggregation groups may consist of non-contiguous and overlapping rows in *R* as illustrated in Fig. 3. This is important since in general it is impossible to reorder *R* in such a way that all aggregation groups consist of contiguous rows for which efficient SQL solutions exist. Second, the θ-MDA does not change the row count of the base table. The number of tuples in *X* is identical to the number of tuples in *B*, and the presence of duplicates in *B* and of tuples with empty aggregation groups does not change this. In contrast, join-based solutions have to rely on non-trivial techniques that use outer joins and duplicate eliminations to handle these cases. Third, it is possible to compute the aggregate results by scanning the data relation only once. No temporary memory and no additional expensive operations such as joins or Cartesian products are required.

The technical contributions of this paper can be summarized as follows:

- We propose a new aggregation operator, termed θ-MDA, which permits aggregation groups that are defined by a general θ condition and are independent of any ordering of the data. This feature supports a succinct and systematic formulation of complex OLAP queries, such as multi-dimensional cumulative aggregates, which SQL does not support adequately.
- We show how the θ-MDA interacts with other relational algebra operators and prove algebraic transformation rules that hold for the θ-MDA operator.
- We propose a number of evaluation algorithms with cost formulas that allow an analytical comparison of query plans that include the θ-MDA.
- We report the results of an empirical evaluation of the θ-MDA. The results confirm the scalability to large data sets and show performance improvements of one order of magnitude over equivalent SQL implementations.

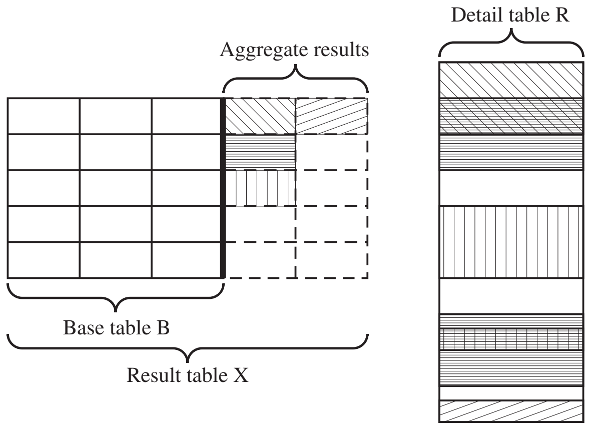

Fig. 3. Illustration of the $\theta-$MDA.

1.3. Organization

The rest of this paper is organized as follows. After the discussion of related work in Section 2, we define the θ-MDA operator in Section 3. In Section 4 we motivate our work by discussing an example query from the meteorological/oceanographic domain. Section 5 presents an effective but inefficient reduction of the θ-MDA to SQL that clarifies the expressiveness of the θ-MDA and illustrates the limitations of SQL. Section 6 formulates a number of transformation rules that show how the θ-MDA interacts with other algebraic operators. Section 7 presents a range of evaluation algorithms and cost formulas for various optimization techniques. Experimental results are described in Section 8. Section 9 concludes the paper.

2. Related work

A significant body of work [12,13,17,25,29,41,42] has considered the optimization of aggregation in the context of a unified groupby/aggregate operator. This research has focused on the reordering of groupby/aggregate operators with respect to other relational operators, primarily with selections and various forms of joins. The proposed solutions to grouping/aggregation often have difficulties to efficiently evaluate complex OLAP queries, such as multi-dimensional cumulative aggregates, since they are composed of multiple joins and aggregations [9], which leads to complex algebraic expressions.

While there has been a significant amount of research that considered data cubes and the proposed `CUBE BY` extension [23,24], very little research considered optimizations and implementations of more complex OLAP expressions. The work on the formalization and modeling of multi-dimensional databases [4,5,26,30,37] almost exclusively investigates the modeling of the data cube as well as roll-up and drill-down operations. Graefe et al. [21] proposed the `UNPIVOT` operator, which permits alternative definitions of the group. Sarawagi et al. [33] proposed SQL extensions and query processing techniques to accommodate complex data analysis for data mining.

Chatziantoniou et al. [9,10] proposed the multi-feature syntax for SQL—an extension that introduces the use of *grouping sets*. The implementation resembles that of the *nest-join* operator [14,36] for complex object models, which has been used for unnesting [15,31]. The nest-join is a generalized form of the outer join that computes the set of right operand tuples that match with each left operand tuple. The idea of grouping sets was carried over to data cubes by Ross et al. [32]. EMF-SQL (extended multi-feature syntax) [6,7,28] extends the multi-feature syntax by permitting the definition of customized aggregate conditions using the definition of grouping variables.

The SQL-99 standard [34] was extended with a number of OLAP specific features. It incorporates features such as grouping sets [9] that allow the computation of a user-controlled selection of roll-ups (instead of all roll-ups as with the `CUBE BY` operator). Another new feature is the `WINDOW` construct which defines an ordered set of data per row (partition), facilitating the formulation of moving and cumulative aggregates as well as rank aggregates. With the additional introduction of a wide variety of new aggregate functions, the SQL/OLAP amendment [35] provides a much needed improvement to the capabilities of SQL with respect to complex OLAP. While the SQL/OLAP amendment improves the capabilities of SQL significantly, the efficient evaluation of general complex OLAP queries remains an open issue. Window functions do not support multi-dimensional OLAP queries with θ-constrained aggregation groups.

The *multi-dimensional join* (MD-join) [8] is an operator for complex OLAP that combines complex group specification with complex aggregate specification in a single relational operator. Unlike the nest-join, the MD-join is an aggregation operator and was initially conceived to map the extended multi-feature syntax to relational algebra [10]. Grouping variables correspond to the aggregate specifications of this operator. The θ-MDA generalizes the MD-join and extends earlier work that describes a translation to SQL [1]. This work did not include a cost model, did not explore the differences with respect to SQL for θ-constrained multi-dimensional aggregates, and did not address the recent advances of commercial DBMS in supporting OLAP queries.

Other work on group-wise processing is described in [11,19,20]. Galindo-Legaria and Joshi [19,20] proposed two operators called *Apply* and *SegmentApply* to model parameterized query execution in an algebraic manner. Chaudhuri et al. [11] proposed the *GApply* operator in the context of XML queries, based on the Apply operator. The evaluation strategy of this operator is to partition the input tuple stream based on the grouping attributes and to execute on each group, which binds to a relation-valued variable, the per-group-query. The θ-MDA subsumes this work in a general aggregation operator.

Users can use spreadsheets to enter business data, define formulas using two-dimensional array abstractions, construct simultaneous equations with recursive models, pivot data, and compute aggregates for selected cells. Witkowski et al. [38–40] proposed spreadsheet-like computations in RDBMSs through extensions to SQL, similar in spirit to grouping sets and the SQL/OLAP amendment. They also presented optimizations, access structures, and execution models for efficient processing. Similar optimizations exist in the MD-join and θ-MDA framework.

Dittrich et al. [16] analyze the gap between OLAP and DBMS and recognized that most OLAP systems have to replicate a great deal of DBMS functionality. In order to bridge and close this gap, the authors propose to extend the relational model with new OLAP features, including the support for order, hierarchies, multi-columns, multi-rows, and multi-dimensional concepts. The θ-MDA framework is a step in this direction, since it provides a powerful, multi-dimensional aggregation operator that can be smoothly integrated into the query optimizer of any DBMS.

3. θ-Constrained multi-dimensional aggregation

This section defines the θ-constrained multi-dimensional aggregation (θ-MDA) operator. Unless stated otherwise, we assume multi-set semantics. Thus, we assume generalized algebraic operators (π, σ, $\cup$, etc.) that are consistent with SQL and operate on and return multi-sets. We use $\mathbf{B}$ to represent a database schema $(B_1, \ldots, B_k)$ and $x.\mathbf{B}$ as a shorthand to refer to $(x.B_1, \ldots, x.B_k)$. We write $E \to C$ to rename E to C. Finally, $attr(E)$ denotes the set of attributes used in E.

Definition 1 (θ-MDA *operator*). Let $B(\mathbf{B})$ and $R(\mathbf{R})$ be tables, θ_i, $1 \le i \le m$, be conditions with $attr(\theta_i) \subseteq \mathbf{B} \cup \mathbf{R}$, and $l_i = (f_{i_1}(A_{i_1}) \to C_{i_1}, \ldots f_{i_{k_i}}(A_{i_{k_i}}) \to C_{i_{k_i}})$, $1 \le i \le m$, be a list of aggregate functions over attributes $A_{i_1}, A_{i_2}, \ldots, A_{i_{k_i}}$ in $\mathbf{R}$. The θ-MDA operator is defined as

$$X = \mathcal{G}^{\theta}(B, R, (l_1, \ldots, l_m), (\theta_1, \ldots, \theta_m))$$

where $\mathbf{X} = (\mathbf{B}, C_{1_1}, \ldots, C_{1_{k_1}}, \ldots, C_{m_1}, \ldots, C_{m_{k_m}})$ is the schema of the result table and each tuple $b \in B$ produces a result tuple $x \in X$ with

- $x.\mathbf{B} = b.\mathbf{B}$.
- $x.C_{i_j} = f_{i_j}(\{r.A_{i_j} | r \in R \wedge \theta_i(b,r)\})$, for each $C_{i_j} \in \mathbf{X}$.

We call B the *base table*, R the *detail table*, and X the *result table* of the θ-MDA. For a base tuple, $b \in B$, the conditions θ_i determine the sets of detail tuples, $r \in R$, over which the aggregates f_{i_j} are evaluated. The aggregation results are the values of attributes C_{i_j} in the result relation.

Example 3. Consider Query Q1 in Example 1, which computes three different aggregates over different aggregation groups of the detail table. This query can be expressed as $\mathcal{G}^{\theta}(B, Lineitem \to L, (l_1, l_2, l_3), (\theta_1, \theta_2, \theta_3))$, where

$B : \pi[Shipdate, Discount]Lineitem$
$l_1 : (count(Quantity) \to CntDD)$
$\theta_1 : L.Shipdate = B.Shipdate \wedge L.Discount = B.Discount$
$l_2 : (count(Quantity) \to CumCntD)$
$\theta_2 : L.Shipdate \le B.Shipdate$
$l_3 : (count(Quantity) \to CumCntDD)$
$\theta_3 : L.Shipdate \le B.Shipdate \wedge L.Discount \le B.Discount$

The step-wise computation of the result is illustrated in Fig. 4. The *Lineitem* relation is processed tuple by tuple.

