International Journal of Data Warehousing and Mining
Volume 18 • Issue 1

Initial Optimization Techniques for the Cube Algebra Query Language: The Relational Model as a Target

Thomas Mercieca, University of Malta, Malta*
Joseph G. Vella, University of Malta, Malta
Kevin Vella, University of Malta, Malta

ABSTRACT

A common model used in addressing today's overwhelming amount of data is the OLAP Cube. The OLAP community has proposed several cube algebras, although a standard has still not been nominated. This study focuses on a recent addition to the cube algebras: the user-centric cube algebra query language (CAQL). The study aims to explore the optimization potential of this algebra by applying logical rewriting inspired by classic relational algebra and parallelism. The lack of standard algebra is often cited as a problem in such discussions. Thus, the significance of this work is that of strengthening the position of this algebra within the OLAP algebras by addressing implementation details. The modern open-source PostgreSQL relational engine is used to encode the CAQL abstraction. A query workload based on a well-known dataset is adopted, and CAQL and SQL implementations are compared. Finally, the quality of the query created is evaluated through the observed performance characteristics of the query. Results show strong improvements over the baseline case of the unoptimized query.

KEYWORDS

Abstraction, Algebra, Analytics, Cube, MDX, OLAP, Optimization, Query Language, Relational Model, SQL

INTRODUCTION

The availability of massive amounts of data in every domain and application has led researchers to develop innovative techniques for representing and handling data, e.g., time-series data (Fu, 2011; Esling & Agon, 2012). Within the database field, such techniques are categorized under Online Transaction Processing (OLTP), dealing with voluminous transactions, and Online Analytical Processing (OLAP), addressing sophisticated data analysis over large and varied data sources.

In more detail, the OLAP field is concerned with keeping insightful analysis intuitive and related computation efficient. The OLAP Council (1995), although currently inactive, has provided the following OLAP requirements: (a) modelling across dimensions and through data hierarchies; (b) trend analysis over sequential time periods; (c) slicing subsets for data visualizations, and (d) drilling down to deeper levels for consolidation. Such requirements emphasize the importance of performance and the need for an adequate data model.

DOI: 10.4018/IJDWM.299016

*Corresponding Author

The prevailing data model for OLAP is the data cube. Several OLAP database algebras have been proposed in the literature, suggesting the need for thinking in this data model. Romero and Abelló (2007) survey several OLAP algebras, observing that such algebras express the same fundamental OLAP operations differently, i.e., through different formalisms and semantics to interact with the OLAP cube abstraction. Although interest in this area for its potential of logical optimization and reaching a consensus on modelling issues exists, work in this aspect has been lacking. A standard algebra has yet to be nominated despite being indicated as important for facilitating future research.

A recent development in the algebras is the Cube Algebra Query Language (CAQL) by Ciferri et al. (2013). The focus here is on a data model which is more straightforward and more intuitive for the end-user than comparable models proposed in the past. Thus, this study provides an overview of several algebras to clarify this point. This work aims to build on CAQL by exploring and commenting on the optimization potential of querying through this algebra, specifically by applying parallelism and logical optimization. The main contributions of this article are:

1. The identification and application of several optimization methods which are applicable for this type of algebra based on parallel computing and logical rewriting.
2. The adoption of a query workload and its application to the cube algebra domain, together with an evaluation that focuses on the quality of the query generated. The observed performance characteristics of the executed query are the metric used for evaluation.
3. The strengthening of CAQL's position by addressing implementation details when several OLAP algebras exist, but a standard is lacking.
4. The identification of a database engine that adequately implements the data cube abstraction. The DBMS must be extensible, allowing for seamless entrenchment of this algebra.

It is not within the scope of this article to study the extension of the cube algebras, e.g., through the proposal of new operators. Moreover, the algebra's expressiveness and its relationship with the relational algebra's expressiveness are not within this article's scope.

The remainder of this article is structured as follows. The following section discusses related work in cube data models and algebras. Then, the article describes the main optimization techniques used in this study. Following this section, the performance and experimental analysis of the techniques used are presented using a case study. The article then concludes with final remarks on this study and ideas for future research related to this work.

BACKGROUND

In interpreting data, the main tasks undertaken by data analysts are the summarization of data and the extraction of trends and patterns. Such workloads are analytical and require interaction with entire datasets, consolidating data from various sources for trend discovery or decision support. Within this domain, OLAP tools represent the datasets in *n*-dimensional space, with the prevailing data model for such OLAP requirements being the data cube model. Thus, this type of workload often involves long-running and complex queries, with the modelling of such queries and improving their response times being a significant challenge within the area.

A central and extensive survey on cube and data warehousing models carried out by Ciferri et al. (2013) give an overview of twenty related models. They classify existing models into three: conceptual models using extensions of the Entity-Relationship (ER) Model (Chen, 1976), conceptual models which extend the Unified Modeling Language (UML), and models based on a view of data as a cube. Each model is based on different formalisms and semantics.

Romero and Abelló review ten OLAP models in detail, observing that all are trying to express the same fundamental operations. The emphasis is on the need for a reference set of operators to help research in the area by facilitating the development of design methodologies focused on improving querying, indexing techniques and query optimization. There is an overlap of the models reviewed by Ciferri et al. (2013) and Romero and Abelló (2007).

