Describing Analytical Sessions Using a Multidimensional Algebra

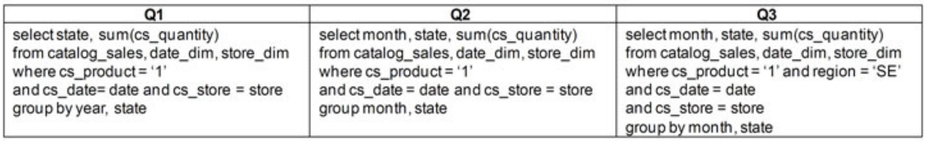
Oscar Romero[1], Patrick Marcel[2], Alberto Abelló[1], Verónika Peralta[2], and Ladjel Bellatreche[3]

[1] Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
{oromero,aabello}@essi.upc.edu
[2] Université François Rabelais de Tours, Blois, France
{patrick.marcel,veronika.peralta}@univ-tours.fr
[3] ENSMA, Poitiers, France
bellatreche@ensma.fr

Abstract. Recent efforts to support analytical tasks over relational sources have pointed out the necessity to come up with flexible, powerful means for analyzing the issued queries and exploit them in decision-oriented processes (such as query recommendation or physical tuning). Issued queries should be decomposed, stored and manipulated in a dedicated subsystem. With this aim, we present a novel approach for representing SQL analytical queries in terms of a multidimensional algebra, which better characterizes the analytical efforts of the user. In this paper we discuss how an SQL query can be formulated as a *multidimensional algebraic characterization*. Then, we discuss how to *normalize* them in order to *bridge* (i.e., collapse) several SQL queries into a single characterization (representing the analytical session), according to their logical connections.

1 Introduction

Although multidimensional (MD) databases (DBs) and OLAP are mature, there is yet a considerable amount of systems devoted to data analysis based on relational technology. Deploying a MD DB to be exploited by OLAP tools is often a long, tedious, risky and expensive process [8], which remains prohibitive for medium-sized (or even some large) companies that prefer to stand close to the well-known relational model. Furthermore, many data analysts who were constrained to learn SQL in order to conduct their analytical sessions are yet reluctant to change their modus operandi. This trend is rather evident for scientists, who are increasingly using relational databases and SQL for conducting analytical sessions over huge repositories of data [11]. For this reason, novel works have focused on supporting analytical tasks over relational sources [3,4,10,11,14,17]. Specifically, it has already been pointed out the necessity to come up with flexible, powerful means for analyzing the issued queries (the keystone of these systems, usually stored in the DB query log), and decompose, store and handle them



Q1	Q2	Q3
select state, sum(cs_quantity) from catalog_sales, date_dim, store_dim where cs_product = '1' and cs_date= date and cs_store = store group by year, state	select month, state, sum(cs_quantity) from catalog_sales, date_dim, store_dim where cs_product = '1' and cs_date = date and cs_store = store group month, state	select month, state, sum(cs_quantity) from catalog_sales, date_dim, store_dim where cs_product = '1' and region = 'SE' and cs_date = date and cs_store = store group by month, state

Fig. 1. Exemplification of three SQL analytical queries within the same session

in a dedicated subsystem in order to better support any decisional task with the knowledge captured in the analytical queries [10]. As examples of such tasks, the system should be able to support the user when formulating new queries and leverage his / her knowledge with other users (query recommendation) [3,4,11] and, at the same time, database administrators should tune their databases to cope with the evolution of queries issued (physical as well as conceptual design) [14,17].

With this spirit, in this paper we present a novel approach for analyzing the issued analytical queries and storing them in a structured way that facilitates their reuse and exploitation (e.g., understanding their semantics, comparing them, clustering into groups, etc.) in future tasks such as query recommendation or physical and conceptual design. Thus, some kind of smart, *normalized form* is required. In this sense, the relational algebra would be a candidate to characterize the input queries. However, in our approach we move a step further as it has already been discussed that the *whole* relational algebra does not properly suit for analytical queries [12]. We propose, instead, using a *multidimensional algebra*. The correspondence between both models has already been studied in the literature and it has been shown that the MD algebra is a subset of the relational one [15]. Note this is sound with the discussion introduced in [12]: the relational algebra is, simply, too expressive (in the sense it provides functionalities not needed) from an analytical point of view. Thus, the MD algebra is simpler, and we can use it to express the analytical efforts of the user in a more concise, effective way.

The rest of the paper is organized as follows. Section 2 motivates our approach and introduces some basic concepts. Section 3 discusses the related work. Section 4 explains how to characterize an analytical SQL query with the MD algebra, Section 5 presents how to normalize this characterization and Section 6 explains how to exploit it to identify analytical sessions. Section 7 concludes the paper.

2 Motivation and Basic Concepts

In our approach we propose to characterize each issued SQL query (i.e., each query in the query log) by means of the set of *MultiDimensional Algebraic* operators presented in [1] (MDA from here on).

Example 1. Consider the following scenario (inspired by the TPC-DS benchmark [18]), where a relational database is accessed with queries expressed in SQL.