X

	Shipdate	*Disc*	*CntDD*	*CumCntD*	*CumCntDD*
x_1	2008.01.23	0.00	0	0	0
x_2	2008.01.23	0.05	0	0	0
x_3	2008.01.23	0.10	0	0	0
x_4	2008.01.24	0.00	0	0	0
x_5	2008.01.24	0.05	0	0	0
x_6	2008.01.24	0.10	0	0	0

Lineitem

	...	*Disc*	*Shipdate*
r_1		0.00	2008.01.23
r_2		0.05	2008.01.23
r_3		0.10	2008.01.23
r_4		0.10	2008.01.23
r_5		0.00	2008.01.24
r_6		0.05	2008.01.24
r_7		0.05	2008.01.24
r_8		0.10	2008.01.24

	Shipdate	*Disc*	*CntDD*	*CumCntD*	*CumCntDD*
x_1	2008.01.23	0.00	1	1	1
x_2	2008.01.23	0.05	0	1	1
x_3	2008.01.23	0.10	0	1	1
x_4	2008.01.24	0.00	0	1	0
x_5	2008.01.24	0.05	0	1	0
x_6	2008.01.24	0.10	0	1	0

	...	*Disc*	*Shipdate*
~~r_1~~		~~0.00~~	~~2008.01.23~~
r_2		0.05	2008.01.23
r_3		0.10	2008.01.23
r_4		0.10	2008.01.23
r_5		0.00	2008.01.24
r_6		0.05	2008.01.24
r_7		0.05	2008.01.24
r_8		0.10	2008.01.24

	Shipdate	*Disc*	*CntDD*	*CumCntD*	*CumCntDD*
x_1	2008.01.23	0.00	1	2	1
x_2	2008.01.23	0.05	1	2	2
x_3	2008.01.23	0.10	0	2	2
x_4	2008.01.24	0.00	0	2	0
x_5	2008.01.24	0.05	0	2	0
x_6	2008.01.24	0.10	0	2	0

	...	*Disc*	*Shipdate*
~~r_1~~		~~0.00~~	~~2008.01.23~~
~~r_2~~		~~0.05~~	~~2008.01.23~~
r_3		0.10	2008.01.23
r_4		0.10	2008.01.23
r_5		0.00	2008.01.24
r_6		0.05	2008.01.24
r_7		0.05	2008.01.24
r_8		0.10	2008.01.24

	Shipdate	*Disc*	*CntDD*	*CumCntD*	*CumCntDD*
x_1	2008.01.23	0.00	1	3	1
x_2	2008.01.23	0.05	1	3	2
x_3	2008.01.23	0.10	1	3	3
x_4	2008.01.24	0.00	0	3	0
x_5	2008.01.24	0.05	0	3	0
x_6	2008.01.24	0.10	0	3	0

	...	*Disc*	*Shipdate*
~~r_1~~		~~0.00~~	~~2008.01.23~~
~~r_2~~		~~0.05~~	~~2008.01.23~~
~~r_3~~		~~0.10~~	~~2008.01.23~~
r_4		0.10	2008.01.23
r_5		0.00	2008.01.24
r_6		0.05	2008.01.24
r_7		0.05	2008.01.24
r_8		0.10	2008.01.24

Fig. 4. Step-wise processing of Query Q1.

For each tuple, the three conditions are evaluated in turn and if a condition is satisfied the corresponding aggregate value in table *X* is updated. For instance, r_1 contributes to the aggregate value *CntDD* in result tuple x_1 and to the aggregate values *CumCntD* and *CumCntDD* in all result tuples.

The possibility to specify aggregation groups by a general θ condition that is independent of any ordering of the data yields an easy-to-use and expressive new operator. Note that the aggregates are computed in a single pass over the *Lineitem* table. Thus, the θ-MDA is a powerful multi-dimensional aggregation operator that calculates in one scan of the detail table multiple aggregation functions with different grouping conditions that include non-equality conditions. For this kind of OLAP query no simple and efficient SQL solutions exist, even with the use of window functions introduced in SQL:2003.

4. Application area

Complex multi-dimensional aggregation of the kind exemplified by Query Q1 becomes particularly prevalent if we look beyond the scope of traditional OLAP. In many application areas, including various scientific disciplines, huge sets of data are stored for analysis. Consider the following example taken from a meteorological/oceanographic context. Historical data about wind and wave measurements are stored in a denormalized fact table, *OceanObs*, where each tuple records the following information: the location (*Loc*) and time (*Time*) of an observation and the measured wave height (*Wa*) and wind speed (*Wi*). The *Bfr* table stores for each Beaufort number the lower (*WiL*) and upper (*WiH*) wind speed limits (Fig. 5).

A relevant query that is asked on these data (e.g., to present on a web page) is the following: *For each location and month, compute the average wave height and the*

OceanObs

	Loc	*Time*	*Wa*	*Wi*
r_1	L1	1997-01-31T18:00	5.7	20.6
r_2	L1	1997-01-31T21:00	6.4	21.1
r_3	L1	1997-02-01T00:00	8.1	23.1
r_4	L1	1997-02-17T19:00	8.2	24 5
r_5	L1	1997-02-21T12:00	8.5	23.9

Bfr

BfNr	*WiL*	*WiH*
...	...	...
8	17.2	20.7
9	20.8	24.4
10	24.5	28.4

Fig. 5. Simplified oceanographic fact table and table with Beaufort numbers.

number of gale ($BfNr = 8$), *strong gale* ($BfNr = 9$), *and storm warnings* ($BfNr = 10$) *issued in the three month period ending in the current month.*

We assume a function *YYMM* that extracts the year and month from a timestamp. With this the query can be expressed as $\mathcal{G}^\theta(B,OceanObs \to L,(l_1,l_2),(\theta_1,\theta_2))$, where

$$
\begin{aligned}
&B : \pi[Loc, YYMM(Time) \to Month, WiL, WiH](OceanObs \times \\
&\quad \sigma[8 \le BfNr \le 10]Bfr) \\
&l_1 : (sum(Wa) \to SWa, count(Wa) \to CWa) \\
&\theta_1 : L.Loc = B.Loc \wedge YYMM(L.Time) = B.Month \\
&l_2 : (count(Wi) \to CWi) \\
&\theta_2 : L.Loc = B.Loc \wedge \\
&\qquad YYMM(L.Time) \le B.Month \wedge \\
&\qquad YYMM(L.Time) > (B.Month - INTERVAL\ '3'\ MONTH) \wedge \\
&\qquad B.WiH \ge L.Wi \wedge L.Wi \ge B.WiL
\end{aligned}
$$

The result structure and the step-wise computation of this query is illustrated in Fig. 6. The first four columns represent the different combinations of groups: the location key (*Loc*), the time at the granularity of month (*Month*), and the lower (*WiL*) and upper (*WiH*) limits for the wind speeds for which the aggregates are computed. The other columns represent the aggregation result: the average wave height for each month is represented as sum (*SWa*) and count (*CWa*), and *CWi* represents the count for each wind speed interval over the last three months. As in the previous examples for Query 1, the *OceanObs* relation is processed tuple by tuple with the different conditions evaluated in turn. Only one scan is required.

Queries of the form above often involve multiple measuring parameters collected over decades at thousands of locations. Complex ad hoc analysis of scientific data is often required, but tends to be impracticable due to the limitations of the technology used. For such applications, leveraging the power of database technology and multi-dimensional aggregation would open the door to significantly faster and more flexible data storage and processing systems.

This kind of complex analysis is not exclusive to the meteorological domain; similar challenges can be found in technical diagnostics and analysis of scientific data (e.g., the analysis of medical patient data). These huge amounts of data, much of it publicly held, tends to be either impossible or very expensive to query in an ad hoc manner. Operators such as the θ-MDA would greatly facilitate the usability and accessibility of databases for the extraction of information from such data stores.

X (after initialization)

	Loc	*Month*	*WiL*	*WiH*	*SWa*	*CWa*	*CWi*
x_1	L1	1997.01	17.2	20.7	0.0	0	0
x_2	L1	1997.01	20.8	24.4	0.0	0	0
x_3	L1	1997.01	24.5	28.4	0.0	0	0
x_4	L1	1997.02	17.2	20.7	0.0	0	0
x_5	L1	1997.02	20.8	24.4	0.0	0	0
x_6	L1	1997.02	24.5	28.4	0.0	0	0

X (after processing r_1 of *OceanObs*)

	Loc	*Month*	*WiL*	*WiH*	*SWa*	*CWa*	*CWi*
x_1	L1	1997.01	17.2	20.7	5.7	1	1
x_2	L1	1997.01	20.8	24.4	5.7	1	0
x_3	L1	1997.01	24.5	28.4	5.7	1	0
x_4	L1	1997.02	17.2	20.7	0.0	0	1
x_5	L1	1997.02	20.8	24.4	0.0	0	0
x_6	L1	1997.02	24.5	28.4	0.0	0	0

X (after processing r_1 and r_2 of *OceanObs*)

	Loc	*Month*	*WiL*	*WiH*	*SWa*	*CWa*	*CWi*
x_1	L1	1997.01	17.2	20.7	12.1	2	1
x_2	L1	1997.01	20.8	24.4	12.1	2	1
x_3	L1	1997.01	24.5	28.4	12.1	2	0
x_4	L1	1997.02	17.2	20.7	0.0	0	1
x_5	L1	1997.02	20.8	24.4	0.0	0	1
x_6	L1	1997.02	24.5	28.4	0.0	0	0

Fig. 6. Step-wise processing *OceanObs* tuples.

5. Reducing the θ-MDA to SQL

In order to illustrate expressiveness and strengths of the θ-MDA operator we describe its reduction to SQL. Despite significant extensions of SQL in recent years, a comprehensive and efficient SQL reduction of the θ-MDA does not exist. However, a systematic transformation to SQL is possible by using a combination of aggregations, joins, and the CASE statement, as shown in the following proposition. The key idea is to use a generate and test approach. Specifically, a Cartesian product (or join if possible; see below) makes sure that the aggregation groups include all possible tuples. Subsequently, CASE statements inside aggregate functions are used to filter out the unwanted tuples during the aggregation. Although this yields a systematic and effective approach, it is not practical since the performance will be poor. Turning generate and test solutions into efficient algorithms is undecidable in general and we will see that DBMSs cannot optimize such statements.

Proposition 1 (*Reducing θ-MDA to SQL*). *Let $B(\mathbf{B})$ and $R(\mathbf{R})$ be base and detail table, respectively, $l_i = (f_{i_1}(A_{i_1}) \to C_{i_1}, \ldots, f_{i_{k_i}}(A_{i_{k_i}}) \to C_{i_{k_i}})$, $1 \le i \le m$, be lists of aggregate functions, and $\theta_i = \theta_C \wedge \theta'_i$, $1 \le i \le m$, be conditions that are divided into a part θ_C that is common to all conditions and an individual part θ'_i. If all tuples in B are distinct, $\mathcal{G}^\theta(B,R,(l_1,\ldots,l_m),(\theta_1,\ldots,\theta_m))$ can be reduced to the* *following SQL expression*:

$$
\begin{array}{l}
\texttt{SELECT} \\
\quad \mathbf{B}, \\
\quad f_{1_1}\ (\texttt{CASE WHEN } \theta_1 \texttt{ THEN } R.A_{1_1} \texttt{ ELSE } N_{1_1} \texttt{ END) AS } C_{i_1}, \\
\quad \ldots, \\
\quad f_{m_{k_m}}\ (\texttt{CASE WHEN } \theta_m \texttt{ THEN } R.A_{m_{k_m}} \texttt{ ELSE } N_{m_{k_m}} \texttt{ END) AS } C_{m_{k_m}} \\
\texttt{FROM} \\
\quad B \texttt{ LEFT OUTER JOIN } R \texttt{ ON } \theta_C \\
\texttt{GROUP BY} \\
\quad \mathbf{B}
\end{array}
$$

N_{i_j} is the neutral element that allows the aggregate function f_{i_j} to ignore the row (0 *for sum and count*, NULL *for min and max*).

Proposition 1 pushes the common part, θ_C, of all conditions down to the FROM clause. The left outer join on B ensures that tuples in B are preserved. If the conditions $\theta_1,\ldots,\theta_m$ are disjoint and no common expression θ_C exists, the condition of the outer-join is true, and the join expression degenerates to a Cartesian product. Note that the transformation only holds if B does not contain duplicates. If duplicates may occur in B, additional steps are required, e.g., count-based duplicate handling techniques.