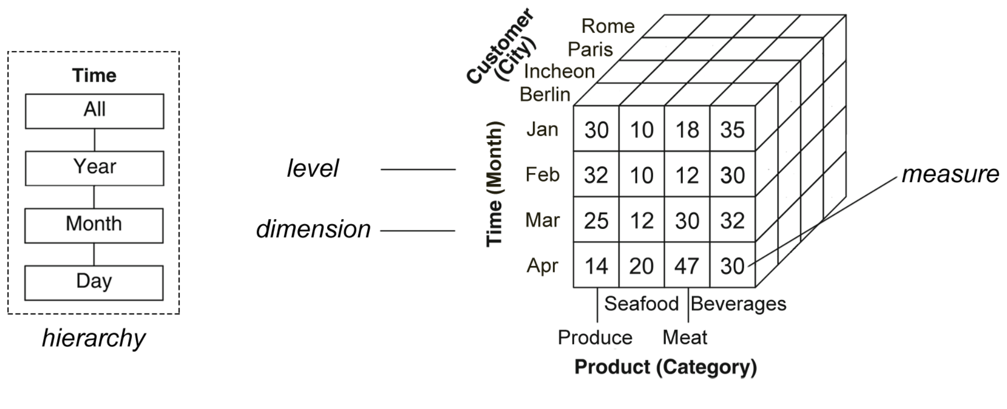
An earlier survey by Vassiliadis and Sellis (1999) also suggests a wide variety of models, that reaching consensus on a standard is essential, and that results on optimization through the use of a generic logical model would prove helpful.Since the survey in Ciferri et al.'s work, several other algebras have been proposed (e.g. Kuijpers & Vaisman, 2016; Banerjee et al., 2021).

In the following subsection, the standard cube data model is introduced in more detail. Then, the ensuing subsections focus on the main algebras in the literature, which are defined over the cube view and provide an algebra without extending existing models.

The Cube Data Model

The data cube is the *de facto* standard data model used in OLAP systems to provide intuitive and flexible multidimensional analysis. Data is structured as a *hypercube*, a multidimensional construct defined in *n*-dimensional space through the dimension modeler's measure and specifications. Conceptually, the data cube allows users to view data from different perspectives and at varying levels of detail. As an abstraction, it is independent of the physical layer. Its implementation can take many forms, such as arrays or relations, with several SQL additions available (Gray et al., 1996). In addition, MultiDimensional eXpressions (MDX) (Microsoft Corporation, 2021) is a popular query language for direct querying an OLAP data cube and makes use of keywords similar to those in SQL but introducing constructs that are natural to the cube, such as selecting measures and dimension hierarchies. However, MDX has received criticism for not having a solid theoretical background, overloading presentational issues with computational issues (Vassiliadis, 1999), not having clearly defined semantics, and being cryptic and unintuitive (Ciferri et al., 2013).

Figure 1. A data cube with dimensions Product, Time and Customer, and measure Quantity. Time dimension hierarchy (left).



A dimension's level indicates the granularity for which a measured value is being shown. In Figure 1, the sales data is being viewed at the levels of Month, City and Category, of the Time, Customer and Product dimensions, respectively. Each dimension is made up of a set of aggregation levels that are hierarchical in nature. The ALL level is the most general level of detail for a dimension, flattening the dimension into a single instance. Each value shown in a cell of the data cube represents a measure relative to the cube's dimensions, e.g. total sales for Month, City and Category. Measured values can change as the data cube is being manipulated, and a data cube can contain several such measures.

Typical operations on the cube (Vassiliadis & Sellis, 1999) include rolling-up, which is the aggregation along the specified dimension hierarchy, dicing and slicing, which is the selection of specific parts of a cube, and pivoting the cube - a reorientation of the cube. Operations on the cube can also include transforming the cube's contents and linking multiple cubes together, drilling across for more detail.

The discussion on the appropriate physical model for OLAP has focused on two views - either the Multidimensional OLAP (MOLAP) approach or the Relational OLAP (ROLAP) approach. MOLAP systems are associated with using multidimensional arrays in a Multidimensional DBMS (MDBMS) as the underlying data model, as well as populating each possible entry in the defined multidimensional space. This is vulnerable to the curse of dimensionality problem (Donoho, 2000); as the dimensions increase, there is an explosion in the size of this dimensional space, with the actual grouped data being very sparse. Sparsity in data is present through there being only an entry for each recorded observation.

Furthermore, MOLAP systems focus on improving performance by reducing data size using the precomputation of possible aggregations. However, this approach introduces the overhead of propagating any changes in data to these pre-computed datasets.

Meanwhile, in ROLAP-based systems, a Relational DBMS (RDBMS) is at the core of OLAP operations: a table abstraction is used to perform OLAP operations. In addition, the limitations of the ROLAP approach are characterized by the limitations of the underlying RDBMS. As ROLAP systems are closer to the operational data source than MOLAP systems, ROLAP systems can allow any operational data source already in an RDBMS to be queried, whereas, in MOLAP, data has to be exported and then loaded in the MOLAP's environment and structures. As a result, ROLAP has an advantage in flexibility for applications without predefined query requirements. Modern RDBMSs with the support of materialized views and triggers can implement similar precomputation techniques to MOLAP; thus, there can be an overlap between approaches, i.e. Hybrid OLAP (HOLAP).

Agrawal et al.'s Data Model

Agrawal et al. (1997) propose a data model focused on the symmetric treatment of dimensions and measures. Symmetric in this context refers to the ability to apply operations typically associated with dimensions on measures. Thus, similarly to how one can aggregate to a monthly level in a Time dimension, the ranges of a Quantity measure can also become candidates for aggregation, e.g., aggregating over the groups of Quantity 0-999, 1000-1999 and so on.

The proposed algebraic operators are closed - they are defined on cubes and produce a cube as output. The operators can be composed, creating a query expression model and introducing optimization options such as disregarding intermediate cubes which are not of interest to the user. Agrawal et al. also remark that the operators are minimal such that one cannot be expressed in terms of the other, and dropping one will lead to reduced functionality. Through this algebra, Agrawal et al. emphasize an approach to stay as close to relational algebra as possible, to benefit translation and compatibility to SQL.

The logical cube data model is based on the following elements. The model has k dimensions, where each dimension has a name D_i and a domain dom_i from which values are taken. Elements are defined as a mapping $E(C)$ from $dom_1 \times \cdots \times dom_k$ to 0, 1, or an n-tuple. An element of 0 or 1 indicates whether the value exists or not, and the n-tuple indicates that additional information is available for that combination of dimension values. They also define a set of basic operators: push, pull, restriction, destroy dimension and join, described in Table 1. Using these operators, Agrawal et al. build higher-level operations such as roll-up, drill-down, and star join.