The database schema consists of the following relations (where foreign keys are represented as $attr_1$ ($\rightarrow$ $attr_2$)):

```
catalog_sales (cs_date (→ date), cs_store (→ store), cs_customer (→ customer),
cs_product (→ product), cs_quantity, cs_amount),
date_dim (date, month, quarter, year),
store_dim (store, address, city, state, region),
customer_dim (customer, name, address, city, state, profession, branch),
product_dim (product, description, line)
```

Suppose queries in Fig. 1, extracted from the query log. Q1 asks for the total sales by state and year for a product, Q2 disaggregates sales by month and Q3 focuses on the south-east region. It turns out that these queries can be expressed in MDA, as evidenced in Section 4.

Characterizing an Analytical SQL Query by Means of MDA: We start by characterizing each analytical SQL query by means of MDA (see Section 4). This *Multidimensional Algebraic Characterization* (MAC from now on) forms a *tree* (like in the relational algebra, due to binary operators -such as union or drill-across, see Section 4-). The leafs are tuples *directly* retrieved from the database (i.e., the *materialized data*) and thus, we refer to them as *raw data*. Note that there is a MD schema associated with each operation of this algebraic characterization, in the sense that we know which attributes (either factual or dimensional) are in the output (further details about the MD operators can be found in Section 4). However, this characterization is not intended to be executed but to keep track of the knowledge captured in analytical queries from a MD point of view. Indeed, it is a characterization giving MD sense to the query.

Example 2. The MAC of query Q1 given in the previous example will express the roll-up (to the state and year levels), the projection over the measure (cs_quantity) and the selection (of the product with code '1').

Normalizing the MAC: Once the query has been characterized according to MDA our next step aims at *normalizing* the MAC with the objective of facilitating its comparison. To do so, it is compulsory to store each MAC in a *normalized* form. In our approach we benefit again from the algebraic structure proposed, and we use a set of *equivalence rules* (based on those of the relational algebra) to pull the MD operators up the algebraic structure, and produce a *Normalized MAC* (NMAC from now on).

Identifying Sessions: Finally, this characterization will be exploited in our last step, in which we are interested in discovering and characterizing analytical sessions. Up to now, current methods focus on isolated queries, which are analyzed on their own without considering the logical connection analytical queries from the same session do have. However, it is well-known that analytical sessions are formed of related queries capturing the reasoning flow during the analytical session [8]. Accordingly, we propose to *represent* those queries logically connected by means of a single structure capturing the whole session. We call *bridging* to the process aimed at identifying how similar two NMACs are. We consider that two queries Q1, Q2 could be coalesced in the same session if we only need to add MD operators to obtain Q2's output from Q1's NMAC. In other words, bridging is the process of producing the same result as Q2 by adding operators to Q1's NMAC (thus, we are *bridging* from Q1 to Q2). Based on the length of the bridge found, we can decide whether it makes sense or not to consider both queries in the same session.

Example 3. Consider the scenario introduced in Example 1. Query Q1 can be bridged with query Q2. Indeed, it can be detected that it corresponds to a drill-down from the year level to the month level. Furthermore, Q2 can be in turn bridged with Q3 since it corresponds to adding a selection over region. The three queries thus, should be characterized as a single session.

By iteration, we eventually obtain the whole session. Otherwise, if two queries are not similar enough as to belong to the same session, we restart the process trying to bridge the next query in the log (i.e., identify the start of a new session). We believe that capturing a whole analytical session provides a richer framework than working with isolate queries, as logical connections between queries are otherwise lost. Our contributions can be summarized as follows:

- In order to facilitate query management, we characterize each analytical SQL query as a MD query by means of the MDA, in what we call MAC.
- Next, we aim at normalizing each MAC obtained in order to facilitate its management in future steps, and obtaining its NMAC.
- Then, we bridge consecutive NMACs in order to produce a single structure for each analytical session performed.

This novel approach to describe sessions can be further exploited for diverse decisional tasks such as: (i) recommend queries to the user, (ii) create / tune the logical and physical schema and (iii) produce a MD schema. For example, MAC and bridging will be of particular interest for recommending analytical queries to the user analyzing a database or a data warehouse. To the best of our knowledge, there exists no database query recommendation technique that can construct a recommended query on-the-fly. Using the information extracted by bridging will enable recommending an OLAP operation to be applied to the user's current query (and therefore, potentially recommending queries that never *happened* in the log). Regarding conceptual MD design, this task could also benefit from NMACs. Following the idea in [16], this knowledge can be used to generate *user-oriented* MD schemas from non-MD DBs analytical logs and thus, we may, for example, decide if investing in MD technology is interesting or not for our organization. Indeed, MD DBs can benefit from analytical queries available beforehand in order to find the facts, dimensions and granularity that better suit our needs. The structural part of NMACs corresponds to the data cube the user is interested in. Thus, we could collapse in the same schema those queries whose structural part coincides. This is much more likely to happen after the normalization process, since it guarantees that all operations there are needed to align data and not just for presentation, as discussed in Section 5. Finally, queries are one of the most relevant inputs for physical design / tuning, and knowing them in detail facilitates the resolution of physical design by proposing efficient algorithms. For example, novel approaches exploit similarities between queries to generate optimized query plans. However, instead of using common intermediate results (which can be identified in our approach), common execution plans are used. Furthermore, queries are considered in isolation, although analytical queries are known to have strong logical connections. Our approach allows to tackle this problem at the session level, and inter and intra similarities can be identified between queries of each session, and exploit this information accordingly.