Example 4. Consider Query Q1 and its θ-MDA formulation in Example 3. Since the θ conditions for the three aggregate functions are disjoint, the common part is empty (i.e., $\theta_C = \texttt{true}$), and the SQL reduction degenerates to a Cartesian product:

```
SELECT
      B.Shipdate,
      B.Discount,
      COUNT(CASE WHEN L.Shipdate = B.Shipdate AND
                      L.Discount = B.Discount
                 THEN Quantity
                 ELSE 0 END) AS CntDD,
      COUNT(CASE WHEN L.Shipdate < = B.Shipdate
                 THEN Quantity
                 ELSE 0 END) AS CumCntD,
      COUNT(CASE WHEN L.Shipdate < = B.Shipdate AND
                      L.Discount < = B.Discount
                 THEN Quantity
                 ELSE 0 END) AS CumCntDD
FROM
    (SELECT DISTINCT Shipdate, Discount FROM Lineitem) B,
      Lineitem L
GROUP BY
      B.Shipdate, B.Discount
```

Proposition 1 provides a systematic way to transform θ-MDA queries to SQL. If the θ_i conditions overlap and the common part, θ_C, is selective, the DBMS will use efficient hash or sort-merge joins. One possibility to increase the range of optimization possibilities for the DBMS is to compute all aggregates separately (using Proposition 1) and at the end join the results together. This will yield m scans of the detail table and m scans of the base table (i.e., both tables are scanned once for each aggregation list). However, since θ conditions can be pushed down to the WHERE clause the DBMS can employ efficient join processing techniques. Still the join remains a significant bottleneck in terms of the performance. Further ad hoc optimizations are possible but typically require programmer interaction as illustrated in the following example.

Example 5. We transform first the SQL query in Example 4 by computing each aggregate separately using Proposition 1, which requires six scans of the *Lineitem* relation. Then, the individual statements are optimized manually: *CntDD* can be computed by a single GROUP BY followed by an aggregation; no join is required. *CumCntD* can be computed with the help of a window function; again no join is required. *CumCntDD* is a multi-dimensional cumulative aggregate, and an inequality join must be used to compute it. The optimized SQL query is then given as follows:

```
WITH
q1 AS (
      SELECT
         L.Shipdate,
         L.Discount,
         COUNT(Quantity) AS CntDD
      FROM
         Lineitem L
      GROUP BY
         L.Shipdate, L.Discount
),
q2 AS (
      SELECT
          Shipdate,
          SUM(COUNT(Quantity)) OVER
            (ORDER BY Shipdate ROWS UNBOUNDED PRECEDING)
            AS CumCntD
      FROM
         Lineitem L
      GROUP BY
         L.Shipdate, L.Discount
),
q3 AS (
      SELECT
         B.Shipdate,
         B.Discount,
         COUNT(Quantity) AS CumCntDD
      FROM
          (SELECT DISTINCT Shipdate, Discount
           FROM Lineitem ) B
         LEFT OUTER JOIN
         (SELECT Shipdate, Discount FROM Lineitem ) L
         ON L.Shipdate < = B.Shipdate AND
            L.Discount < = B.Discount
      GROUP BY
         B.Shipdate, B.Discount
)
SELECT * FROM q1 NATURAL JOIN q2 NATURAL JOIN q3
```

In Section 8 we will show empirically that the SQL statement in Example 5 is more efficient than the one generated by Proposition 1, but still less efficient than the θ-MDA evaluation. In particular, the evaluation of the inequality join is expensive, and it is clear that the number of database scans for the optimized statement depends on the number of aggregate functions being computed, while the θ-MDA requires only one scan. Also note that DBMSs will not automatically rewrite the SQL statement in Example 4 into the SQL statement in Example 5.

Summarizing, the θ-MDA has significant advantages over equivalent SQL statements as defined by Proposition 1. First, SQL with CASE expressions are generally very hard to optimize, whereas the θ-MDA allows the application of algebraic transformations and optimization strategies (cf. Section 6). Second, although the θ-MDA evaluation is similar to join evaluation, the fact that there is no need to compute a full join and the intermediate result size does not exceed the final result size evades a lot of the performance pitfalls inherent to standard SQL join-aggregate evaluations.

6. Algebraic transformations

This section describes how the θ-MDA operator interacts with the other operators of the relational algebra by defining algebraic transformation (equivalence) rules that hold for the θ-MDA operator. The θ-MDA possesses a variety of properties that make it flexible with respect to the manipulation of algebraic expressions. Table 2 provides a summary of the transformation rules that can be used to optimize algebraic expressions. In the following we discuss representative rules. For some rules a proof is given in Appendix A.

6.1. Projections

Given a θ-MDA followed by a projection, it is possible to push down the projection to the base table, provided that the projection does not discard attributes required by the θ-MDA processing. Pushing down projections permits an early reduction of the size of the result table, X, and helps to avoid disk I/O by eagerly eliminating attribute columns from $\mathbf{B}$ that are not part of the final result.

Rule E1 states that, if the attributes of the projection, $\mathbf{A}$, contain all attributes from $\mathbf{B}$ that are used in the conditions $\theta_1, \ldots, \theta_m$, then we can push the projection to the base table, where the projection attributes $\mathbf{A}$ are replaced by $\mathbf{A}' = \mathbf{A} \backslash \mathbf{C}$ (i.e., $\mathbf{A}$ minus any aggregates computed by the θ-MDA). A proof of this rule is given in Appendix A.

Rule E2 allows to push down a projection to the base table even if $\mathbf{A}$ does not contain all attributes from $\mathbf{B}$ that are needed by the θ-MDA processing. In this case we add the $\mathbf{B}$-attributes that are used in $\vec{\theta}$ to the projection attributes, $\mathbf{A}'$, and the outer projection on the θ-MDA result remains. This transformation is useful if $\mathbf{A}'$ is smaller than $\mathbf{B}$ and the memory requirements of the result table are excessive.

6.2. Selections

Consider a θ-MDA followed by a selection with condition θ_S. If θ_S involves only attributes of the base table, i.e., $attr(\theta_S) \subseteq \mathbf{B}$, the selection can commute with the θ-MDA. This is expressed in Rule E3 and follows directly from the observation that the θ-MDA preserves the rows and attributes of the base table.

Rule E4 shows a transformation that generates a selection from the conditions of the θ-MDA. Here $\theta_i = \theta_i' \wedge \theta_i^R$, where θ_i^R contains only constraints on the detail table, i.e., $attr(\theta_i^R) \subseteq \mathbf{R}$; if no such conditions exist, $\theta_i^R = \texttt{true}$. If $\theta_i^R(r) = \texttt{false}$ for a tuple $r \in R$, that tuple is not used for the computation of the aggregate list l_i. If a tuple r neither fulfills θ_1^R nor θ_2^R nor $\ldots$ nor θ_m^R, then that tuple does not contribute to any of the aggregate lists l_1, $\ldots$, l_m. Thus, under certain conditions we can avoid a full scan of the detail relation, e.g., when R is sorted according to a timestamp and the condition constrains this timestamp. A proof of Rule E4 is given in Appendix A.

In OLAP we often want to view only tuples for which a result has been computed. This allows for further optimization rules. Let $\sigma[>0]$ represent a range check on the tuples of X, i.e., a condition that eliminates all tuples of X for which no aggregates have been updated in the θ-MDA. A typical example would be $\sigma[>0] \equiv (CNT1 > 0 \wedge \cdots \wedge CNTm > 0)$. This range check requires that each l_i contains the aggregate $count(*) \rightarrow CNTi$. Such a count aggregate can easily be added if it is not present initially. Given such a range check over the θ-MDA result, we can employ a strategy of eager selection on the tuples of the base table as expressed in Rule E5. Let $\theta_i = \theta_i' \wedge \theta_i^B$ such that $attr(\theta_i^B) \subseteq \mathbf{B}$; if no such conditions exist, $\theta_i^B = \texttt{true}$. If $\theta_i^B(b) = \texttt{false}$ for a tuple $b \in B$, no aggregate functions for tuple b and aggregate list l_i are computed, and all aggregates in l_i of b keep their initial value. Thus, if a tuple b neither fulfills θ_1^B nor θ_2^B nor $\cdots$ nor θ_m^B, no aggregates are computed for b at all.

6.3. θ-MDAs

Sequences of θ-MDAs can be commuted and coalesced if the θ-MDAs are independent. Given a nested θ-MDA sequence, $\mathcal{G}^\theta(\mathcal{G}^\theta(B,R_1,\vec{l_1},\vec{\theta_1}),R_2,\vec{l_2},\vec{\theta_2})$, the outer $\mathcal{G}^\theta$ is *independent* of the inner $\mathcal{G}^\theta$ iff $attr(\vec{\theta_2}) \subseteq (\mathbf{B} \cup \mathbf{R}_2)$. Thus, the outer θ-MDA is independent of the inner θ-MDA if the

Table 2
Transformation rules for the θ-MDA.

$\pi[\mathbf{A}]\mathcal{G}^\theta(B,R,\vec{l},\vec{\theta}) = \mathcal{G}^\theta(\pi[\mathbf{A}']B,R,\vec{l},\vec{\theta})$	$(attr(\vec{\theta}) \cap \mathbf{B}) \subseteq \mathbf{A}, \mathbf{A}' = \mathbf{A} \backslash \mathbf{C}$	(E1)
$\pi[\mathbf{A}]\mathcal{G}^\theta(B,R,\vec{l},\vec{\theta}) = \pi[\mathbf{A}]\mathcal{G}^\theta(\pi[\mathbf{A}']B,R,\vec{l},\vec{\theta})$	$\mathbf{A}' = (\mathbf{A} \cup (attr(\vec{\theta}) \cap \mathbf{B})) \backslash \mathbf{C}$	(E2)
$\sigma[\theta_S]\mathcal{G}^\theta(B,R,\vec{l},\vec{\theta}) = \mathcal{G}^\theta(\sigma[\theta_S]B,R,\vec{l},\vec{\theta})$	$attr(\theta_s) \subseteq \mathbf{B}$	(E3)
$\mathcal{G}^\theta(B,R,\vec{l},\vec{\theta}) = \mathcal{G}^\theta(B,\sigma[\theta_1^R \vee \cdots \vee \theta_m^R]R,\vec{l},\vec{\theta})$	$\theta_i = \theta_i' \wedge \theta_i^R, attr(\theta_i^R) \subseteq \mathbf{R}$	(E4)
$\sigma[>0]\mathcal{G}^\theta(B,R,\vec{l},\vec{\theta}) = \sigma[>0]\mathcal{G}^\theta(\sigma[\theta_1^B \vee \cdots \vee \theta_m^B]B,R,\vec{l},\vec{\theta})$	$\theta_i = \theta_i' \wedge \theta_i^B, attr(\theta_i^B) \subseteq \mathbf{B}$	(E5)
$\mathcal{G}^\theta(\mathcal{G}^\theta(B,R_1,\vec{l}_1,\vec{\theta}_1),R_2,\vec{l}_2,\vec{\theta}_2) = \mathcal{G}^\theta(\mathcal{G}^\theta(B,R_2,\vec{l}_2,\vec{\theta}_2),R_1,\vec{l}_1,\vec{\theta}_1)$	$attr(\vec{\theta_2}) \subseteq (\mathbf{B} \cup \mathbf{R}_2)$	(E6)
$\mathcal{G}^\theta(\mathcal{G}^\theta(B,R,\vec{l}_1,\vec{\theta}_1),R,\vec{l}_2,\vec{\theta}_2) = \mathcal{G}^\theta(B,R,(\vec{l}_1,\vec{l}_2),(\vec{\theta}_1,\vec{\theta}_2))$	$attr(\vec{\theta_2}) \subseteq (\mathbf{B} \cup \mathbf{R})$	(E7)
$\mathcal{G}^\theta(\mathcal{G}^\theta(B,R_1,\vec{l}_1,\vec{\theta}_1),R_2,\vec{l}_2,\vec{\theta}_2) = \mathcal{G}^\theta(B,R_1,\vec{l}_1,\vec{\theta}_1) \rightarrow U \bowtie_{\theta_\mathbf{B}} \mathcal{G}^\theta(B,R_2,\vec{l}_2,\vec{\theta}_2) \rightarrow V$	$attr(\vec{\theta_2}) \subseteq (\mathbf{B} \cup \mathbf{R}_2),\ \theta_\mathbf{B} = (U.\mathbf{B} = V.\mathbf{B})$	(E8)
$\mathcal{G}^\theta(B,R,\vec{l},\vec{\theta}) = \mathcal{G}^\theta(B_1,R,\vec{l},\vec{\theta}) \cup \cdots \cup \mathcal{G}^\theta(B_n,R,\vec{l},\vec{\theta})$	$B = B_1 \cup \cdots \cup B_n,\ B_i \cap B_j = \emptyset$	

θ-conditions of the outer θ-MDA are independent of the θ conditions of the inner θ-MDA; otherwise, the outer $\mathcal{G}^\theta$ is dependent on the inner $\mathcal{G}^\theta$.