Table 1. Agrawal et al.'s (1997) proposed core operators

Operator	Description
Push	Convert dimensions into elements that can be manipulated using a function.
Pull	The converse of the PUSH operator. Creates a new dimension for a specific element, converting an element into a dimension.
Destroy Dimension	Reduces the dimensionality of the cube through removing a dimension which only has a single value in its domain. A dimension that has multiple values cannot be directly destroyed because the elements might not be functionally determined by dimension values in that scenario.
Restriction	Operates on a dimension of a cube, removing the cube values from the dimension that do not satisfy a provided condition.
Join	Relates the information across two cubes. Both cubes need not have the same number of dimensions and a mapping function is used for mapping the dimension being joined to the resulting dimension.

The authors describe their algebra as at least as powerful as relational algebra. However, they point out that a precise indication of the expressive power of the proposed model is an open problem. In addition, a related open question is that of defining the formal notation of completeness for multidimensional queries and evaluating how complete their algebra is through this notation.

The authors also discuss the implementation aspect in their proposal. They remark that translation of each proposed operator to SQL is possible and that it is unclear what the expected performance can be. The authors comment that such algebraic operators can also be implemented on a specialized engine and a relational one.

Vassiliadis Base Cube Data Model

In the multidimensional data model proposed by Vassiliadis (1998), the author defines the concepts of dimension and cube while addressing challenges with hierarchies e.g. the de-aggregation of data. A particular contribution of this work is the use of the base cube, core to some of this algebra's operators. The basic operations in this cube model (shown in Table 2) are LEVEL CLIMBING, PACKING, FUNCTION APPLICATION, PROJECTION, and DICING. NAVIGATION and SLICING are then defined on top of these basic operators.

Table 2. Vassiliadis's (1998) cube operators

Operator	Description
Level Climbing	Replaces all values of a set of dimensions with values of dimension levels of a higher level.
Packing	Merges multiple data instances having the same dimension values into one.
Projection	Removing specific dimensions from both the cube and its base cube entry.
Function Application	Applies a specific function to the measure of a cube.
Dicing	Selects data to satisfy a given expression. It is applied to both cube and base cube.

Vassiliadis defines a cube C_b as a 3-tuple $<D_b, L_b, R_b>$, where D_b is a list of dimensions representing the measures, L_b is a list of dimension levels, and R_b is a set of cell data (the data of the cube containing elements of the dimension and measure values) at the lowest dimension levels. The definition of a cube is then altered to define a cube as the base cube of itself with C_b as a 4-tuple $<D_b,$

L_b, C_b, R_b>. This definition of the base cube enables a direct and correct evaluation of operations. As a cube is aggregated, a challenge is presented with regard to specific operations. One example of this is getting the average sales per year from a cube whose measurements have been aggregated using a count function, i.e., going from count data to average data. Related to this problem is also de-aggregation operations. Once the data is aggregated through going up the dimension hierarchy, one might be interested in going back down. For example, after aggregating up from a daily to a yearly level and then manipulating the cube, one might be interested in interacting with the data at the daily level. Vassiliadis remarks that not only does the base cube definition provide a mechanism to preserve data in such a series of operations, and avoids the possibility of using expensive JOIN operations to try answering such queries, i.e., a performance implication.

The mapping of this algebra to a relational model is done based on two mapping functions. One maps a dimension level to an attribute of a relation, and the other maps an attribute to a dimension level. A dimension level can be mapped to more than one attribute, e.g. a dimension level may be expressed in two columns, possibly across two relations. In addition, mapping to a multidimensional array is said to be trivial, following the work of Cabbibo and Torlone (1997).

Vassiliadis indicates that the optimization challenges of this algebra are related to the execution of operations and suggests the application of view usability (Levy et al., 1995) techniques. Workloads change the volume of the cube; therefore, it is of interest to weigh computing directly from this base cube against computing from an intermediate result.

Ciferri et al.'s Cube Algebra

Ciferri et al. (2013) present the CUBE ALGEBRA QUERY LANGUAGE (CAQL). Its main characteristics are that it focuses on the view of a cube and on providing an intuitive set of operators to general users (such as managers) by which to manipulate a cube.

The model in this algebra is based on cube and dimension definitions which the user specifies through the language's proposed CUBE ALGEBRA DEFINITION LANGUAGE (CADL). A cube schema in this model is defined as a 3-tuple <*nameCS, D, M*> with *nameCS* being the name of the cube, *D* is a set of dimension levels, and *M* is a set of measures. Meanwhile, a dimension schema is a 3-tuple <*nameDS*, $L \rightarrow$ > with *nameDS* being the name of the dimension, *L* is a set of pairs of the form <*l, A*> such that *l* is a level and *A* is a set of attributes describing a level. There is a particular level <*All*, θ >, and an attribute in *A* identifies a member in level *l*. $\rightarrow$ is a partial order on the levels, creating a hierarchy and defining a graph described by the attributes in *A*. $\rightarrow$ also has a unique bottom level.

The algebra is based on four core operators: ROLL-UP (with DRILL-DOWN as its inverse), SLICE, DICE, and DRILL-ACROSS, and is extended with the MAP operator. The authors remark that this set of operators can be extended further in practice. Additional details on these operators are provided in the following list:

- **DICE:** Returns a cube containing only the cells satisfying an expression over dimension levels and measures.
- **SLICE:** Removes a selected dimension from a cube, returning a cube with *n-1* dimensions. The dimension to be removed must be a singleton; if the dimension has more than one value, one must either apply a ROLL-UP to the ALL level or apply a DICE operator prior to slicing.
- **ROLL-UP:** Aggregates measures according to the dimension hierarchy. An aggregate function must be specified for each measure. This operator reduces the numerosity of the data rather than the number of dimensions.
- **DRILL-DOWN:** Undoes the aggregation of a cube in an aggregated state. The authors suggest implementing it through tracking the paths followed during a ROLL-UP operation.
- **DRILL-ACROSS:** Given two cubes, compose a single consolidated cube with a new cube structure. This operation is subject to cube compatibility as two cubes can be related, but their structure

can differ. In those cases, the authors recommend controlling the granularity of the cubes before using this operator or applying a mapping function.

