Implementing Security for Multidimensional Expression Queries

AHMAD MOUSA ALTAMIMI, MAHMOOD GHALEB ALBASHAYREH
Computer Science Department
Applied Science Private University
Amman
JORDAN
a_altamimi@asu.edu.jo, m_albashayreh@asu.edu.jo

Abstract: - As the security takes center stage in the recent years, most important projects underway are those associated within security. Organizations collect and process data to operate their day-to-day business successfully. Such data is stored in enterprises data warehouses and presented so that users can make decisions more easily. Online Analytical Processing (OLAP) is the prevalent component of these systems where data is organized in the form of multidimensional representation known as a data cube. A specialized query language called multidimensional expression (MDX) is then used with specific syntax and semantics for manipulating OLAP cubes. However, MDX users can override- either intentionally or unintentionally - the security policy of the system to disclose sensitive information and breach an individual's privacy. In this paper, we present a framework that removes security threats by re-writing MDX queries to ensure consistent data accessing. Experimental results demonstrate that security can be ensured without affecting the usefulness of OLAP systems.

Key-Words: - OLAP, data cube, MDX, data privacy, security polices, query rewriting

1 Introduction

Current database systems, with large numbers of users, require security control mechanisms that restrict access to the stored sensitive data/objects. While numerous mechanisms have been provided for relational model [1,2,3], little effort has been made to address the unique security requirements of the multi-dimensional context. Due to the difference between the natures of the two models objects, the former mechanisms cannot be applied to the latter. In particular, objects in a relational context include logical elements such as tables, records within those tables, and fields within each record. In contrast, objects in a multi-dimensional model are elements of the more abstract conceptual model and include the dimensions of the cube, the hierarchies within each dimension, and the aggregated cells (or facts). This changes the logic or focus of the security mechanism. For instance, a user in a relational environment may be allowed direct access to a specific record or field in that record; while a user in cube model may be given permission to dynamically aggregate measure values at a specific level of detail in one dimension hierarchies. Anything below this level would also be considered sensitive, and hence should be protected.

In fact, the hierarchical nature of the cube allows users to bypass partial constraints defined at alternate aggregation levels - intentionally or unintentionally - and as consequence reveals the sensitive data.

In this work, we target the multi-dimensional model that uses Multidimensional Expressions (MDX) language. MDX is a query language for OLAP databases, much like SQL is a query language for relational databases. MDX has become widely used in the field of analytical applications such as Microsoft Analysis Services [4], IBM Cognos [5], Mondrian OLAP server [6], and Essbase [7]. Most of these systems provide simple, very basic security control. As result, there is a clear need to provide advanced security countermeasure for these given DBMS platforms. As a result, many security frameworks have been proposed, most of them deal with SQL language such as [8,9]. Theses frameworks aim to eliminate unauthorized accesses for SQL queries.

In this paper, we provide a security framework that achieves the same security functionality but for MDX language and it consists of three main parts. The Interpreter and Decomposer module parses and translates the MDX query into an intermediate representation algebra (IR). The IR identifies the core operations associated with the data cube model like the Selection and Projection operations. We note at the outset is that MDX is not a language that is based upon extensive formal research, but is instead of the product of a corporate entity. As such, its structure can sometimes be obscure, with an “ad hoc” feel that makes it difficult to guarantee full IR/MDX equivalency. Our motivation therefore is demonstrate how MDX/IR translation works in the general case. We leave it to future work to address some of the possible edge cases. Finally, the Analyser and Evaluator module which explores, and modifies the query (if necessary) before evaluating it. Our framework employed a set of powerful and efficient methods to provide significant performance improvements for query checking. It can be sited on top of an OLAP server, so it provides reliable and useable DBMS.

To further ground the research, we have considered five distinct types of MDX queries: The simple form with members and children functions, the slicer specification with the WHERE clause, the drilling down and rolling up, queries with filters, and the calculated member's queries. As a second round of testing, we have employed a well-known benchmark APB-1, release I to demonstrate that the framework can translate various types of queries. The experimental analysis support the claim that translating and security checking can, in fact, be carried out without a meaningful impact upon final query execution times.

The rest of the paper is organized as follows. The related work is presented in Section 2, Section 3 describes briefly the architecture of our framework. Experiment results are presented in Section 4, and the final conclusions and future works are then offered in Section 5.