3 Related Work

Management of queries issued on data is rather limited in current DBMS. Most proposals are related to relational databases, which are transaction instead of query-oriented. As stressed in [10], these capabilities are reduced to query-by-example, graphical tools for composing queries, and query logging aimed at physical tuning. More elaborated management was not needed as operational systems mainly issue canned (i.e., known in advance) queries.

To our knowledge, only two works go beyond and propose a framework to manage the knowledge captured in the issued analytical queries (i.e., the DBMS log) to support query recommendation [3,4] or query completion [11] for interactive analysis of relational sources. The first approach follows the idea of recommender systems in the exploration of Web data. In this framework, the queries of a past user can serve as a guide for a new user if both have a similar querying behaviour (thus, they are interested in the same data). To do so, the authors present two approaches. The first one, presented in [4], keeps track of a matrix capturing all the tuples retrieved by each query. Later, by means of a distance function they compute how *similar* two queries are. In their second approach [3], given the time-consuming task of generating these matrixes and compare them, the authors extend their previous work to incorporate two recommendation engines, a tuple-based one that recommends queries that touch similar parts of the database, and a query fragment-based one that recommends structurally similar queries (i.e., a syntactical approach).

Similarly, [11] presents an approach to autocomplete SQL queries while the user writes them. The idea is slightly different but it follows a similar approach. The authors create a directed graph from SQL fragments (i.e., a syntactical approach). Two SQL fragments found in the same query are linked in the built graph. This graph is built in such a way that it guarantees any arbitrary navigation on the graph produces a compilable SQL query.

Finally, although capturing and exploiting this kind of knowledge should be a must for any decisional system, this issue has not been properly addressed for MD DBs and OLAP either. To our knowledge, the only works proposing

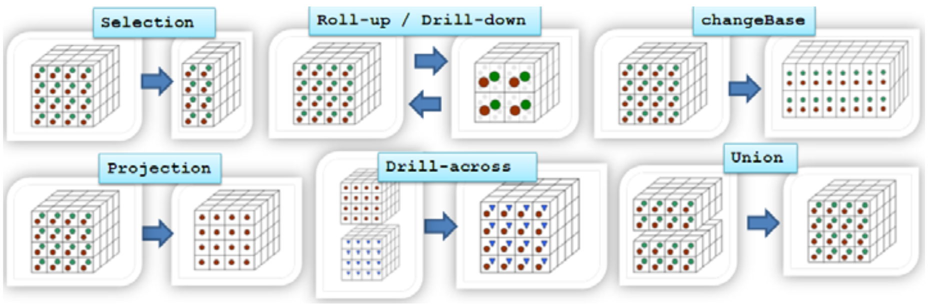


Fig. 2. Conceptual exemplification of the MDA operators

to exploit the issued queries for tackling other processes within the decisional database lifecycle are [6,7]. These works propose an approach for recommending MDX queries for MD DBs. There, the authors keep track of the tuples retrieved by each query and compute a distance function on it, in order to eventually compute their similarity. A possible reason for the lack of efforts on this topic is that we do not benefit from a standard MD language and algebra to query MD DBs. For the same reason, physical tuning depends on the underlying technology used to implement the database.

Our approach in this paper differs from all these as we do not deal with single SQL queries but with characterizations capturing the behaviour of the user all over an analytical session. Also, we take advantage of the well-known MD paradigm to characterize the input SQL queries and facilitate our task.

4 Obtaining the MAC of an SQL Query

In this section, we define a MAC using MDA. First, though, we set a MD notation and terminology. Multidimensionality is based on the **fact** / **dimension** dichotomy. The **fact**, or subject of analysis is placed in the n-dimensional space produced by the analysis dimensions. We consider a **dimension** to contain an aggregation hierarchy of **levels** representing different granularities (or levels of detail) to study data, and a **level** to contain **descriptors** (i.e., level attributes). We differentiate between identifier descriptors (univocally identifying each instance of a level) and non-identifier. In turn, a **fact** contains analysis indicators known as **measures** (which, in turn, can be regarded as fact attributes). A level of detail for each dimension produces a certain data granularity or **data cube**, in which place the measures. Finally, we denote by **base** of the space a minimal set of levels identifying univocally a certain data granularity (this definition is equivalent to *group by set* in [8]).

4.1 The Multidimensional Algebra

MDA was proven to be closed, complete (regarding the cube-query in [12]) and minimal (see [2]), and consists of the following operators (we suggest to check

Figure 2, where dots and triangles represent measures in a cell, for grasping their intuition). All the operators are unary (i.e., apply within a cube) except for drill-across and set operators, which operate over two cubes.