Rule E6 allows the commutation of nested θ-MDAs that are independent. Rule E7 allows to coalesce two independent, nested θ-MDAs into a single θ-MDA. This transformation permits the computation of several θ-MDAs in a single pass over the detail table. The rule can also be used to split a single θ-MDA into multiple θ-MDAs, which might allow a more efficient parallel or distributed processing [3]. Coalescing and splitting together with commutation of θ-MDAs are only possible because the θ-MDA preserves the rows of the base table B.

6.4. Joins and union

Rule E8 transforms two nested independent θ-MDAs into two separate θ-MDAs connected by an equi-join. This rule only holds if B is duplicate-free with schema $\mathbf{B}=(A_1,\ldots,A_k)$, and $\theta_{\mathbf{B}}=(U.A_1=V.A_1)\wedge\cdots\wedge(U.A_k=V.A_k)$ is composed of equality conditions over the attributes in $\mathbf{B}$. Rule E8 is relevant in the context of parallel and distributed evaluation of θ-MDAs, since it allows to compute the θ-MDAs in parallel and join the results.

Rule E9 allows to replace a single θ-MDA by the union of n θ-MDAs, based on a partitioning of the base table, $B=B_1\cup\cdots\cup B_n$. This allows to develop query plans where the result tables will always reside in memory, even if the base table is large.

7. Evaluation algorithms

This section presents evaluation algorithms for the θ-MDA together with a cost analysis. We start with a simple nested loop approach for distributive (e.g., *count*, *sum*, *min*, *max*) and algebraic (e.g., *avg*) aggregates, which is then extended in various directions. The aim of the cost formulas is to characterize and compare the complexity of different evaluation algorithms. Table 3 summarizes the notation used in the cost analysis.

The cost to transfer V_i pages from disk to memory (or vice versa) through a buffer of M_j pages is given as $C_{IO}(V_i,M_j)=\lceil V_i/M_j\rceil T_k+V_i\cdot T_t$.

Table 3
Notation used in cost formulas.

V_R	Number of disk pages in detail table
V_B	Number of disk pages in base table
V_X	Number of disk pages in result table
M_X	Memory pages for the result table
M_A	Memory pages for indexes
M_I	Memory pages for the input buffer
M	Total number of memory pages ($M = M_X + M_A + M_I$)
T_k	Sum of average seek and latency times
T_t	Time for transferring a page between disk and memory
T_h	Time for hashing a page
T_j	Time for $\theta-$MDA ing a single page using a hash table
T_u	Time for updating the tuples of a page
A_j	Avg. number of pages in B a page in R matches with
C_{op}	Cost of a (complex) operation *op*

7.1. Basic evaluation algorithm

BasicTCMDA takes as input a base table B, a detail table R, (lists of) aggregate functions $(l_1,\ldots,l_m)$, and the corresponding grouping conditions $(\theta_1,\ldots,\theta_m)$. The algorithm works in four steps: it (1) replaces algebraic aggregates by their distributive sub-aggregates (e.g., *avg* is replaced by *sum* and *count*), (2) constructs the result table X from B and initializes the aggregation results, (3) scans the detail table and computes the aggregates, and (4) computes the final result by applying the super-function to the values of the sub-aggregates (e.g., divide *sum* by *count* to get *avg*).

Algorithm. BasicTCMDA(B,R,$(l_1,\ldots,l_m)$,$(\theta_1,\ldots,\theta_m)$).

```
// Step 1: Replace algebraic aggregates by distributive sub-aggregates
Let $l'_i = l_i$, $1 \le i \le m$;
foreach algebraic aggregate $f_{i_j}$ in $l'_1,\ldots,l'_m$ do
    Replace $f_{i_j}$ with its distributive sub-aggregates $f^1_{i_j},\ldots,f^{p_{i_j}}_{i_j}$;
// Step 2: Construct result table X
Construct a one tuple relation $N(\mathbf{N})$, where:
$\mathbf{N}=(C_{1_1}',\ldots,C_{m_1}',\ldots,C_{m_{k_m}}')$ and the attribute values in tuple $N$ are
the initial values of the aggregate functions (0 for sum and count,
NULL for max and min);
Let $X = B \times N$;
// Step 3: Compute the aggregates
foreach tuple $r \in R$ do
    foreach row $x \in X$ do
        foreach $\theta_i \in \{\theta_1,\ldots,\theta_m\}$ do
            if $\theta_i(x,r)$ is true then
                Update the aggregates $f_{i_1},\ldots,f_{i_{k_i}}$ in $x$;
//Step 4: Apply the super-functions
if $l_1,\ldots,l_m$ contains algebraic aggregates then
    for each row $x \in X$ do
        for each algebraic aggregate $f_{i_j}$ in $l_1,\ldots,l_m$ do
            Let $g_{i_j}$ be the super-function of $f_{i_j}$;
            In $x$, replace $f^1_{i_j},\ldots,f^{p_{i_j}}_{i_j}$ by a single column $f_{i_j}$ and set
            $x.f_{i_j} = g_{i_j}(x.f^1_{i_j},\ldots,x.f^{p_{i_j}}_{i_j})$;
return $X$;
```

The efficiency of the algorithm depends mainly on two aspects. First, only the result table, X, is kept in memory and X fits into memory. If this is not the case, the algorithm must frequently access the disk to fetch matching rows from X. (Solutions for this problem are described in Section 7.3.1.) Second, the algorithm performs only one scan of the detail table.

For the cost analysis we follow steps 2–4 of the algorithm. In the construction phase, the base table B is read and extended with one column for each aggregate. The cost formula for the construction part is

$$C_{Construct:X}=C_{Read:B}+C_{Init:X}=C_{IO}(V_B,M_X)+V_B\cdot T_u$$

The subsequent computation phase scans R and examines for each tuple $r\in R$ each row $x\in X$. If r and x satisfy a grouping condition, the corresponding aggregate values in x are updated. The cost formula for this part is

$$\begin{aligned}C_{Compute:X}&=C_{Read:R}+C_{Update:X}+C_{Write:X}\\&=C_{IO}(V_R,M_I)+V_R\cdot V_X\cdot T_u+C_{IO}(V_X,M_X)\end{aligned}$$

For the computation of the algebraic aggregates (step 4) we iterate through the tuples in X and get $C_{Algebraic:X} = V_X \cdot T_u$. The additional pass over the result table to compute the final result from the sub-aggregates assumes that the result table fits in memory.

Putting all parts together, we get the overall cost formula

$$\begin{aligned}C_{BasicTCMDA} &= C_{Construct:X} + C_{Compute:X} + C_{Algebraic:X}\\ &= C_{IO}(V_B,M_X) + V_B \cdot T_u + C_{IO}(V_R,M_I)\\ &\quad + V_R \cdot V_X \cdot T_u + C_{IO}(V_X,M_X) + V_X \cdot T_u\end{aligned}$$

7.2. Indexing the result table

BasicTCMDA serves as a base-line algorithm for the θ-MDA evaluation, but becomes expensive if the result table grows, since for each tuple $r \in R$ all rows in X are considered. This can be avoided if the result table is indexed. Then, for a tuple $r \in R$ and a condition θ we can efficiently identify the set of tuples in X that need to be updated.

Example 6. Consider the θ-MDA formulation of Q1 in Example 3. An index on *Shipdate* in the result table supports the efficient identification of the relevant result tuples for condition θ_1. The computation of the relevant result tuples for a detail tuple r proceeds as follows: use the index on *Shipdate* to identify all tuples $X' \subseteq X$ with a shipdate equal to *r.Shipdate*, and then check for these tuples the rest of the condition, i.e., $r.Discount = X'.Discount$.

Algorithm. HashTCMDA(B,R,$(l_1,\ldots,l_m)$,$(\theta_1,\ldots,\theta_m)$).

```
// Construct result table and indexes
Construct and initialize $N(\mathbf{N})$ (as in BasicTCMDA);
Let $X = B \times N$;
Build hash-indexes for $X$;
// Compute the aggregates
foreach tuple $r \in R$ do
    foreach $\theta_i \in \{\theta_1,\ldots,\theta_m\}$ do
        Fetch the rows $X_i = \{x \in X | \theta_i(x,r)\}$ using the index;
        foreach $x \in X_i$ do
            Update the aggregates $f_{i_1},\ldots,f_{i_{k_i}}$ in $x$;
return $X$;
```

The algorithm HashTCMDA extends and improves the basic θ-MDA evaluation algorithm with a result table that is augmented with one or more hash-indexes. The HashTCMDA evaluation algorithm is divided into a construction phase and a computation phase, hence $C_{HashTCMDA} = C_{Construct:X} + C_{Compute:X}$. The construction phase has now additionally to build the hash indexes on X, yielding

$$\begin{aligned}C_{Construct:X} &= C_{Read:B} + C_{Init:X} + C_{Hash:X}\\ &= C_{IO}(V_B,M_X) + V_B \cdot T_u + H_X \cdot (V_X \cdot T_h)\end{aligned}$$

where H_X is the total number of hash-indexes on X and is bounded by the number of grouping conditions, i.e., $H_X \leq m$. Since the index structures reside in memory, the relative cost is usually low compared to the cost of the other operations. In the computation phase the algorithm scans R and updates the aggregates in X, using the hash-index to efficiently determine that have to be updated. Thus, we have

$$\begin{aligned}C_{Compute:X} &= C_{Read:R} + C_{Update:X} + C_{Write:X}\\ &= C_{IO}(V_R,M_I) + V_R \cdot (H_X \cdot T_j) \cdot A_j + C_{IO}(V_X,M_X)\end{aligned}$$

7.3. Advanced memory management

Despite the use of indexes, an important aspect for an efficient evaluation is a result table that fits into main memory. Otherwise, each tuple of R might require one or more disk accesses to retrieve elements of X that are currently not in memory. Several of the transformation rules in Section 6 help to reduce the size of the result table. Here we discuss memory management strategies that ensure an efficient computation of the θ-MDA for large result tables that do not fit in memory.

7.3.1. Partitioning the result table

A solution that ensures an efficient computation of the θ-MDA regardless of the size of the base table, B, is partitioning. We can always divide B into k partitions, $B = B_1 \cup \cdots \cup B_k$, such that the result table X_i for each B_i fits in memory, i.e., $V_{X_i} \leq M_X$. Then we compute the θ-MDA for each partition in isolation and take the union of the individual results. Algorithm PartTCMDA implements this strategy.