2 Related Work

Mathematical Researchers have considered security in relational databases for a long time [10,11]. Several defined frameworks are likely too restrictive for production databases. For instance, while view based access control approach is used to define the accessible elements for a set of users, it is not applicable when the number of users is large [20,21]. Fine grained access control has not received a lot of attention until recently [3], which allows to give access permits at more granularity level of rows or columns. Another performs access control enforcement in the application code instead of the database which needs to ensure that set of access control policy rules are consistent across different applications [22].

In contrast a number of security models that restrict data warehouse access have been proposed in the literature [12,13,14]. Some of them focus strictly on the design process. Others have extended the Unified Modeling Language to specify security constraints on multi-dimensional data [12]. In fact, a number of researchers have looked at similar techniques for setting access constraints at an early stage in the OLAP design process. Others have developed security requirements for the entire Data Warehouse life cycle [15]. In this case, they first propose a model (agent-goal-decision-information) to support the early and late requirements for the development of DWs, then extend that model to capture security aspects in order to prevent illegitimate attempts to access the warehouse. Such models have great value of course, particularly if one has the option to create the warehouse from scratch. That being said, their focus is not on authentication and authorization algorithms per se, but rather on design methodologies that that use the already existing technologies, such as: Model Driven Architecture (MDA) and Software Process Engineering Meta-model Specification (SPEM) [23,24].

Query rewriting has also been explored in DBMS environments in a variety of ways, with search and optimization being common targets [25]. Beyond that, however, rewriting has also been utilized to provide fine grained access control in Relational databases [16]. To answer a query Q, a masked versions of related tables are generated by replacing all the cells that are not allowed to be seen with NULL. After that, Q is evaluated as a normal query on the masked versions of the tables. This approach does not leak information not allowed to be seen, but it returns incorrect results when a query contains any negation, as expressed using the keywords MINUS, NOT EXISTS or NOT IN [17].

Ultimately, MDX is designed as a highly multidimensional expression syntax for querying and manipulating multidimensional data stored in OLAP cubes [28,29]. MDX was invented by Microsoft as part of the OLE DB for OLAP specification in 1997, then followed by commercial release of Microsoft OLAP Services 7.0 [30]. Many researches have been conducted on MDX in the field of Data Warehousing and Big Data [31,32]. They mainly focused on MDX as a tool for analysing and mining big data to extract useful patterns. Others are considered the recommending MDX queries to better navigating data cubes [26,27]. Authors are described their works theoretically and then validated through experiments.

On the other hand, researchers have provided security mechanisms and frameworks that are applied on the schema used by MDX. Specifically, portions of OLAP schema (total, dimension or cell) can be restricted from being viewed by defining a set of restrictions and then assigned these restrictions to certain user roles. Herein, only the permissible parts of the schema will be accessible by specified roles [5,6]. This is different from what we are presenting, our framework uses query re-writing technique to secure multi-dimensional OLAP environments by applying a series of rules that dynamically and transparently transform the query.

3 Framework Design

Our security framework serves as guard for data to perform a prerequisite security checking to decide which query should be passed for executing and which should be rejected without any optimization or resolution. In this section, we will describe briefly its components and the relationship between them by describing the security process as shown in Fig.1.

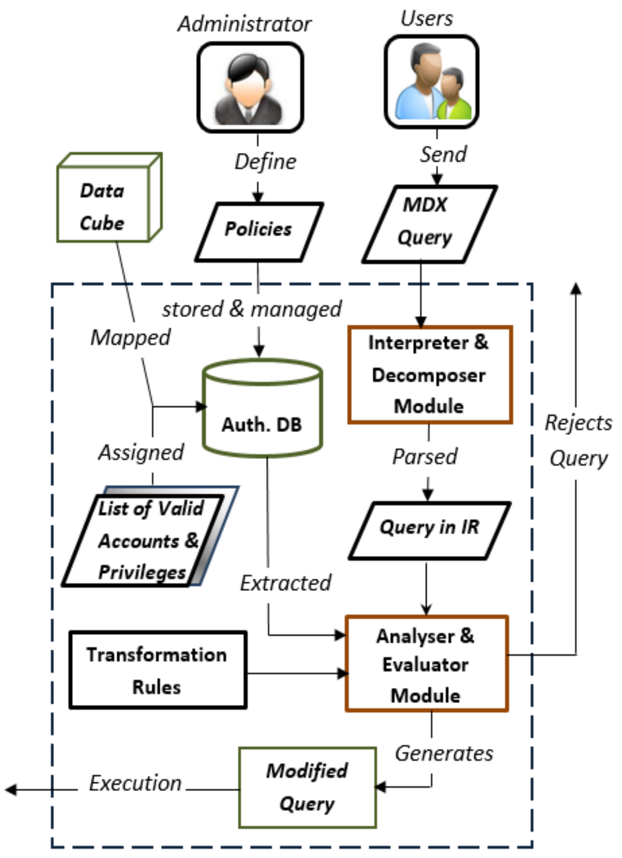

Figure 1. The Framework Architecture.