- **Map:** Applies a mapping function to a measure such that the resulting cube contains the updated measure.

Discussion and Other Related Work

Up until this point, the well-established OLAP algebras in the literature have been presented. The essential themes in this review are that (a) implementation and optimization details have remained relatively unexplored, despite the indication that they are indeed possible and necessary, (b) that a lack of standard data model is present, and (c) that OLAP algebras seem to have, over time, prioritized having user-intuitive operators, with the latter being the primary justification of the authors of this work for choosing CAQL. A number of more novel algebras have been proposed in the area and will also be covered next in this section.

Kuijpers and Vaisman (2016) propose a formal OLAP algebra, emphasizing that this formalization is required within the Big Data landscape to perform real-time OLAP operations. In their contribution, the common operators manipulating a data cube have their semantics clearly defined, and a formal proof showing that operators can be composed is provided. The authors claim that their work is the first to provide formal proof in the area due to work, at the time, lacking formalism or applicability.

Banerjee et al. (2021) propose a formal OLAP algebra for NoSQL systems. They highlight that the lack of a common OLAP specification is also a problem within the NoSQL world, and therefore, data warehousing portability is limited. In their work, a uniform syntax for different OLAP operations is proposed. A key insight is that the operators proposed are independent of the physical level and can therefore work on NoSQL systems as a result.

ADDRESSING OPTIMIZATION IN THE CUBE ALGEBRA QUERY LANGUAGE

As OLAP queries are expected to be very intensive and complex, often having to interact with extensive and varied datasets, careful consideration of their response time is crucial. Other OLAP algebras have briefly referred to the optimization potential of their algebra. Namely, Agrawal et al. (1997) identified that the use of a query expression model which can disregard intermediate result sets introduces optimization options. Vassiliadis (1998) has also emphasized that the optimization challenges of this algebra are related to the execution of operations and has suggested the use of views within the implementation.

As a large number of cube representations exist, the application of a technique is not guaranteed to be directly transferable to another model. Romero and Abelló (2007), as well as Vassiliadis and Sellis (1998) have already indicated that there is a wide variety of models and that discussion on optimization through the use of a generic logical model would prove useful. Hence, the authors are restricting this work to CAQL, which has already been established for a number of years and has been designed to be user-centric.

Cube algebras are sometimes compared to the relational model as a possible target for translation into such a model. Through their multidimensional algebra, Agrawal et al. (1997) emphasize the similarity to relational algebra, where this connection can assist in translation and compatibility to SQL. Ciferri et al. (2013), when discussing CAQL consider the drill-across operator as similar to the relational natural join, the dice operator as analogous to the relational selection and the slice as comparable to the relational projection. In this section, optimization for CAQL is discussed using this relationship to relational algebra.

Logical Rewriting

CAQL, as an algebra, can have its expressions rewritten based on logical equivalences. Due to its similarity to relational algebra, the authors of this work aim to show that it is possible to leverage well-known relational logical equivalences (Ullman et al., 2008) for CAQL expression rewriting. In this section, two such examples are discussed to highlight this point. However close this similarity might be, some CAQL operators have certain nuances distinguishing them from the relational model (e.g., a SLICE operator can optionally be combined with a ROLL-UP to the level ALL). Finally, the authors emphasize that although the relational model is being consulted to identify and apply rewriting, the implementation does not necessarily have to be a relational one, i.e., these rules are expected to find application in specialized OLAP engines as well.

One rule is the pushing down of the DICE operator across the DRILL-ACROSS operator. Looking at relational algebra, the related identity is that the relational SELECT (σ) can be distributed over the NATURAL JOIN (∞), so long as the attributes in θ exist in both input relations E_1 and E_2:

$$\sigma_{,}\left(\boldsymbol{E}_1 \infty \boldsymbol{E}_2\right) = \sigma_{,}\left(\boldsymbol{E}_1\right) \infty \sigma_{,}\left(\boldsymbol{E}_2\right) \tag{1}$$

In the case of the DRILL-ACROSS, its specification states that the result of the DRILL-ACROSS operator will have the same dimensions as both input cubes. Then, this means that a reference in the DICE expression to a dimension can be pushed down to both sides. Meanwhile, pushing down a measure in this scenario would require first identifying which input cube it came from.

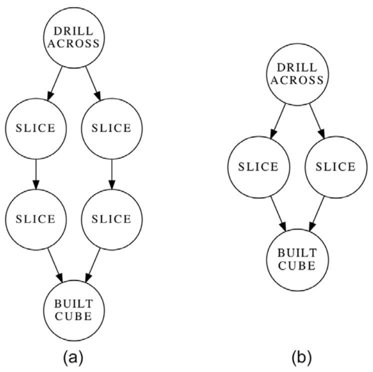
Pushing down the SLICE under a DRILL-ACROSS will relax the JOIN condition, subsequently producing a different output. Consider a JOIN between two cubes and pushing down a SLICE to remove a dimension. Then the rewriting effect of such a rule will result in instead joining on one dimension less: a different JOIN condition that can be expected to alter the answer of the query. In addition, unlike the relational projection, one would need to consider that the SLICE operator can include a ROLL-UP to the level ALL.