Algorithm. PartTCMDA(B,R,$(l_1,\ldots,l_m)$,$(\theta_1,\ldots,\theta_m)$).

```
// Construct result table
Construct and initialize $N(\mathbf{N})$ (as in BasicTCMDA);
Let $X = B \times N$;
Let $k = \lceil V_X / M_X \rceil$;
Partition $X$ into $X_1,\ldots,X_k$, s.t. $V_{X_j} \leq M_X$, $1 \leq j \leq k$;
// Compute the aggregates
foreach $X_j$ where $1 \leq j \leq k$ do
    Construct hash-indexes for $X_j$;
    foreach tuple $r \in R$ do
        foreach $\theta_i \in \{\theta_1,\ldots,\theta_m\}$ do
            Fetch the rows $X_{j_i} = \{x \in X_j | \theta_i(x,r)\}$ using the index;
            foreach $x \in X_{j_i}$ do
                Update the aggregates $f_{i_1},\ldots,f_{i_{k_i}}$ in $x$;
return $X$;
```

The cost of algorithm PartTCMDA is an increase in the number of scans of R. In the construction phase we scan the result table, divide it into k partitions, and write it back to disk, yielding

$$\begin{aligned}C_{Construct:X} &= C_{Read:B} + C_{Init:X} + C_{Write:X}\\ &= C_{IO}(V_B,M_X) + V_B \cdot T_u + C_{IO}(V_X,M_X)\end{aligned}$$

In the computation phase each partition is considered in turn. If we assume that all partitions, X_j, have the same size, we get

$$\begin{aligned}C_{Compute:X} &= k \cdot (C_{Read:X_j} + C_{Hash:X_j} + C_{Read:R} + C_{Update:X_j} + C_{Write:X_j})\\ &= k \cdot (C_{IO}(V_{X_j},M_X) + H_{X_j} \cdot (V_{X_j} \cdot T_h)\\ &\quad + C_{IO}(V_R,M_X) + V_R \cdot (H_X \cdot T_j) \cdot A_j + C_{IO}(V_X,M_X))\end{aligned}$$

7.3.2. Result completion

During the evaluation of a θ-MDA, parts of the result table might become obsolete, that is, some entries in the result table will not be affected by the detail tuples that are not yet processed.

Definition 2 (*Completed result tuples*). Let $\mathcal{G}^{\theta}(B,R,(l_1,\ldots,l_m),(\theta_1,\ldots,\theta_m))$ be a θ-MDA call with result table X and detail table R, where the tuples $r_i \in R$ are processed in the following order: $r_1,\ldots,r_i,r_{i+1},\ldots,r_n$. A result tuple, $x \in X$, is *completed* after processing r_i if

$$\{r | r \in \{r_{i+1},\ldots,r_n\} \wedge (\theta_1(x,r) \vee \cdots \vee \theta_m(x,r))\} = \emptyset$$

By $Com(X,r_i)$ we denote the set of all completed tuples in X after processing r_i.

Once a result tuple is completed it can be removed from memory without affecting the correctness of the θ-MDA evaluation. To turn this definition into an optimization rule, we need syntactic criteria to identify completed tuples. While this is impossible in the general case without performing the complete computation, for a number of interesting queries such criteria exist. Two specific cases are considered next.

First, consider a θ-MDA followed by a selection with predicate P, i.e., $\sigma[P]\mathcal{G}^{\theta}(\cdots)$. If the processing of a detail tuple, $r_i \in R$, changes the result tuple, $x_j \in X$, such that the selection predicate P will return false for x_j, then tuple x_j is completed. A specific instance of this case is formalized in the following proposition.

Proposition 2 (*Range selection on θ-*MDA). *Consider the algebraic expression*

$$\sigma[=0]\mathcal{G}^{\theta}(B,R,(l_1,\ldots,l_m),(\theta_1,\ldots,\theta_m))$$

The set of completed tuples in X after processing $r_i \in R$ is defined as

$$Com(X,r_i) = \{x | x \in X \wedge (\theta_1(x,r_i) \vee \cdots \vee \theta_m(x,r_i))\}$$

The selection $\sigma[=0]$ filters the θ-MDA result with condition $CNT1 = 0 \wedge \cdots \wedge CNTm = 0$. For any tuple x in the result table, for which there exists a detail tuple r such that at least one θ_i evaluates to true, one of the counters will be greater than 0. The selection predicate will evaluate to false, and tuple x is discarded from the final result. Since no other detail tuples can affect the output of x, the tuple x is said to be completed after processing r. An important class of queries for the application of Proposition 2 is the computation of SQL subqueries using the θ-MDA [2].

The second case considers a θ-MDA call where the tuples of the detail table are sorted (clustered) on an attribute A and each grouping condition performs an equality comparison on A. Then we can determine tuple completion when transitioning to a new cluster.

Proposition 3 (*Clustered detail data*). *Let $\mathcal{G}^{\theta}(B,R,(l_1,\ldots,l_m),(\theta_1,\ldots,\theta_m))$ be a θ-*MDA *call, R be a detail table with tuples sorted on attribute A, and r_i, r_{i+1} be two tuples of R that are processed consecutively in that order. If $\theta_1,\ldots,\theta_m$ are conjunctive conditions, and each θ_j, $1 \le j \le m$, contains an equality comparison, $B.A = R.A$, the set of completed tuples is defined as*

$$Com(X,r_i) = \{x | x \in X \wedge x.A = r_i.A \wedge x.A \neq r_{i+1}.A\}$$

Proposition 3 minimizes the memory requirements for θ-MDA processing in a highly efficient way since, instead of maintaining the entire result table in memory, we maintain only the result tuples with a matching clustering attribute. If the result tuples with a matching clustering attribute fit in M_X pages the algorithm can clear the memory buffer in pace with the fetching of detail tuples from disk. Thus, the θ-MDA is computed in a single pass over the result table (and a single scan of the detail table), although the complete X exceeds the available memory (i.e., $V_X > M_X$). Note that this specific instance of completed result tuples, cf. Example 7, is a simple but effective optimization that also relational DBMSs use (a merge join with a streaming aggregate).

Example 7. Consider a simplified version of Query Q1, where only the first aggregate *CntDD* is computed, i.e., the number of sales orders per day and discount rate. This query can be formulated as $\mathcal{G}^{\theta}(B,Lineitem \to L,(l_1),(\theta_1))$, where

$$B : \pi[Shipdate,Discount]Lineitem$$
$$l_1 : (count(Quantity) \to CntDD)$$
$$\theta_1 : L.Shipdate = B.Shipdate \wedge L.Discount = B.Discount$$

Let the *Lineitem* relation be sorted first on *Shipdate* and then on *Disc* (as shown in Fig. 1) and assume that the tuples are processed in this order. Fig. 7 shows the evolution of the result table and detail table at different steps during the query evaluation when tuple completion is applied. Only the result tuples printed in bold have to be maintained in main memory.

8. Experimental studies

This section reports the results of various experimental studies that we run with a prototype implementation of the θ-MDA operator.

8.1. The θ-MDA query engine

For the experimental studies we implemented a θ-MDA engine in GNU C on top of Oracle. The query engine implements the `BasicTCMDA` algorithm, enhanced with indexes and partitioning on the result table. Depending on the θ conditions, hashing, binary search, or linear search is used. Partitioning is applied if the result table does not fit in memory. The OCI API is used to extract the base table and the detail table from the database. Simple algebraic operations (e.g., projections, selections, and grouping of the base and detail relations) that the DBMS can handle efficiently are pushed down to the DBMS. This is the case, for example, when algebraic transformations from Section 6 are applied, e.g., instead of extracting R, the θ-MDA query engine requests $\sigma[P]R$ from the DBMS.

X

	Shipdate	*Disc*	*CntDD*
x_1	2008.01.23	0.00	0
x_2	2008.01.23	0.05	0
x_3	2008.01.23	0.10	0
x_4	2008.01.24	0.00	0
x_5	2008.01.24	0.05	0
x_6	2008.01.24	0.10	0

Lineitem

	…	*Disc*	*Shipdate*
r_1		0.00	2008.01.23
r_2		0.05	2008.01.23
r_3		0.10	2008.01.23
r_4		0.10	2008.01.23
r_5		0.00	2008.01.24
r_6		0.05	2008.01.24
r_7		0.05	2008.01.24
r_8		0.10	2008.01.24

	Shipdate	*Disc*	*CntDD*
x_1	**2008.01.23**	**0.00**	**1**
x_2	2008.01.23	0.05	0
x_3	2008.01.23	0.10	0
x_4	2008.01.24	0.00	0
x_5	2008.01.24	0.05	0
x_6	2008.01.24	0.10	0

	…	*Disc*	*Shipdate*
~~r_1~~		~~0.00~~	~~2008.01.23~~
r_2		0.05	2008.01.23
r_3		0.10	2008.01.23
r_4		0.10	2008.01.23
r_5		0.00	2008.01.24
r_6		0.05	2008.01.24
r_7		0.05	2008.01.24
r_8		0.10	2008.01.24

	Shipdate	*Disc*	*CntDD*
x_1	2008.01.23	0.00	1
x_2	**2008.01.23**	**0.05**	**2**
x_3	2008.01.23	0.10	0
x_4	2008.01.24	0.00	0
x_5	2008.01.24	0.05	0
x_6	2008.01.24	0.10	0

	…	*Disc*	*Shipdate*
~~r_1~~		~~0.00~~	~~2008.01.23~~
~~r_2~~		~~0.05~~	~~2008.01.23~~
r_3		0.10	2008.01.23
r_4		0.10	2008.01.23
r_5		0.00	2008.01.24
r_6		0.05	2008.01.24
r_7		0.05	2008.01.24
r_8		0.10	2008.01.24

Fig. 7. Step-wise processing with tuple completeness.

8.2. Setup and data

The experiments were run on an Intel Pentium workstation with a 3.06 GHz processor and 1 GB main memory. The DBMS and the θ-MDA engine were on the same machine.

For the experiments we use five *Lineitem* relations of different size from the TPC-H benchmark that were created using `dbgen`: 1.3 GB (10M tuples), 2.6 GB (20M tuples), 3.9 GB (30M tuples), 5.2 GB (40M tuples), and 6.5 GB (50M tuples). We use Query Q1 of our running example, i.e., the θ-MDA formulation in Example 3, the SQL reduction in Example 4, and the optimized SQL solution in Example 5.

Unless stated otherwise we assume that the result table fits in memory. This is a realistic assumption for many OLAP applications where the result is small compared to the large input relation. We run a separate experiment to show the scalability of θ-MDA when the result table does not fit into memory and partitioning of the group table has to be applied. We did not create indexes on the *Lineitem* relation and we did not declare key attributes. Instead we rely on the indexes that are computed on the fly by the θ-MDA and Oracle's query engine, respectively.

8.3. Experiments

We report the results of a number of experiments that compare the θ-MDA evaluation with the SQL evaluation. We analyze the time and the scale-up properties in various settings and study the performance of selected optimizations discussed in Section 6.

Varying the size of the detail table and base table: Fig. 8(a) shows the processing time for Q1 with the size of the detail table varying from 1.3 GB (10M tuples) to 6.5 GB (50M tuples), and a base table with 550 tuples. The θ-MDA is more than one order of magnitude faster than both SQL solutions. The poor performance of the SQL solution that is based on Proposition 1 (SQL) is due to the join, which here degenerates to a Cartesian product and produces a large intermediate result. The highly optimized SQL solution that we discussed in Example 5 (SQLOPT) performs better, but is still far less efficient than the θ-MDA. The main reason for that are the inequality join, the four scans of the input relation, and the subsequent joins. An analysis of the query plan revealed that Oracle performs a sort-merge join for the plain SQL solution and hash joins for the optimized solution SQLOPT. The execution time of the θ-MDA increases linearly with the size of the detail table. Thus, θ-MDA scales up to large detail tables.