- **Selection** ($\sigma_p cube$): By means of a logic predicate p compound of clauses over descriptors (of the kind *descriptor operator constant*; e.g., $age >= 5$), this operator allows to choose the subset of points of interest out of the whole n-dimensional space. As a side note, MDA allows selections over descriptors even if they are not selected to identify the MD space.
- **Roll-up** ($\gamma^{level_i \rightarrow level_j}_{f(measure_1),\ldots,f(measure_n)} cube$): It groups data instances in the cube based on an aggregation hierarchy. This operator modifies the granularity of data by means of a many-to-one relationship which relates instances of two levels in the same dimension, corresponding to a part-whole relationship. As argued in [9] about drill-down (i.e., the counterpart of roll-up, represented with the same formalization but with a one-to-many relationship between $level_i$ and $level_j$), it can only be applied if we previously performed a roll-up and did not lose the correspondences between instances.
- **Projection** ($\pi_{measure_1,\ldots,measure_n} cube$): It selects a subset of measures.
- **ChangeBase** ($\chi_{base_1 \rightarrow base_2} cube$): This operator reallocates exactly the same instances of a cube into a new n-dimensional space with exactly the same number of points, by means of a one-to-one relationship. Actually, it allows to replace the current base by one of the alternatives, if more than one set of dimensions identifying the data instances (i.e., alternative bases) exist.
- **Drill-across** ($cube_1 \bowtie cube_2$): This operator fuses the measures in two cubes related by means of a one-to-one relationship. The n-dimensional space remains exactly the same, only the instances placed on it change.
- **Set Operations** ($cube_1 \Theta cube_2$): These operators allow to operate two cubes if both are defined over the same n-dimensional space. We consider union ($\cup$), difference ($\setminus$) and intersection ($\cap$). Nevertheless, from here on we focus on union, since the same considerations can be applied to the others. Also, we assume a perfect data cleaning and ETL phase (if the same cell appears in two different cubes, the values coincide). As a side note, the MDA union allows to unite two different cubes whenever they have the same schema.

The expressive power of this algebra is thoroughly discussed in [2]. Briefly, it fully matches the well-known cube-query pattern presented in [12]. For this reason, it is assumed to be expressive enough for capturing analytical efforts.

4.2 Formulating an SQL Query as a MAC

In [15], we shown how MD operators can be expressed in terms of restricted operators of the relational algebra. We take advantage of this work to identify the MDA operators, given an SQL query. First, we briefly refresh the relationship between both algebras and later, we discuss how to formulate the MAC of an SQL query. Without loss of generality, we denote by raw data (over which apply the MDA operators) the universal relationship of the tables in the FROM.

Reference Operator		"Selection"	"Projection"	"Join"	"Union"	"Group by"	"Aggregation"
Selection		$\checkmark_{Descs}$					
Projection			$\checkmark_{Measures}$				
Roll-up						$\checkmark_{Descs_{id}}$	$\checkmark_{Measures}$
Drill-across			$\checkmark_{Descs_{id}}$	$\checkmark_{Descs_{id}}$			
changeBase	**Add Dim.**			$\checkmark_{Descs_{id}}$			
	Remove Dim.		$\checkmark_{Descs_{id}}$				
	Alter Base		$\checkmark_{Descs_{id}}$	$\checkmark_{Descs_{id}}$			
Union					✓		

Fig. 3. Comparison table between the relational and MD algebras

Table 3 summarizes the mapping between both sets of algebraic operators. Note that we are considering the extended operators of the relational algebra as in [5]. We use the following notation in the table: $\checkmark_{measures}$ if the MD operator is equivalent to the relational one but it can be only applied over measures, $\checkmark_{descs}$ if the MD operator must be applied over descriptors and finally, $\checkmark_{descs_{id}}$ if it can be only applied over level identifiers. Consequently, a ✓ without restrictions means both operators are equivalent without additional restrictions. If the translation of a MD operator combines more than one relational operator, both appear ticked in the same row.

It is important to state also the constraints of the MD model that affect the usage of these operations:

1. Fact/Dimension dichotomy must be preserved, which is reflected in that descriptors and measures are disjoint.
2. Summarizability necessary conditions (as in [13]) must be preserved, which is reflected in the multiplicities of relationships used in the operations as follows:
   (a) **Roll-up**: $level_i \rightarrow level_j$ must be one-to-many (or many-to-one, if it actually corresponds to a drill-down operation).
   (b) **ChangeBase**: $base_1 \rightarrow base_2$ must be one-to-one.
   (c) **Drill-across**: $cube_1 \rightleftharpoons cube_2$ must be one-to-one.

The first item can be easily validated, whereas testing the cardinalities in the second one is reduced to discover *functional dependencies* among the set of attributes involved in the relationship, by sampling the relational source. Note that we are able to formulate the MAC by directly applying this result (we address the reader to [15] for a detailed justification of this table). In short, The query result is represented as the tree root, the source tables (i.e., raw data) as the leafs and the SQL query is decomposed as a set of MDA operators. If an SQL query cannot be fully formulated in terms of MDA operators it means that, according to MDA, it does not make MD sense and thus, it should be discarded. For example, given an analytical SQL query, any relational selection found in the WHERE clause must be read as a MD selection, and the attribute involved in such selection is known to play a dimensional role in our MAC interpretation. Similarly for the rest of operators described in the table.