The second rule being used in this work is an application of the relational identity, which says that sequences of the relational PROJECT (Π) can be omitted, up until the last projection:

$$\Pi_{L1}\left(\Pi_{L^2}(\ldots(\Pi_{L_n}(E))\right) = \Pi_{L_1}(E) \tag{2}$$

The inverse nature of the relational projection (state which attributes are desired) to the SLICE (state which dimension attributes are not desired) makes the compatibility across data models precise. Thus, in the CAQL model, this can be implemented through merging all contiguous SLICE operations (as in Figure 2) so long as the result of the merged output is compatible with all parent instances.

Figure 2. Merging of multiple SLICE operators in a query graph, input (a) resulting in (b)



The idea behind the selected rules is to push down the operators closer to the data sources, as a result reducing the amount of intermediate space produced for all subsequent operations such that they can be performed on smaller-sized data (e.g., perform DRILL-ACROSS over less voluminous cubes). For simplicity, in this work, it is assumed that such rewriting will always yield a performance benefit, although this might not always be the case with other rules such as interactions between SLICE and DICE (i.e., is it worth reducing the volume of the cube through reducing its dimensionality or through filtering the data first?). Thus, one would have to consider more contrived solutions to guide the rewriting process the more identities that are identified, e.g., applying a cost function from collected statistics about the cubes.

Parallel Execution

A query graph, other than illustrating how the query is to be resolved, also indicates which parts are safe to be executed concurrently. In a relational environment, the underlying CAQL operator is compatible with the relational engine, i.e. is a translation to SQL, with each operator providing input for another operator. Thus, one can achieve parallelism at SQL level here by controlling the numbers of workers allocated to each operator.

The main challenge in this approach is the compilation of an SQL query in isolation from each other; if two or more translated operators were compiled together, there would have been additional information supplied to the SQL optimizer, enabling it to make a more informed decision. Additionally, it is a form of bulk execution: an entire table is being supplied to the other operator at once, as opposed to streaming (i.e. Volcano's tuple-at-a-time execution (Graefe, 1994)).

Thus, an alternative approach that addresses these concerns is to rewrite the translated operators under a single SQL query. In contrast, the limitation of this approach is that the final query will be substantial in size, which may result in too much time planning or missing potential optimization opportunities if planning is stopped prematurely due to a time-out attached to the SQL engine's optimization effort. Furthermore, as a single SQL query, then execution can be streamed (as opposed to executed in bulk as several SQL queries), should the underlying engine support such kind of processing. Keeping in mind that parallel workers are allocated on a per-query basis, then in this technique, the control of the allocation of resources is delegated to the underlying engine.

This technique of rewriting the operators under a single SQL query can be implemented through Common Table Expressions (CTEs). When dealing with CTEs, one must account for the MATERIALIZED parameter: taking the PostgreSQL DBMS as an example, up until PostgreSQL 12 (2020), CTE execution has always been materialized, and this has been referred to as an optimization fence with such a query not being executed in parallel. This has been addressed in PostgreSQL 12 by introducing the NOT MATERIALIZED option for CTEs, which inlines the inner CTE queries into the outer query allowing the SQL query optimizer to then to be able to be optimized further in this manner and also supports parallel execution.

So far, the discussion on parallel execution has been focused on executing a single SQL statement in parallel (i.e. intra-node parallelism). In effect, the simplest strategy is that of allocating all available resources to the current operator being executed. However, at the abstract level of CAQL, it is possible to execute multiple operators in parallel, i.e. allocating resources over several SQL statements, resulting in inter-node parallelism. Using this technique, the sequential component of query planning is alleviated through planning several SQL queries simultaneously. The challenge behind this technique is finding a policy by which to distribute resources across operators such that minimal processor idle time is observed.

EXPERIMENTAL EVALUATION

This section evaluates CAQL from a performance standpoint. The testing setup used is covered first. What follows is an explanation of the baseline comparisons and a presentation of the query workload. Finally, the section concludes with a discussion on the results observed.

Test Setup

All testing is performed on an i5-2500k 4-cores 3.3GHz CPU, with 16GB DDR3 RAM and a 512GB SSD computer setup running the Manjaro 18.1 (4.19 Linux Kernel) OS. The PostgreSQL 12 DBMS is used as the underlying engine for the experiment, mainly due to its open-source nature, wide adoption, and focus on extensibility (Stonebraker et al., 1987). It has been compiled with the -O2 flag and has the following configurations:

- Shared_buffers - 6GB
- Effective_cache_size - 12GB
- Work_mem - 2GB
- Random_page_cost - 1.1

A tool was written to translate CAQL and CADL statements into SQL, i.e., a translation to the relational model. In addition, this tool is used to facilitate the application of the optimizations discussed in this work. Throughout this experiment, cubes are implemented using tables which are UNLOGGED (PostgreSQL, 2020), to reduce disk interactions, and the CTEs used are NOT MATERIALIZED, which allow the DBMS to optimize the translated query further.

Variance in results can be attributed to the many intricacies of the background DBMS and OS processes. For this reason, each test is performed ten times in total, and the mean is taken. An additional warm-up phase is performed at the start of the experiment to ensure that the system caches are not cold.