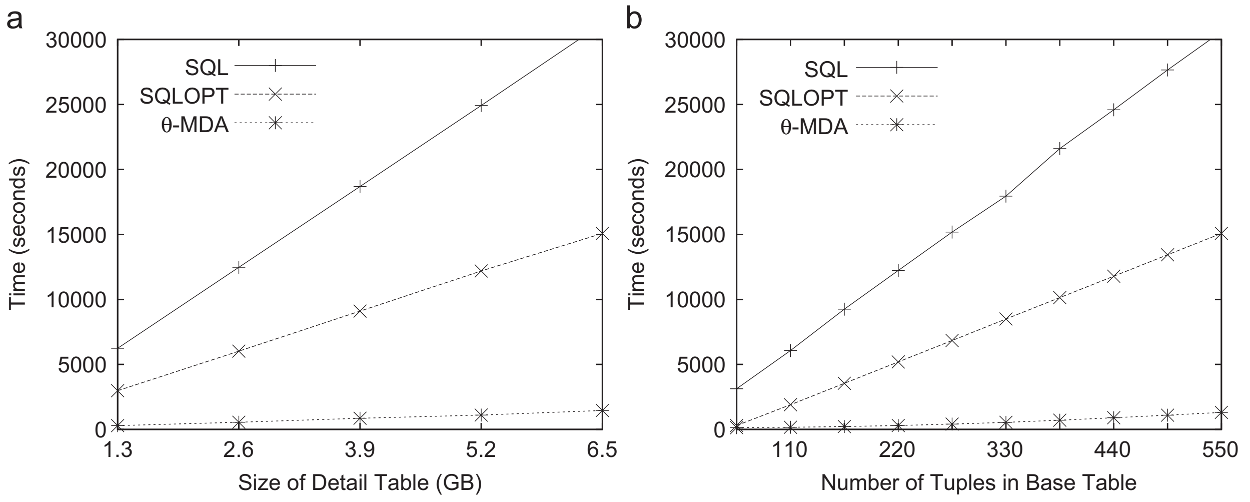

Fig. 8. Varying the size of the detail table and base table. (a) Varying detail table ($|B|$= 550 tuples). (b) Varying base table ($|R|$= 50M tuples).

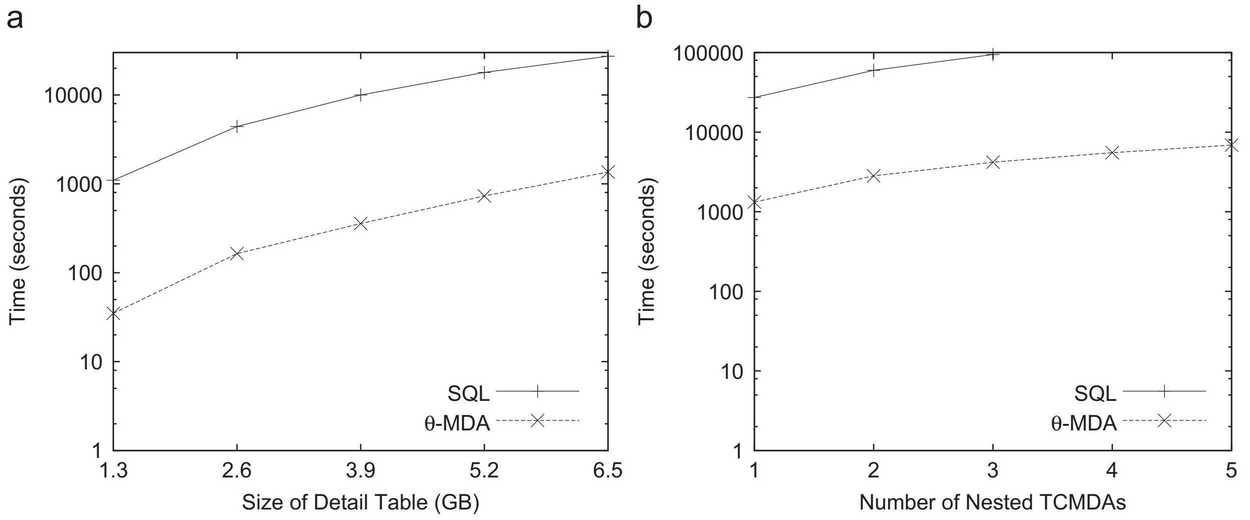

Fig. 9. Varying base table/detail table and number of nested θ-MDA calls. (a) Varying B and R. (b) Multiple nestings.

Fig. 8(b) shows the runtime for Query Q1 with a fixed detail table of 6.5 GB (50M tuples) and varying sizes of the base table. The base table varies between 55 and 550 tuples, corresponding to 0.0002% and 0.001% of the detail table, respectively. Again, the θ-MDA is the clear winner with a slow linear increase because it does not produce a large intermediate result and does not require multiple scans of the detail relation.

Varying the size of base/detail table and varying the number of nested calls: Fig. 9(a) shows the results of varying both the size of the base table and the size of the detail table. The detail table ranges from 10M to 50M tuples, and the base table ranges from 110 to 550 tuples. As before, θ-MDA is significantly faster than the SQL solution.

Fig. 9(b) demonstrates the effect of an increasing number of nested θ-MDAs to compute correlated aggregates that cannot be coalesced into a single θ-MDA. We simulate such a query by repeatedly calling Q1. Each call uses the result of the previous call as base table. The size of the detail table is 6.5 GB, and the size of the base table is 550 tuples. Similar to the findings of the previous experiments, θ-MDA clearly outperforms the SQL solution.

Partitioning the result table: Fig. 10 demonstrates the scalability of θ-MDA for a large base table that does not fit in memory, hence partitioning is applied. The horizontal axis shows the number of partitions that are used. We fixed the size of the base table and measured the evaluation from one partition (when the entire base table fits in memory) up to five partitions (when only 20% of the base table fit in memory).

Note that with large base tables it is more common to have window aggregates (e.g., aggregate over the past month; see Section 4) rather than cumulative aggregates.

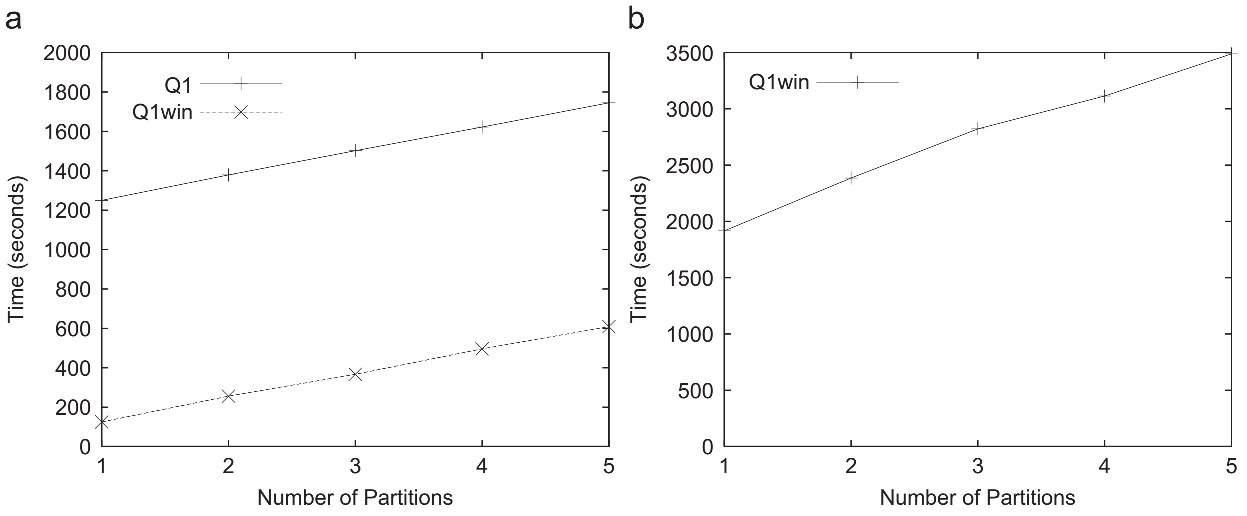

Fig. 10. Partitioning the result table. (a) Base table with 550 tuples. (b) Base table with 5M tuples.

Table 4
Detailed runtime analysis.

B-size	Empirical values				Cost Estimation					
	Init	*Hash*	*Compute*	*Total*	$C_{Read:B}$	$C_{Init:X}$	C_{Hash}	$C_{Read:R}$	$C_{Update:X}$	*Total*
1M	3	1	634	631	2.5	0.75	0.75	125	375	504
2M	7	3	969	980	5.0	1.50	1.50	125	750	883
3M	9	6	1257	1273	7.5	2.25	2.25	125	1125	1262
4M	12	8	1596	1617	10.0	3.00	3.00	125	1500	1641
5M	16	10	1895	1922	12.5	3.75	3.75	125	1875	2020

*Q*1*win* is equivalent to *Q*1, but uses a window aggregate rather than a cumulative aggregate. Specifically, the window constrains the shipdate to be within a window of 1 month, i.e., $\theta_3 \equiv L.Shipdate \leq B.Shipdate \wedge L.Shipdate \geq B.Shipdate - INTERVAL\ '1'\ MONTH \wedge L.Discount \leq B.Discount$, and similar for θ_2.

In Fig. 10(a) the base table has 550 tuples as in our previous experiments, and the detail table has 50M tuples. We force a partitioned evaluation for this setting to work out the differences between the partitioned and unpartitioned evaluation. As expected, the runtime is linearly increasing in the number of partitions, since each partition requires an additional scan of the detail table. *Q*1*win* is faster since the window selects fewer tuples from the base table that have to be updated.

In Fig. 10(b) the base table has 5M tuples, and the detail table has 50M tuples. For *Q*1*win* on average a detail tuple matches 12 base tuples (between 0 tuples and maximal 50 tuples). As before, the runtime of *Q*1*win* increases linearly with the number of partitions, since each partition requires an additional scan of the detail table.

Validation of the cost model: Table 4 demonstrates the correspondence between our cost model and the runtime values from the empirical evaluation. In the experiment we use Query *Q*1*win* with a detail table of 50M tuples, and we vary the size of the base table from 1M to 5M. The table shows in the left part the total runtime (Total) broken down into the initialization of the result table including the reading of the base table (Init), the creation of indexes (Hash), and the computation of the aggregates including the scanning of the result table (Compute). The right part shows the cost estimation using the cost formulas from Section 7.2. We assume the following parameters in the cost formulas: a block size of 400 tuples for the base table, the detail table, and the result table; a block transfer time of 0.001 s; time for updating/hashing/θ-MDAing a block of the result table $T_u = T_h = T_j = 0.0001$ s; and $H_X = 3$ indexes (one hash index and two ordered indexes).

The cost estimation follows the trend of the empirical values. Roughly the sum of the two columns $C_{Read:B}$ and $C_{Init:X}$ of the cost estimation corresponds to the column *Init* of the empirical values. Similarly, the sum of the columns $C_{Read:R}$ and $C_{Update:X}$ corresponds to the column *Compute*.

Algebraic transformations: The final experiment investigates the effects of applying selected algebraic transformations from Section 6, namely Rule E4 to push selections down to the detail table and Rule E7 to coalesce nested θ-MDAs. The same transformations have also been applied to the SQL solution.