3.1 The security process

The proposed security process is a multistep:

- Receive the user query/request in MDX format. However, the framework does not understand the MDX, but an internal algebra that identifies the core operations associated with the data cube model. Thus, the Interpreter and Decomposer module (IDM) parses, interprets and decomposes the user query into the core operations of the internal algebra, which can be used to efficiently express queries, and easily reflect the changes that will be done by our model.
- The Analyser and Evaluator module (AEM) extracts the user identity/credentials and the requested elements (e.g., dimensions, attributes, hierarchies, etc.) from the query. Then the module authenticates the user by verifying her credentials against a list of valid accounts. If the provided credentials are valid, the authentication is successful.
- The module then determines if the user is permitted to access the requested resources. A set of query re-writing rules is applied to ensure consistent data access over the data cube. The query rewriting process is accomplished by adding or changing specific conditions within the query according to a set of concise but robust transformation rules.
- Once the query passes the previous checking, the query will be converted back to its original format and transparently delivered to the server for execution. Results are then returned to the user. However, the query will be rejected and the user will be informed otherwise.

3.2 Parsing the MDX Query

As mentioned in the previous subsection, the user's query is represented in MDX format. In order to properly authenticate the query, it must first be parsed and decomposed into its algebraic components. The purpose of this step is to determine if the query is semantically and syntactically valid and it is done in two phases:

In the first phase, the DOM parser utility is employed and used by the IDM to produce a DOM tree that represents the raw contents of the MDX query. In this phase, the parser not only builds the tree but also verifies that the received query has valid syntax corresponding to a DTD query grammar. A query is considered as valid if it contains only those elements defined in the DTD. If the query is syntactically valid, the query proceeds to the second phase. Otherwise, a parsing error message is returned to the user.

For example, suppose a marketing company is storing its data in a simple data cube called (Furniture Sales) with four dimensions (Customer, Supplier, Part and Date) and one fact table (Lineorder). Assume a user John queries the data cube that summarizes the total sales of Quebec's stores in 2011. The corresponding node tree is shown in Figure 2.

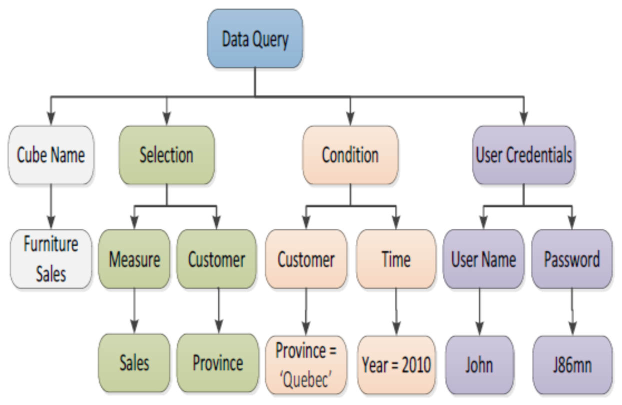

Figure 2. The DOM tree.

We can easily see that the content of this parse tree is equivalent to the original query. Specifically, it is executed against the cube Furniture Sales and consists of two OLAP operations (Projection and Selection). The projection operation returns the dimension attribute Customer.Province, as well as one measure attribute --- Sales. The Selection operation filters the returned information via two conditions on the dimensions Customer (i.e., Province = Quebec) and Time (Year = 2010). The user name ``John'' and the password ``J86mn'' represent the user credentials. This tree is stored as cached object in the memory for further processing.

In the second phase, the DOM tree is converted into an internal algebra representation (IR). This algebra represents all the query elements (i.e., returned attributes, query conditions along with its dimensions and attributes, and user credentials). The purpose of this conversion process is to transform the user query into a simple, minimal data structure that represents the query in a compact but expressive form. IR is discussed in details in the next section.