Baseline Comparison

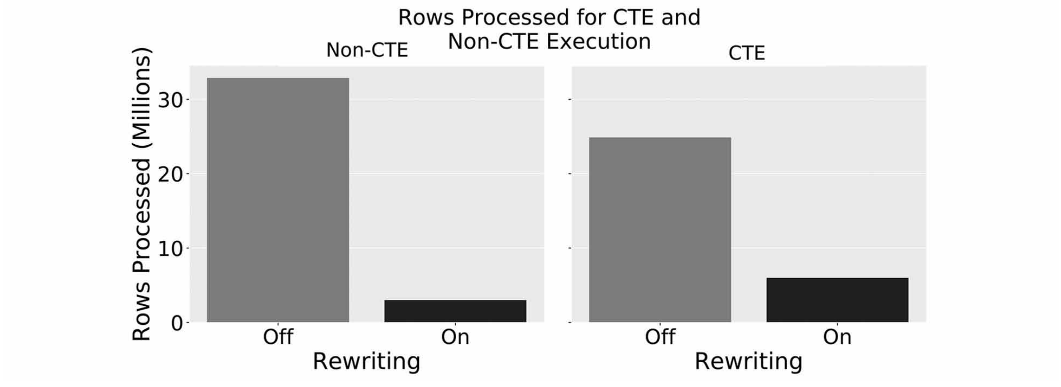
For the first experiment, a query is first executed without the application of logical rewriting, without CTE translation and with just one worker assigned to both cases. This is being referred to as the baseline case. Then, the query is executed with the logical rewriting effect for both CTE and non-CTE translations. This demonstrates the improvements that are achievable at the logical level. The following experiment introduces parallelism through varying the number of parallel resources allocated, demonstrating the expected speedup benefit. The term *bulk* used here refers to how the available parallel workers were allocated: all were allocated to one operator at a time. Finally, the third experiment compares the baseline against an SQL implementation's execution time, thus highlighting the difference between a query expressed in CAQL and in the widely adopted SQL. The query processor configuration settings are set to default for SQL execution, i.e., parallelism was not forced in this case.

Workload

The dataset used in this project is the Star Schema Benchmark (SSB) (O'Neil et al., 2007). It is a widely used benchmark (Sanchez, 2016) for star schema models in an OLAP environment. The SSB was scaled with the configuration of SCALE FACTOR = 1.

A Simple Moving Average (SMA) query, a query common for several users (Nadler & Kros, 2005), was used on this dataset. Its specification, in both CAQL and SQL, is provided in the *Appendix*. The result of both CAQL and SQL queries have been matched against each other to ensure that the queries being executed are at least producing the same output.

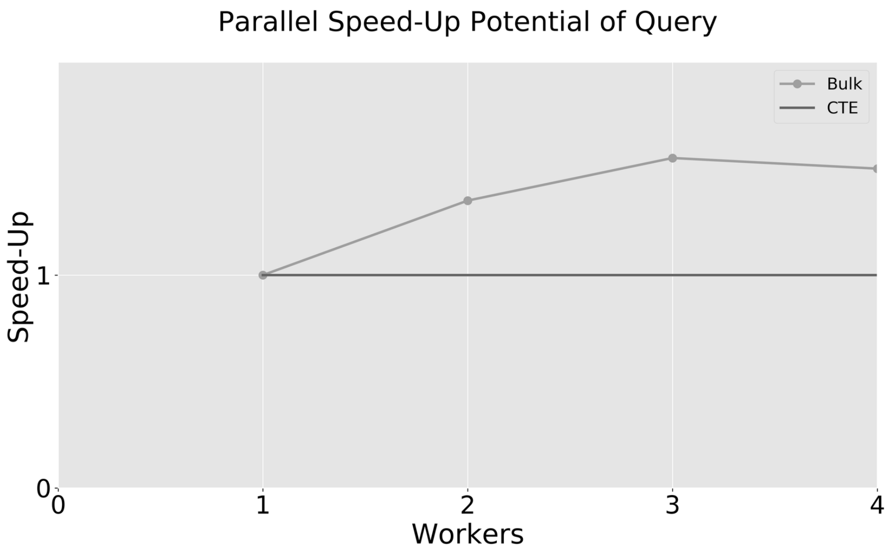
Figure 3. Total rows processed by one worker for a Simple Moving Average query with the rewriting effect on, and off.



Discussion of Results

Figure 3 shows a difference in the effect of rows processed using rule rewriting for both CTE and non-CTE modes, with the CTE mode showing a less dramatic difference. This suggests two things: (a) that rule rewriting can reduce the workload in both CTE and non-CTE scenarios by a substantial amount and (b) that a seemingly better effect, by the metric of rows processed, is produced in the non-CTE translation.

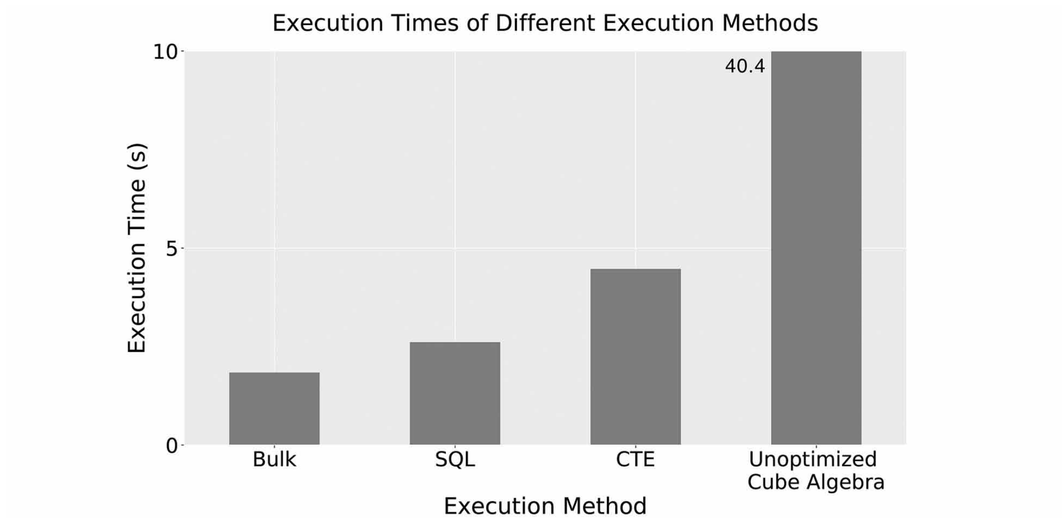
Figure 4. Parallelism speedup potential against the number of workers for each execution mode, comparing a single worker with an increasing number of workers for a Simple Moving Average type query.



Adding workers to the query with query rewriting (as shown in Figure 4) results in the *bulk* execution method being parallelized and the CTE execution to not benefit from parallelism. The lack of a linear speedup is expected due to the disk interactions. The lack of a speedup as far as CTE is concerned suggests that the functions (i.e. window functions) used to express the query hijack the parallelism of the entire query.