Example 4. Consider Q1 from Example 1. The MAC of this query is (where raw data is the universal relation for `catalog_sales` × `store_dim` × `date_dim`, i.e., tables in the FROM):

$$\gamma^{date\rightarrow year}_{sum(cs\_quantity)}(\gamma^{store\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{cs\_product='1'}(raw\_data))))$$

Example 5. Consider now query Q4, which unites two cubes with the total sales by customer state and month, for year 1999, for the south-east and south-west regions:

```
SELECT d_month_seq as month, state, SUM(cs_quantity) AS sales
FROM catalog_sales, date_dim, customer_dim
WHERE cs_sold_date_sk = d_date_ski AND cs_customer_id = c_customer_id AND d_year = 1999
    AND region = 'SE'
GROUP BY d_month_seq, state
UNION
SELECT d_month_seq as month, state, SUM(cs_quantity) AS sales
FROM catalog_sales, date_dim, customer_dim
WHERE cs_sold_date_sk = d_date_ski AND cs_customer_id = c_customer_id AND d_year = 1999
    AND region = 'SW'
GROUP BY d_month_seq, state;
```

The MAC of this query is:

$$(\sigma_{region='SE'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data))))))$$
$$\cup$$
$$(\sigma_{region='SW'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data))))))$$

4.3 Interpreting the MAC

A MAC represents the MD counterpart of the analytical SQL statement analyzed. By definition, a MAC is a tree-shaped structure. Like in the relational algebra, this is because of binary operators. The grammar capturing its semantics is as follows (χ, σ, π, γ, $\cup$, $\bowtie$ represent the MDA operators; see Sec. 4.1):

$$\mathcal{MAC} \rightarrow \text{rawData}\ \mathcal{NP} \mid (\mathcal{MAC} \cup \mathcal{MAC})\mathcal{NP} \mid (\mathcal{MAC} \bowtie \mathcal{MAC})NP \qquad \mathcal{Q} \rightarrow \mathcal{CB}\ \mathcal{S}\ \mathcal{R}\ \mathcal{P}$$
$$\mathcal{NP} \rightarrow \mathcal{Q} \mid \mathcal{Q}\ \mathcal{NP} \quad \mathcal{CB} \rightarrow \emptyset \mid \chi\mathcal{CB} \quad \mathcal{S} \rightarrow \emptyset \mid \sigma\mathcal{S} \quad \mathcal{R} \rightarrow \emptyset \mid \gamma\mathcal{R} \qquad \mathcal{P} \rightarrow \emptyset \mid \pi\mathcal{P}$$

From now on, we will talk about the root-side and the leaf-side of the MAC. The tree leafs are raw data (i.e., with no transformations). Furthermore, we call a *navigation path* (NP from here on) to any *partially ordered set of unary operations* consecutive within the tree. These NPs can be thought as data manipulation to produce the desired presentation or alignment (i.e., the data cube MD space -*changeBases*-, slicers -*selections*-, data granularity produced -*roll-ups*- and subset of measures shown -*projections*-), whereas nodes collapsing two *branches* (from here on, we simply refer to the input NPs of binary operators as branches) are generating a new set of tuples (if desired, we may keep manipulating the result with a new NP). Thus, note that a single MAC can contain more than one NP.

Indeed, data might need to be aligned before being able to collapse them. For example, we may need to roll-up to the same granularity level before uniting

Operator	Projection	Roll-up	Selection	ChangeBase
Projection	×	✓	✓	✓
Roll-up	✓	∼	∼	∼
Selection	✓	✓	✓	∼
ChangeBase	✓	∼	∼	✓
Drill-across	✓	↗↖	↗↖	↗↖
Union	↗↖	↗↖	↗↖	↗↖

Fig. 4. MDA equivalence rules

or drilling-across data from two different cubes (i.e., align the input branches of binary operators to produce the one-to-one relationship demanded by *union* and *drill-across*).

Example 6. Consider the MAC given in Example 5. It contains two NP, namely:

$$\sigma_{region='SE'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data)))))$$

and

$$\sigma_{region='SW'}(\gamma^{date\rightarrow month}_{sum(cs\_quantity)}(\gamma^{customer\rightarrow state}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\sigma_{year=1999}(raw\_data)))))$$

Finally, we talk about the *pivotal node* as the tree node dividing the MAC into two well-differentiated layers: the *structural layer* and the *presentation layer* (i.e., the first binary ancestor of the root). In other words, the pivotal node identifies the set of tuples (i.e., the *structural* part) over which we only apply unary operations (i.e., a NP representing how data is *presented* to the user). In Example 5, the pivotal node is the union, because in this case it represents the structural part. We do not have presentation layer in this case, since it is also the root.