Fig. 11(a) compares the result of evaluating Q1 on the entire detail table and when restricting the detail table by applying Rule E4. To this end, we extend the θ condition of Q1 with a term $Shipdate < \nu$ to consider only line-items with a shipdate that is smaller than value

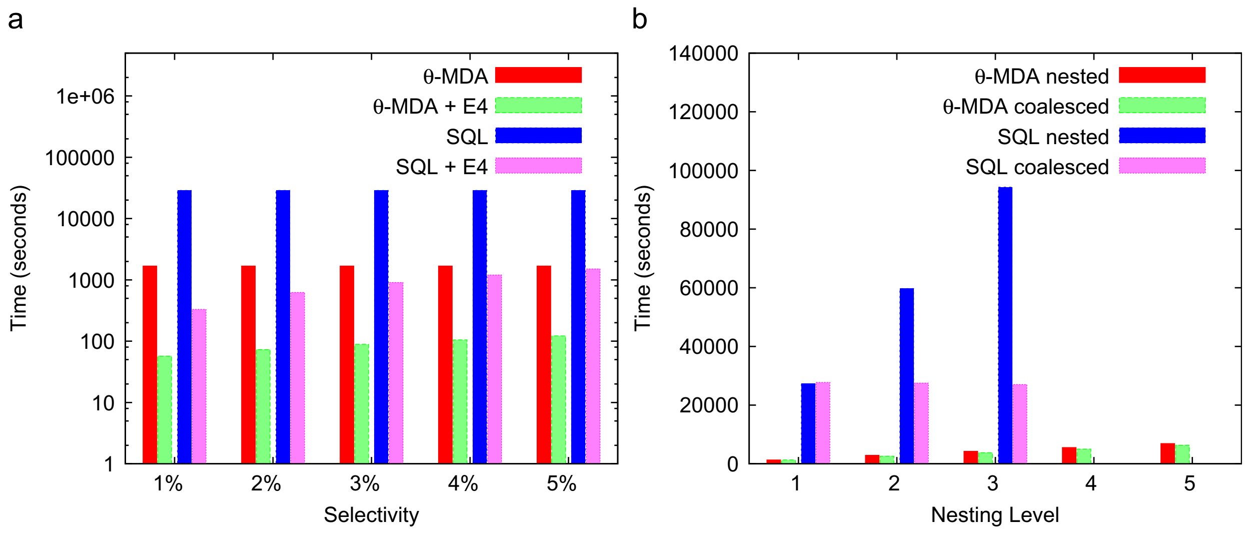

Fig. 11. Applying transformation rules. (a) Rule E4. (b) Rule E7.

v, i.e., $\theta_1 \equiv L.Shipdate = B.Shipdate \wedge L.Discount = B.Discount \wedge Shipdate < v$. The selectivity of this condition over the (full) detail table varies between 1% and 5%. The use of this transformation results in a significant reduction of the query time. Both SQL and θ-MDA take advantage of this optimization rule. In the SQL implementation the restriction on the detail table allows the DBMS to reduce the size of the join. The θ-MDA engine receives the benefit of extracting less data from the DBMS. However, without specifically rewriting the SQL query to take advantage of this transformation, the target DBMS is unable to detect the potential for this optimization rule due to the complexity of the SQL expressions.

In Fig. 11(b) we analyze transformation Rule E7 which allows to coalesce independent nested θ-MDA calls. Coalescing nested θ-MDAs is not very effective. The reason for this is that the θ-MDA evaluation has very little overhead besides the updating of aggregate values. Since for each θ condition most argument tuples contribute to a large number of base tuples, the main part of the computation costs comes from the necessary update of aggregate values in the result table. For the SQL evaluation the coalescing rule is very effective, since it reduces the query to a single Cartesian product (rather than a Cartesian product for each nesting level).

8.4. Summary

The experimental studies confirm the following main results. First, for Query Q1, which is representative for multi-dimensional analytic queries with θ-constrained aggregation groups, the θ-MDA clearly outperforms the SQL solutions in all possible settings. Second, the θ-MDA scales to large base tables (partitioning) and detail tables (one scan only). Third, the transformation rules are effective. It is interesting to note that the SQL implementation actually performs well when optimized using θ-MDA transformations; for some rules the increase in performance is comparably better for SQL than for θ-MDA. Still, θ-MDA remains more efficient than SQL.

In experiments using queries with θ-conditions that include equality conditions only (such as θ_1 in Query Q1) we found a comparable performance of the θ-MDA and the SQL solutions.

9. Conclusion

Many real-world applications require complex analysis of large data sets, e.g., the analysis of meteorological data, the diagnosis of network flows, the analysis of telecommunications records, or the analysis of cash card transactions (for fraud detection purposes) to name a few examples. However, complex OLAP queries are often difficult and expensive to evaluate using conventional query processing techniques.

This paper defines the θ-MDA, a general algebraic operator for complex OLAP queries. Different from window functions in SQL, the θ-MDA is not based on an ordering of the argument relation, over which the aggregates are computed. Instead, it allows the specification of complex grouping conditions along multiple dimensions. This provides a tremendous amount of flexibility in expressing OLAP queries, including queries for which the data cannot be sorted to apply SQL window functions. Such queries are termed multi-dimensional OLAP queries with θ-constrained aggregation groups. The θ-MDA unifies these queries in a single relational framework and facilitates the implementation of efficient and optimized query plans.

We develop and prove the correctness of a series of algebraic transformations for the θ-MDA and provide cost formulas that permit to incorporate the operator into a cost-based query optimizer. We ran a series of experiments with a prototype implementation of the operator and observed that the θ-MDA clearly outperforms equivalent SQL-based solutions.

Future research is possible in various directions. It could be interesting to identify sub-classes of θ-MDA expressions that can benefit from specialized evaluation algorithms. This would be particularly relevant for queries with very large results, e.g., data cube queries. Another direction is to study the interaction of the θ-MDA with specialized algebraic operators, such as the `apply` operator that is implemented in commercial database engines. Finally, given the formal framework, concrete algorithms, and detailed cost formulas in this paper, an interesting next step would be to integrate the θ-MDA into a full-featured query optimizer. This could be done either by implementing the θ-MDA within an extensible query optimizer (e.g., Starburst [27] or Volcano [22]) or by using user-defined routines that work directly in the database engine (e.g., DataBlades in Informix).

Appendix A. Monoid comprehension calculus and proofs

This appendix presents the proofs for the transformation Rules E1 and E4 from Section 6. For the proofs, the monoid comprehension calculus is used.

A.1. Monoid comprehension calculus

Queries in the monoid comprehension calculus (MCC) [18] are expressed as monoid comprehensions. Informally, a monoid comprehension over the monoid $\oplus$ takes the form $\oplus\{e \| q_1,\ldots,qn\}$. The merge function $\oplus$ is called the accumulator of the comprehension, and expression e is called the head of the comprehension. Each term in q_1, …, q_n is called a qualifier and can either be a generator of the form $v \leftarrow e'$, where v is a range variable and e' is an expression that constructs a collection, or a filter P, where P is a predicate. Examples of monoids are $+$, $\times$, $\cup$, $\uplus$. Monoids like $+$ and $\times$ are called primitive monoids because they construct values of a primitive type. $\cup$ and $\uplus$ are called collection monoids; $\cup$ collects values into a set, whereas $\uplus$ collects values into a multi-set.

Definition 3 (*Monoid comprehension calculus*). The monoid comprehension calculus consists of the syntactic forms in Table 5, where $\oplus$ is a monoid, e, e_1, …, e_n are terms in the monoid calculus, v is a variable, t is a monoid type, and q_1, …, q_n are qualifiers of the form $v \leftarrow e$ or e.

In the MCC, relations and tables correspond to set and multi-set expressions, and they can be collected using the $\cup$ and $\uplus$ monoids, respectively. Thus, a duplicate preserving projection, $\pi[A]R$, is written as $\uplus\{r.A \| r \leftarrow R\}$. The multi-set collection monoid collects the tuples generated by $r \leftarrow R$, and conforms them to the record projection $r.A$. For a duplicate-eliminating projection, the set collection monoid, $\cup$, is used instead. Similarly, a selection $\sigma[P]R$ is expressed as $\uplus\{r \| r \leftarrow R,P\}$. The join of two relations $R_1(A_1,A_2)$ and $R_2(A_2,A_3)$ is expressed as $\cup\{f(r_1,r_2) \| r_1 \leftarrow R_1, r_2 \leftarrow R_2, P(r_1,r_2)\}$, where P is the join predicate, and f constructs an output set element given two elements from R_1 and R_2, respectively. A join of two tables (instead of relations) is expressed by using $\uplus$ instead of $\cup$.

The monoid comprehension calculus can be put into a canonical form by a number of rewrite rules. The following are of interest for this paper:

$$\oplus\{e \| \overline{q}, v \leftarrow \otimes\{e' \| \overline{r}\}, \overline{s}\} \longrightarrow \oplus\{e \| \overline{q}, \overline{r}, v \equiv e', \overline{s}\} \tag{1}$$

This normalization rule is meaning preserving [18]. The shorthand notation $x \equiv u$ represents the binding of the variable x to the value u. The meaning of this construct is given by the following rule:

$$\oplus\{e \| \overline{r}, x \equiv u, \overline{s}\} \longrightarrow \oplus\{e[u/x] \| \overline{r}, \overline{s}[u/x]\} \tag{2}$$

$e[u/x]$ is the expression e with u substituted for all the free occurrences of x.

Definition 4 (*MCC definition of* θ-MDA). Let $\sqcup$ be our collection monoid, that is the set collection $\cup$ for relations and the multi-set collection $\uplus$ for multi-sets. The θ-MDA, $X = \mathcal{G}^{\theta}(B,R,(l_1,\ldots,l_m),(\theta_1,\ldots,\theta_m))$, where $l_i = (f_{i_1}(A_{i_1}) \rightarrow C_{i_1}, \ldots, f_{i_{k_i}}(A_{i_{k_i}}) \rightarrow C_{i_{k_i}})$, is defined in the monoid comprehension calculus as

$$\begin{aligned} X = \sqcup\{b.\mathbf{B},\ & \\ & C_{1_1} : f_{1_1}\{r.A_{1_1} \| r \leftarrow R, \theta_1\}, \\ & \ldots \\ & C_{i_j} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\}, \\ & \ldots \\ & C_{m_{k_m}} : f_{m_{k_m}}\{r.A_{m_{k_m}} \| r \leftarrow R, \theta_m\} \| b \leftarrow B\} \end{aligned}$$

Thus, for each tuple in the generator B, we compute the aggregate function f_{i_j} over all tuples in R that satisfy condition θ_i.

The following shorthand notations will be used: $\vec{l}$ for $(l_1,\ldots,l_m)$, $\vec{\theta}$ for $(\theta_1,\ldots,\theta_m)$, and $\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\}$ for $C_{1_1} : f_{1_1}\{r.A_{1_1} \| r \leftarrow R, \theta_1\}, \ldots, C_{m_{k_m}} : f_{m_{k_m}}\{r.A_{m_{k_m}} \| r \leftarrow R, \theta_m\}$. $\mathbf{C} = \{C_{1_1},\ldots,C_{m_{k_m}}\}$ denotes the aggregates computed by the θ-MDA.

A.2. Proof of Rule E1

Proof. In order to prove Rule E1 (cf. Table 2) we rewrite both sides using the MCC and show their equivalence. The left-hand side is expressed as

$$\sqcup\{x.\mathbf{A}', x.C_{i_j} \| x \leftarrow \sqcup\{b.\mathbf{A}, b.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\} \| b \leftarrow B\}$$

Let $bZ = \{b.\mathbf{A}, b.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\}\}$. By applying the rewrite rules of the MCC we transform the above

Table 5
Monoid comprehension calculus elements used in the paper.