A clear improvement can be seen when comparing the unoptimized CAQL with the alternative optimization techniques in Figure 5. The CTE is the least appealing of the alternatives; this can be explained through the lack of parallelism discussed previously. It is interesting that the gap between the *bulk* and SQL execution methods is minimal, suggesting that the optimizations brought CAQL execution to a reasonable level of performance.

Figure 5. Comparing CAQL execution modes against the SQL implementation for a Simple Moving Average type query implementation.



CONCLUSION

Several algebras have been proposed for working with multidimensional data in an OLAP environment, but a standard algebra has not yet been established, even if this is considered to be beneficial. Of these multidimensional algebras, a number focus on operating directly over the view of data as a cube, but discussions on optimization remain relatively unexplored.

In this work, the authors focus on a well-established and intuitive addition to these set of algebras and apply several optimization ideas. These ideas focus on exploring parallelism and logical rule rewriting at a conceptual level, showing that ideas from the relational model are also applicable in this domain. The open-source PostgreSQL was used to work with this algebra at a physical level. Although this is a post-relational engine, these ideas are expected to find application in specialized OLAP engines as well. A Simple Moving Average style query was presented to demonstrate the effect of these ideas, with up to a $16\times$ improvement over the baseline case having been observed. This result is also close to that of an SQL implementation of the query.

Future research is possible in a number of areas. First, an extensive set of logical equivalences, in addition to those proposed in this work, can be identified and expressed formally. Second, the parallelism in this work involved using all available resources on a single operator at a time; future work can explore different ways of allocating resources. As the number of identified optimization techniques increases, then it is of interest to explore the problem of choosing an optimized plan from many alternatives in a reasonable amount of time, e.g., through a cost function tailored for the OLAP algebra environment. Finally, as this OLAP algebra becomes more common, it would be interesting to see the addition of CAQL queries and case studies, as well as a standard benchmark test tailored to this algebra.

ACKNOWLEDGMENT

This research was supported by the Malta Council of Science & Technology (MCST) [R&I-2016-013T].

FUNDING AGENCY

The publisher has waived the Open Access Processing fee for this article.

REFERENCES

Agrawal, R., Gupta, A., & Sarawagi, S. (1997). Modeling multidimensional databases. *Proceedings 13th International Conference on Data Engineering*. doi:10.1109/ICDE.1997.581777

Banerjee, S., Bhaskar, S., Sarkar, A., & Debnath, N. (2021). A formal OLAP algebra for NoSQL based Data Warehouses. *Annals Of Emerging Technologies In Computing*, *5*(5), 154–161. doi:10.33166/AETiC.2021.05.019

Cabibbo, L., & Torlone, R. (1998). A logical approach to multidimensional databases. *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, 183–197.

Chen, P. P.-S. (1976). The entity-relationship model—Toward a unified view of data. *ACM Transactions on Database Systems*, *1*(1), 9–36. doi:10.1145/320434.320440

Ciferri, C., Ciferri, R., Gómez, L., Schneider, M., Vaisman, A., & Zimányi, E. (2013). Cube Algebra: A generic user-centric model and query language for OLAP cubes. *International Journal of Data Warehousing and Mining*, *9*(2), 39–65. doi:10.4018/jdwm.2013040103

Datta, A., & Thomas, H. (1999). The cube data model: A conceptual model and algebra for on-line analytical processing in data warehouses. *Decision Support Systems*, *27*(3), 289–301. doi:10.1016/S0167-9236(99)00052-4

Donoho, D. L. (2000). High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture,* 1-32.

Esling, P., & Agon, C. (2012). Time-series data mining. *ACM Computing Surveys*, *45*(1), 1–34. doi:10.1145/2379776.2379788

Fu, T. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence*, *24*(1), 164–181. doi:10.1016/j.engappai.2010.09.007

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). *Database systems: The complete book*. Prentice Hall Press.

Graefe, G. (1994). Volcano-an extensible and parallel query evaluation system. *IEEE Transactions on Knowledge and Data Engineering*, *6*(1), 120–135. doi:10.1109/69.273032

Gray, J., Bosworth, A., Lyaman, A., & Pirahesh, H. (1996). Data cube: a relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS. *Proceedings of the Twelfth International Conference on Data Engineering*. doi:10.1109/ICDE.1996.492099

Kuijpers, B., & Vaisman, A. (2017). An algebra for OLAP. *Intelligent Data Analysis*, *21*(5), 1267–1300. doi:10.3233/IDA-163161

Levy, A. Y., Mendelzon, A. O., & Yehoshua, S. (1995). Answering queries using views. *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. doi:10.1145/212433.220198

Microsoft Corporation. (2021). *Multidimensional expressions (MDX) reference*. http://msdn.microsoft.com/en-us/library/ms145506.aspx

Nadler, S., & Kros, J. F. (2007). Forecasting with excel: Suggestions for managers. *Spreadsheets in Education, 2*(2).

O'Neil, P., O'Neil, B., & Chen, X. (2007). *The Star Schema Benchmark (SSB)*. https://www.cs.umb.edu/~poneil/StarSchemaB.pdf

OLAP Council. (1995). OLAP and OLAP server definitions. *Research Technology*. http://www.olapcouncil.org/research/glossaryly.htm

Postgre, S. Q. L. (2020). CREATE TABLE. *PostgreSQL Documentation*. https://www.postgresql.org/docs/12/sql-createtable.html

Postgre, S. Q. L. (2020). WITH queries (Common Table Expressions). *PostgreSQL Documentation*. https://www.postgresql.org/docs/12/queries-with.html

Rizzi, S. (2007). Conceptual modeling solutions for the Data Warehouse. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, 1–26.