5 Normalizing the MAC

Once we have formulated the MAC for a given statement, we aim at normalizing it. In our approach we benefit from the algebraic structure proposed, and we use a set of equivalence rules to pull the MD operators up the algebraic structure. Thus, the MDA equivalence rules (shown in Table 4) are an immediate consequence of considering the MDA operator semantics over the relational algebra equivalence rules (explained in [5]) and considering the constraints introduced in Section 4.2. The meaning of each cell in the table is the following: if the MDA operator in the column *can be pulled up*[1] the operator in the row, the cell is ticked ("✓"). If there is a conflict, the cell is crossed ("×"). Like in the relational algebra equivalence rules, a "∼" denotes a partial conflict: the operator can be pulled up whenever the row operator does not remove the attribute needed by the column operator. For example, a selection can only be pulled up a roll-up if the attribute used to select is not rolled-up. Finally, a "↗↖" refers

[1] We recall that a MAC is a tree-shaped structure and consequently, we talk about *pulling up an operator* through the structure.

to binary operators. A unary operator can be pulled up the binary operator if it appears in both branches as explained below. For example, we can only pull up a projection through a union if the same measures are projected in both branches. Note that we assume well-formedness of MAC in the sense that no attribute is used in an operation if it is not present in the output schema of the previous operation(s).

The final aim of normalization is to distinguish between operators producing the set of tuples retrieved by the query (i.e., the structural layer) and operators manipulating these tuples before being presented to the user (i.e., the presentation layer). However, as discussed in Section 4, MACs can contain more than one NP (some of them interleaved in the structural layer for aligning binary operators), although only the root-most NP (i.e., the one amid the pivotal and root nodes) represents the presentation layer. Thus, we *normalize* MACs by *pulling operators in the NPs of the structural part to the presentation layer* (i.e., to the MAC root-side), and we do so by applying Table 4. If an operator remains stuck in the structural part after normalization then, it is needed for retrieving tuples rather than for presentation purposes.

Interestingly, note that, unlike the relational algebra logical optimization that aims at pushing operators as much as possible to the leafs, we aim at pulling MDA unary operators towards the root. Moreover, we find more ticks in Table 4 than we would find if using the relational equivalence rules and, when something needs to be checked, it is much easier, because in MDA we introduce additional constrains that simplify these rules. For example, we know that the relational selection can be pulled up a projection if the attribute involved in the selection is not projected out by the projection. Furthermore, we know that the MDA projection can only be applied to measures, whereas selection only makes sense over descriptors. Consequently, the MDA projection and selection can always be swapped in a MAC, as the set of attributes involved in each operator will always be disjoint (see Section 4 for further details). In the general case, special difficulties arise dealing with the relational group by (basis of roll-up, OLAP key operator). Interestingly, we want to remark the gain when dealing with our restricted group by (i.e., roll-up) instead of the generic one, whose difficulty is discussed in depth in [5] (where it is explicitly said that no law is stated) and specially in [19] (where the whole work is devoted to analyze all possibilities between join and group-by).

The normalization algorithm is just a postorder traversal of the MAC, considering that the nodes to visit are NPs and binary operations (thus, being a postorder algorithm, for each binary operator, it first visits its branches and later the binary operator itself). We then deal with these two kinds of nodes in a different way:

a) For each NP we visit, for each unary operator it contains (from root-side to leafs-side), we pull it up in the direction of the root as much as possible within the NP, following the rules in the white and light gray cells of Table 4.

b) Next, for each binary operator we visit, if both left and right branches are non-empty NPs and some operation coincides in their topmost Qs (see the grammar in Section 4.3), which can be pulled up through its successors in Q according to the light and dark gray cells of Table 4, the unary operator is pulled up from both and added once at the leafs-side of the parent NP of the binary operator. Note that, every binary operator will always have a parent NP (in the trivial case, the one containing the root node). Only exception, according to Table 4, is that it is not necessary that a projection must coincide at both branches of a drill-across to be pulled up.

As a result of this algorithm, we say that a NMAC is a MAC with the following properties:

i) Many NP can appear in a MAC. NPs stuck in the structural part are needed for aligning the inputs of binary operators (and not for presentation purposes).
ii) Many Q may appear at each NP, but the minimum number would be generated, each potentially containing χ, σ and γ, in this order.
iii) π can only appear in the topmost Q of every NP, and following the order imposed by the containment of attributes.

Furthermore, we force a partial order aimed at facilitating the NPs comparison during next step, as follows:

i) γ in Q will be sorted by dimension and then aggregation level.
ii) σ in Q will follow the inverse order the user posed them.
iii) χ in Q will follow the inverse order the user posed them.

Example 7. The MAC in Example 5 is not in normal form, since properties (i) (many operations can be pulled up through the union), (ii) (operations in the NPs are not sorted properly) and (iii) (the projections are not in the topmost position) do not hold. Following an postorder traversal, we would first visit both NPs (case (a)), which would be sorted to result in:

$$\pi_{cs\_quantity}(\gamma^{customer \rightarrow state}_{sum(cs\_quantity)}(\gamma^{date \rightarrow month}_{sum(cs\_quantity)}(\sigma_{year=1999}(\sigma_{region='SE'}(raw\_data)))))$$

and

$$\pi_{cs\_quantity}(\gamma^{customer \rightarrow state}_{sum(cs\_quantity)}(\gamma^{date \rightarrow month}_{sum(cs\_quantity)}(\sigma_{year=1999}(\sigma_{region='SW'}(raw\_data)))))$$

Afterwards, we would visit their parent (i.e. $\cup$, case (b)), yielding the following:

$$\pi_{cs\_quantity}(\gamma^{customer \rightarrow state}_{sum(cs\_quantity)}(\gamma^{date \rightarrow month}_{sum(cs\_quantity)}(\sigma_{year=1999}(\sigma_{region='SE'}(raw\_data) \cup \sigma_{region='SW'}(raw\_data)))))$$

Finally, we should normalize the presentation layer, but it already is.