Element	Description
NULL	Null value
c	Constant
v	Variable
$e.A$	Record projection
$\langle A_1 = e_1,\ldots,A_n = e_n \rangle$	Record construction
e_1 op e_2	op is a primitive binary function, such as $+$, $=$, $<$, $>$
$\oplus\{e \| q_1,\ldots,q_n\}$	Comprehension

expression into

$$\overset{(1)}{=} \sqcup \{x.\mathbf{A}', x.\vec{C} \| b \leftarrow B, x \equiv bZ\}$$

$$\overset{(2)}{=} \sqcup \{b.\mathbf{A}', b.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\} \| b \leftarrow B\}$$

Similarly, we can formulate the right-hand side of Rule E1 as

$$\sqcup\{x.\mathbf{A}', x.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\} \| x \leftarrow \sqcup \{b.\mathbf{A}', b \leftarrow B\}\}$$

and with $bY = \{b.\mathbf{A}'\}$ we can rewrite it as

$$\overset{(1)}{=} \sqcup \{x.\mathbf{A}', x.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\} \| b \in B, x \equiv bY\}$$

$$\overset{(2)}{=} \sqcup \{b.\mathbf{A}', b.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\} \| b \leftarrow B\}$$

This proves the equivalence of both sides of Rule E1. □

A.3. Proof of Rule E4

Proof. In order to prove Rule E4 (cf. Table 2) the left-hand side of Rule E4 is formulated in MCC as

$$\sqcup\{b.\mathbf{A}, b.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta_i\} \| b \leftarrow B\}$$

With $\theta^R = \theta_1^R \vee \cdots \vee \theta_m^R$, the right-hand side translates into

$$\sqcup\{b.\mathbf{A}, b.\vec{C} : f_{i_j}\{v.A_{i_j} \| v \leftarrow \sqcup \{r \| r \leftarrow R, \theta^R\}, \theta_i\} \| b \leftarrow B\}$$

$$\overset{(1)}{=} \sqcup \{b.\mathbf{A}, b.\vec{C} : f_{i_j}\{v.A_{i_j} \| r \leftarrow R, \theta^R, v \equiv r, \theta_i\} \| b \leftarrow B\}$$

$$\overset{(2)}{=} \sqcup \{b.\mathbf{A}, b.\vec{C} : f_{i_j}\{r.A_{i_j} \| r \leftarrow R, \theta^R, \theta_i\} \| b \leftarrow B\}$$

Now, if $\theta_i \equiv \theta^R \wedge \theta'_i$ for all $1 \le i \le m$, it follows that the left-hand and right-hand side of Rule E4 are equivalent. The essential claim is that

$$(\theta_1^R \vee \cdots \vee \theta_i^R \vee \cdots \vee \theta_m^R) \wedge \theta_i^R \wedge \theta_i' \equiv \theta_i^R \wedge \theta_i'$$

Note that $(C \vee D) \wedge C$ is equivalent to C, i.e., if C is true then $(C \vee D) \wedge C$ is true, if C is false then $(C \vee D) \wedge C$ is false. This proves the above equivalence, and hence Rule E4. □

References

[1] M.O. Akinde, M.H. Böhlen, Generalized MD-joins: evaluation and reduction to SQL, in: Databases in Telecommunications II, International Workshop Co-located with VLDB-01, Lecture Notes in Computer Science, vol. 2209, Rome, Italy, September 2001, pp. 52–67.

[2] M.O. Akinde, M.H. Böhlen, Efficient computation of subqueries in complex OLAP, in: Proceedings of the 19th International Conference on Data Engineering (ICDE'2003), Bangalore, India, March 2003, 12pp.

[3] M.O. Akinde, M.H. Böhlen, T. Johnson, L.V.S. Lakshmanan, D. Srivastava, Efficient OLAP query processing in distributed data warehouses, Information Systems 28 (1–2) (2003) 111–135.

[4] E. Baralis, S. Paraboschi, E. Teniente, Materialized views selection in a multidimensional database, in: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece, August 1997, pp. 156–165.

[5] L. Cabibbo, R. Torlone, Querying multidimensional databases, in: Database Programming Languages, Sixth International Workshop, DBPL-6, Lecture Notes in Computer Science, vol. 1369, Estes Park, Colorado, USA, August 1997, pp. 319–335.

[6] D. Chatziantoniou, Evaluation of ad hoc OLAP: in-place computation, in: 11th International Conference on Scientific and Statistical Database Management (SSDBM), Proceedings, Cleveland, Ohio, USA, July 1999, pp. 34–43.

[7] D. Chatziantoniou, The PanQ tool and EMF-SQL for complex data management, in: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, California, USA, August 1999, pp. 420–424.

[8] D. Chatziantoniou, M.O. Akinde, T. Johnson, S. Kim, MD-join: an operator for complex OLAP, in: Proceedings of the 17th International Conference on Data Engineering (ICDE'2001), Heidelberg, Germany, April 2001, pp. 524–533.

[9] D. Chatziantoniou, K.A. Ross, Querying multiple features of groups in relational databases, in: VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, Mumbai (Bombay), India, September 1996, pp. 295–306.

[10] D. Chatziantoniou, K.A. Ross, Groupwise processing of relational queries, in: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece, August 1997, pp. 476–485.

[11] S. Chaudhuri, R. Kaushik, J. Naughton, On relational support for XML publishing: beyond sorting and tagging, in: ACM SIGMOD, Conference on Management of Data, 2003, pp. 476–485.

[12] S. Chaudhuri, K. Shim, Including group-by in query optimization, in: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, September 1994, pp. 354–366.

[13] S. Chaudhuri, K. Shim, Optimizing queries with aggregate views, in: Advances in Database Technology—EDBT'96, Fifth International Conference on Extending Database Technology, Lecture Notes in Computer Science, vol. 1057, Avignon, France, March 1996, pp. 167–182.

[14] S. Cluet, G. Moerkotte, Nested queries in object bases, in: DBPL, 1993, pp. 226–242.

[15] U. Dayal, Of nests and trees: a unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers, in: VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, Brighton, England, September 1987, pp. 197–208.

[16] J.-P. Dittrich, D. Kossmann, A. Kreutz, Bridging the gap between OLAP and SQL, in: VLDB, 2005, pp. 1031–1042.

[17] R. Elmasri, S.B. Navathe, Fundamentals of Database Systems, second ed., Benjamin/Cummings Publishers, 1994.

[18] L. Fegaras, D. Maier, Optimizing object queries using an effective calculus, ACM Transactions on Database Systems 26 (4) (2000) 457–516.

[19] C. Galindo-Legaria, Parameterized queries and nesting equivalences, Technical Report, Microsoft, MSR-TR-2000-31, 2001.

[20] C.A. Galindo-Legaria, M.M. Joshi, Orthogonal optimization of subqueries and aggregation, in: VLDB'2001, Proceedings of 27th International Conference on Very Large Data Bases, Santa Barbara, California, USA, May 2001.

[21] G. Graefe, U. Fayyad, S. Chaudhuri, On the efficient gathering of sufficient statistics for classification from large SQL databases, in: Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD'98), New York City, New York, USA, August 1998, pp. 204–208.

[22] G. Graefe, W.J. McKenna, The Volcano optimizer generator: extensibility and efficient search, in: Proceedings of the Ninth International Conference on Data Engineering (ICDE'93), Vienna, Austria, April 1993, pp. 209–218.

[23] J. Gray, A. Bosworth, A. Layman, H. Pirahesh, Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total, in: Proceedings of the 12th International Conference on Data Engineering (ICDE'96), New Orleans, Louisiana, USA, February 1996, pp. 152–159.

[24] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, Datacube: a relational aggregation operator generalizing group-by cross-tab and sub-totals, Data Mining and Knowledge Discovery 1 (1) (1997) 29–53.

[25] A. Gupta, V. Harinarayan, D. Quass, Aggregate-query processing in data warehousing environments, in: VLDB'95, Proceedings of 21st International Conference on Very Large Databases, Zurich, Switzerland, September 1995, pp. 358–369.

[26] M. Gyssens, L.V.S. Lakshmanan, A foundation for multi-dimensional databases, in: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece, August 1997, pp. 106–115.

[27] L.M. Haas, W. Chang, G.M. Lohman, J. McPherson, P.F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M.J. Carey, E.J. Shekita, Starburst midflight: as the dust clears, IEEE Transactions on Knowledge and Data Engineering 2 (1) (1990) 143–160.

[28] T. Johnson, D. Chatziantoniou, Joining very large data sets, in: Databases in Telecommunications, International Workshop Co-located with VLDB-99, Lecture Notes in Computer Science, vol. 1819, Edinburgh, Scotland, UK, September 1999, pp. 170–179.

[29] A.Y. Levy, I.M. Mumick, Reasoning with aggregation constraints, in: Advances in Database Technology—EDBT'96, Fifth International Conference on Extending Database Technology, Lecture Notes in Computer Science, vol. 1057, Avignon, France, March 1996, pp. 514–534.

[30] C. Li, X.S. Wang, A data model for supporting on-line analytical processing, in: CIKM '96, Proceedings of the Fifth International Conference on Information and Knowledge Management, Rockville, Maryland, USA, November 1996, pp. 81–88.

[31] M. Muralikrishna, Improved unnesting algorithms for join aggregate SQL queries, in: VLDB'92, Proceedings of 18th International Conference on Very Large Data Bases, Vancouver, British Columbia, Canada, August 1992, pp. 91–102.

[32] K.A. Ross, D. Srivastava, D. Chatziantoniou, Complex aggregation at multiple granularities, in: Advances in Database Technology—EDBT'98, Sixth International Conference on Extending Database Technology, Lecture Notes in Computer Science, vol. 1377, Valencia, Spain, March 1998, pp. 263–277.

[33] S. Sarawagi, S. Thomas, R. Agrawal, Integrating mining with relational database systems: alternatives and implications, in: SIGMOD 1998, Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, June 1998, pp. 343–354.

[34] SQL Standards Committee—American National Standards Institute, ISO/IEC 9075:1999 Information Technology—Database Languages—SQL, 1999.

[35] SQL Standards Committee—American National Standards Institute, Information Technology—Database Languages—SQL—AMENDMENT 1: On-Line Analytical Processing (SQL/OLAP), 2001, Supplement to ISO/IEC 9075:1999.

[36] H.J. Steenhagen, P.M.G. Apers, H.M. Blanken, Optimization of nested queries in a complex object model, in: Advances in Database Technology—EDBT'94, Fourth International Conference on Extending Database Technology, Lecture Notes in Computer Science, vol. 779, Cambridge, United Kingdom, March 1994, pp. 337–350.

[37] P. Vassiliadis, Modeling multidimensional databases, cubes and cube operations, in: 10th International Conference on Scientific and Statistical Database Management, Proceedings, (SSDBM'98), Capri, Italy, July 1998, pp. 53–62.

[38] A. Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Sheng, S. Subramanian, Spreadsheets in RDBMS for OLAP, in: ACM SIGMOD, Conference on Management of Data, 2003, pp. 52–63.

[39] A. Witkowski, S. Bellamkonda, T. Bozkaya, N. Folkert, A. Gupta, J. Haydu, L. Sheng, S. Subramanian, Advanced SQL modeling in RDBMSs, ACM Transactions on Database Systems 30 (1) (2005) 83–121.

[40] A. Witkowski, S. Bellamkonda, T. Bozkaya, A. Naimat, L. Sheng, S. Subramanian, A. Waingold, Query by excel, in: VLDB, 2005, pp. 1204–1215.

[41] W.P. Yan, P. Larson, Performing group-by before join, in: Proceedings of the 10th International Conference on Data Engineering (ICDE'1994), Houston, Texas, USA, February 1994, pp. 89–100.

[42] W.P. Yan, P. Larson, Eager aggregation and lazy aggregation, in: VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, Zurich, Switzerland, September 1995, pp. 345–357.