Romero, O., & Abelló, A. (2007). On the need of a reference algebra for OLAP. *International Conference on Data Warehousing and Knowledge Discovery*, 99–110. doi:10.1007/978-3-540-74553-2_10

Salley, C. T., Codd, E. F., & Codd, S. B. (1993). *Providing OLAP to user-analysts: An IT mandate*. Academic Press.

Sanchez, J. (2016). *A review of Star Schema Benchmark*. Computing Research Repository.

Sellis, T. K., Tsois, A., & Karayannidis, N. (2001). MAC: conceptual data modeling for OLAP. *Proceedings of the International Workshop on Design and Management of Data Warehouses*, *39*(5).

Stonebraker, M., Anton, J., & Hirohama, M. (1987). Extendability in Postgres. *A Quarterly Bulletin of the Computer Society of the IEEE Technical Committee on Data Engineering*, *10*, 16–23.

Vassiliadis, P. (1998). Modeling multidimensional databases, cubes and cube operations. *Tenth International Conference on Scientific and Statistical Database Management*. doi:10.1109/SSDM.1998.688111

Vassiliadis, P., & Sellis, T. (1999). A survey of logical models for OLAP databases. *SIGMOD Record*, *28*(4), 64–69. doi:10.1145/344816.344869

Joseph G. Vella lectures and researches in the areas of database technology at the University of Malta. His first degree was a BSc in Mathematics and Computing (UM) and a doctoral degree from the University of Sheffield Engineering Faculty. Dr Vella has participated in numerous projects at national and EU levels mainly dealing with data integration and consolidation for data warehousing, data science, and cloud services.

Kevin Vella is an Associate Professor of Computer Science at the University of Malta's Faculty of ICT. His teaching and research activities focus on the scientific and technical aspects of concurrent and distributed computing, operating systems, and programming languages. He holds a PhD in Computer Science from the University of Kent, UK, and a BSc in Mathematics and Computing from the University of Malta.

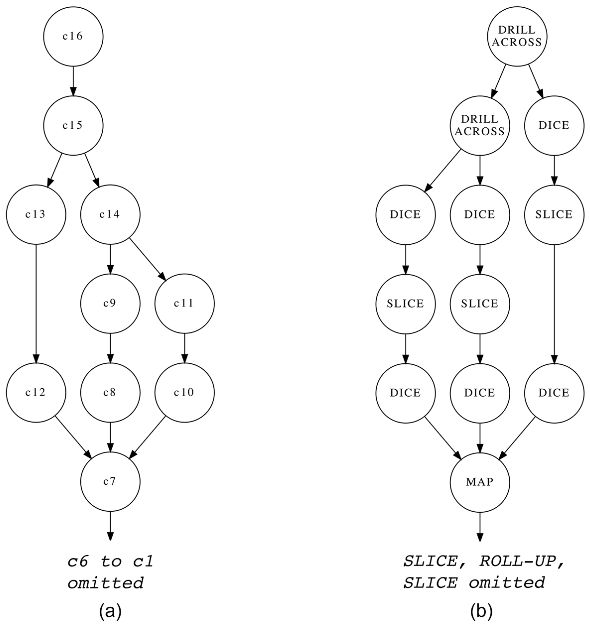
APPENDIX A - SIMPLE MOVING AVERAGE STYLE QUERY

A manager is interested in comparing monthly operations between the first half of the years 1992, 1993 and 1994. He sees that there is variation and is interested in smoothening out this data. His initial attempt consists of applying a 3-month lagging Simple Moving Average.

Using CAQL

```
c1:= SLICE(LineOrder_Cube, Customer, ROLL-UP{SUM});
c2:= SLICE(c1, Part, ROLL-UP{SUM});
c3:= SLICE(c2, Supplier, ROLL-UP{SUM});
c4:= SLICE(c3, Day_View, ROLL-UP{SUM});
c5:= ROLL-UP(c4, Time->Month, {(quantity, SUM)});
c6:= SLICE(c5, Time, ROLL-UP{SUM});
c7:= MAP(c6, {(qty_lag_sma_3,SUM_quantity,month_year_lag_sma(3))});
c8:= DICE(c7,"Year_View.Year.value"=1992);
c9:= SLICE(c8, Year_View);
c10:= DICE(c7,"Year_View.Year.value"=1993);
c11:= SLICE(c10, Year_View);
c12:= DICE(c7,"Year_View.Year.value"=1994);
c13:= SLICE(c12, Year_View);
c14:= DRILL-ACROSS(c9,c11);
c15:= DRILL-ACROSS(c13,c14);
c16:= DICE(c15, text_to_month("Month_View.Month.value") <= 6);
```

Figure 6. Query rewriting, input query (a) producing (b)



Using SQL

```
WITH sma AS
    (
    -- calculate a 3-SMA over ordered months
    SELECT “Time.Year.value”,”Time.Month.value”, sum_quantity,
    ((sum_quantity + COALESCE(LAG(sum_quantity,1)
    OVER (ORDER BY “Time.Year.value”,”Time.Month.value”),0)
    -- + ...
    ) / 3) AS sma
    FROM
    (
    -- rollup to Time.Month with sum of quantity (measure)
    SELECT “Time.Year.value”, “Time.Month.value”, SUM(quantity)
    FROM
    (
    SELECT “Time.Year.value”, “Time.Month.value”, quantity
    FROM lineorder_cube
    ) y_m_q
    GROUP BY “Time.Year.value”, “Time.Month.value”
    ) y_m_sum_q
    )
    SELECT sma.”Time.Month.value”, sma.sma -- project remaining attributes
    -- get the sma of each year from 1992 to 1994 and join them on month
    FROM sma, sma sma1, sma sma2
    WHERE sma.”Time.Year.value” = 1992 AND sma1.”Time.Year.value” = 1993
    AND sma2.”Time.Year.value” = 1994
    AND sma.”Time.Month.value” = sma1.”Time.Month.value”
    AND sma.”Time.Month.value” = sma2.”Time.Month.value”
    -- get first half of year
    AND text_to_month(sma.”Time.Month.value”) <= 6
```