6 *Bridging* NMACs

Working with algebraic expressions under normal form makes it easier to detect if, syntactically, two expressions are similar to each other. In our context, similar

NMACs may be considered logically related from an analytical point of view, and if two NMACs are *close enough* to each other, they are considered to belong to the same analytical session. In that case, they are *coalesced* into a session and both NMACs are logically related by *annotating* their *bridging operators.* Formally, given two NMACs n_1, n_2, we say we can bridge them if by means of some MDA operators (the bridging operators), we can transform the output of n_1 into that of n_2.

In our current approach we only analyze those queries whose *structural part* coincide by comparing their *presentation layers* (both concepts have been previously introduced in Section 5). Let P_1 and P_2 be the presentation layer of n_1 and n_2, respectively, and CS_1 and CS_2 their *cube schemas.* A cube schema is a MD interpretation of the output produced by each query. According to the MDA semantics, we can characterize it as follows (see Section 4.3): (i) the set of measures (i.e., data) shown to the user; (ii) the set of dimensional attributes selected to produce the MD space at a certain granularity level and (iii) the set of slicers applied. Now, we take advantage of the MDA minimality (operators cannot be derived by composition) and closeness (their output is a cube) properties. Since MDA is close and every operator has its inverse, by definition, we can transform CS_1 into CS_2 by means of a finite set of MDA operators (in the worst case, it would entail to *undo* all the operators that lead to CS_1 and *redo* those in CS_2). Furthermore, given its minimal property, we know which operators can be applied in order to align each cube schema feature. In other words, we can split the comparison of P_1 and P_2 into smaller comparisons regarding the cube schema part affected by the MDA operators:

i) **Measures:** Let $m1_1, \ldots, m1_n$ and $m2_1, \ldots, m2_t$ the list of measures in CS_1 and CS_2, respectively.
   - If $m1_1, \ldots, m1_n$ and $m2_1, \ldots, m2_t$ coincide nothing has to be done.
   - $\forall\ m1_i \in CS_1$, s.t. $m1_i \notin CS_2$ the corresponding projection disregarding $m1_i$ is annotated in the bridge between n_1 and n_2.
   - $\forall\ m2_i \in CS_2$, s.t. $m2_i \notin CS_1$ the corresponding drill-across is annotated to add $m2_i$ to the output schema.

ii) **MD space:** First, we analyze the relationships between the MD spaces in CS_1 and CS_2.
   - If CS_1 and CS_2 are exactly the same, nothing has to be done.
   - Else, for each one-to-many or many-to-one relationship identified between CS_1 and CS_2 we need to modify the output granularity accordingly (see Section 4). If a one-to-many relationship is identified, a proper drill-down operator is annotated in the bridge. Else, in case of a many-to-one relationship, the corresponding roll-up is added.
   - In any other case, given that the structural part of n_1 and n_2 coincide, a 1-1 relationship, as a whole, should be identified between CS_1 and CS_2. Thus, we need to navigate from the MD space in CS_1 to the alternative space in CS_2 (see Section 4) and the corresponding changeBase is added to the bridge.

iii) **Slicers:** Being p_1 and p_2 the conjunction of predicates in the selections of n_1 and n_2, respectively.

– If $p_1 \equiv p_2$ nothing has to be done.
– Else if $p_1 \sqsubset p_2$, the proper union(s) is (are) added to the bridge.
– Else if $p_1 \sqsupset p_2$, a selection(s) is (are) added.
– Else a union(s) (to undo p_1) and a selection(s) (to carry out p_2) are added.

Again, note that this algorithm is sound thanks to the MDA properties, which allow us to undo and redo complementary operators (i.e., projection Vs. drill-across, union Vs. selection, roll-up Vs. drill-down and changeBase Vs. changeBase) to produce CS_2 from CS_1. The produced bridge is then evaluated to decide whether n_1 and n_2 are similar enough, if so we consider both NMACs to belong to the same session. It is out of our current objectives to provide an empirical function to identify when two NMACs are similar enough as to be coalesced in the same session, as it is an application-dependent task. As result, both NMACs are stored in an ordered structure (i.e., a list of NMACs) representing the session and we annotate their relationship with the bridging operators NP_b (to keep track of their logical connection). Finally, we use the last NMAC to keep looking for other queries in the session, whereas the annotated bridging is kept in order to exploit it in future tasks such as query recommendation.

Example 8. Consider Q2 introduced in Example 1. Its NMAC is:

$$\pi_{cs\_quantity}(\gamma^{date \rightarrow month}_{sum(cs\_quantity)}(\gamma^{store \rightarrow state}_{sum(cs\_quantity)}(\sigma_{cs\_product='1'}(raw\_data))))$$

This query can be bridged with Q1 (whose MAC is given in Example 4, which happens to be already normalized), and, as explained in the motivation example, their cube schemas are exactly the same except for their MD spaces, among which we can identify a many-to-one relationship (from year to month). Thus, a drill-down is annotated as the bridge from Q1 to Q2. In this case, it is clear they are close enough and thus, both NMACs are stored in the same session s. Semantically, the annotated bridge means that Q2's output can be obtained by bridging Q1's NMAC with the annotated drill-down (this is represented in the MAC below, where the drill-down is represented by the left-most operator):

$$\gamma^{year \rightarrow month}_{sum(cs\_quantity)}(\pi_{cs\_quantity}(\gamma^{date \rightarrow year}_{sum(cs\_quantity)}(\gamma^{store \rightarrow state}_{sum(cs\_quantity)}(\sigma_{cs\_product='1'}(raw\_data)))))$$

7 Conclusions

We have presented a novel approach to capture analytical SQL queries in a structured way (i.e., a MAC) based on MD algebra. First, we have shown how to normalize MACs in order to compute the similarity between queries in the log and bridge them to obtain the whole session.

This paper mainly sets a new research line for our next future. Specifically, we aim at exploiting the foundations introduced by producing novel solutions for query recommendation, physical tuning and MD design. Furthermore, this framework can also be useful for testing if the way a relational database is used is compliant with the data warehouse and OLAP practices and thus, to which extent it is worth investing in such technology. We also plan to carry out empirical studies to determine how close two NMACs must be in order to be considered part of the same session. Finally, the implementation of this framework is currently under work.

Acknowledgements. This work has been partly supported by the Ministerio de Ciencia e Innovación under project TIN2008-03863. We also thank Jovan Varga for his insights.

References

1. Abelló, A., Romero, O.: On-Line Analytical Processing. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, pp. 1949–1954. Springer, Heidelberg (2009)
2. Abelló, A., Samos, J., Saltor, F.: YAM^2 (Yet Another Multidimensional Model): An extension of UML. Information Systems 31(6), 541–567 (2006)
3. Akbarnejad, J., Chatzopoulou, G., Eirinaki, M., Koshy, S., Mittal, S., On, D., Polyzotis, N., Varman, J.S.V.: Sql querie recommendations. PVLDB 3(2), 1597–1600 (2010)
4. Chatzopoulou, G., Eirinaki, M., Polyzotis, N.: Query recommendations for interactive database exploration. In: Winslett, M. (ed.) SSDBM 2009. LNCS, vol. 5566, pp. 3–18. Springer, Heidelberg (2009)
5. Garcia-Molina, H., Ullman, J.D., Widom, J.: Database Systems. Prentice-Hall, Englewood Cliffs (2008)
6. Giacometti, A., Marcel, P., Negre, E.: Recommending multidimensional queries. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2009. LNCS, vol. 5691, pp. 453–466. Springer, Heidelberg (2009)
7. Giacometti, A., Marcel, P., Negre, E., Soulet, A.: Query recommendations for olap discovery driven analysis. In: DOLAP, pp. 81–88. ACM, New York (2009)
8. Golfarelli, M., Rizzi, S.: Data Warehouse Design: Modern Principles and Methodologies. McGraw-Hill, New York (2009)
9. Hacid, M.S., Sattler, U.: An Object-Centered Multi-dimensional Data Model with Hierarchically Structured Dimensions. In: Proc. of IEEE Knowledge and Data Engineering Exchange Workshop (KDEX 1997).IEEE, Los Alamitos (1997)
10. Khoussainova, N., Balazinska, M., Gatterbauer, W., Kwon, Y., Suciu, D.: A case for a collaborative query management system. In: CIDR (2009), `http://www.crdrdb.org`
11. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: Snipsuggest: Context-aware autocompletion for sql. PVLDB 4(1), 22–33 (2010)
12. Kimball, R., Reeves, L., Thornthwaite, W., Ross, M.: The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing. In: Developing and Deploying Data Warehouses, John Wiley & Sons, Inc., Chichester (1998)
13. Lenz, H.J., Shoshani, A.: Summarizability in OLAP and Statistical Data Bases. In: Ninth Int. Conf. on Scientific and Statistical Database Management (SSDBM), pp. 132–143. IEEE Computer Society Press, Los Alamitos (1997)
14. Platform, S.B.A.: Siebel business analytics server administration guide: Administering the query log, `http://download.oracle.com/docs/cd/E12103_/books/admintool/admintool_AdministerQuery14.html`

15. Romero, O., Abelló, A.: On the Need of a Reference Algebra for OLAP. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2007. LNCS, vol. 4654, pp. 99–110. Springer, Heidelberg (2007)
16. Romero, O., Abelló, A.: Automatic validation of requirements to support multidimensional design. Data Knowl. Eng. 69(9), 917–942 (2010)
17. TechNet, M.: Sql server techcenter: Configuring the analysis services query log, http://www.microsoft.com/technet/prodtechnol/sql/2005/technologies/config_ssas_querylog.mspx
18. TPC: TPC-DS specification (2010), http://www.tpc.org/tpcds/
19. Yan, W.P., Larson, P.-Å.: Performing Group-By before Join. In: ICDE, pp. 89–100. IEEE Computer Society, Los Alamitos (1994)