Once the parsing is completed, the Analyser and Evaluator module extracts the user credentials to verify them against a valid account stored in a back database. If the verification is successful, the DBMS proceeds with the authorization process. Otherwise, the query is rejected and the user is notified.

3.3 Converting the Parse Tree to an Internal Representation

While there are some similarity between MDX and SQL such as same keywords serving similar functions, there are some differences also. For example, in the SQL case we receive the data from the table, and in MDX case we receive data from the cube. Moreover, SQL basically gives us relational view of the data, which is always two-dimensional, while in the MDX world, any number of dimensions (axes) can be specified to form result of the query. It can zero, one, two, three or any other number of axes.

Before discussing the converting process, we give a brief description of the Internal Representation (IR), however a details description could be found on [18]. The IR identifies the core operations associated with the data cube model. For example, the SELECTION operator identifies one or more cells from within the full d-dimensional search space. Its application produces what is commonly referred to as "slicing and dicing". PROJECTION Operator, identifies the schema of the output cube, including both the measure attribute(s) and dimension members. IR also provides OLAP set operations (UNION, INTERSECTION and DIFFERENCE) that can be applied to data cubes. The FROM clause represents the cube source name.

For example, recall the Furniture Sales data cube with the four dimensions (Customer, Supplier, Part and Date) and one fact table (Lineorder). Assume the following simple query, which aggregates the discount on parts of category Cat#11 that sold in Canada on 2013. An equivalent query is then illustrated in Query' written in our algebra.

Query:
SELECT *[Measures].[LoQuantity] ON COLUMNS,*
[part].[pCategory].[Cat#11] ON ROWS
FROM *[Sales]*
WHERE *([date].[dYear].[2013],*
[customer].[cNation].[CANADA])

Query':
PROJECTION*:*
Attributes: part.pCategory
Measures: LoDiscount
SELECTION*:*
part.pCategory = Cat#11
date.dYear = 2013
customer.cNation = CANADA
FROM
Sales

It is important to mention that our IR can represents five main forms of MDX queries: Simple form with members and children functions, Slicer Specification with the WHERE clause, queries with drilling down and rolling up using descendants and ancestors functions, queries with filters, and queries with calculated members. In the next subsections we give a detailed descriptions of how our framework translate these forms.

1. *Simple form with members and children functions:*

The simple form of MDX queries consists of two main clauses (SELECT and FROM). These clauses serve exactly the same purpose they do in SQL. Indeed, the SELECT keyword tells what data will be returned back after query execution, the FROM keyword specifies the source of the data. For example, to show quantities of sales and there discounts over the Asian customer's region, the MDX query will look like:

Query1:
SELECT *[Measures].[LoQuantity],*
[Measures].[LoDiscount] ON COLUMNS,
[Customer].[cRegion].CHILDREN,
[Customer].[cRegion].[ASIA] ON ROWS
FROM *[Sales]*

There is no special meaning to rows and to columns in MDX other than for display purposes. Therefore in our example COLUMNS will be definition of the axis displayed vertically, and ROWS will be definition of the axis displayed horizontally. The CHILDREN and MEMBERS functions are used often in formulating expressions. The CHILDREN function returns the child members for a particular member within the dimension. The MEMBERS function returns the members for the specified dimension or dimension level.

To translate such query to our algebra, the measures and returned attributes are translated into the PROJECTION Element (i.e., to Attributes and Measures elements). If the query has specified members, they are translated to the SELECTION Element. Finally, the cube name is translated to the FROM clause. The translation of the previous query is illustrated next:

Query1':
***PROJECTION**:*
Attributes: Customer.cRegion
Measures: LoQuantity, LoDiscount
***SELECTION**:*
Customer.cRegion = ASIA
FROM
Sales

2. *Slicer specification with the WHERE clause:*

Slicing is one of the basic operations in data analysis. Slicing supported through the syntax of WHERE keyword. The WHERE clause in SQL different from WHERE in MDX. In the former, it means filtering of rows by specified criteria, while in the later means slicing the multidimensional query. If one were required to query the sales quantities for the year 2013, for the customers summarized at the region level, cross referenced against the part category, one would have to define a slicer specification. This requirement can be expressed by the following query:

Query2:
SELECT *[customer].[cRegion].MEMBERS ON COLUMNS,*
[part].[p Category].MEMBERS ON ROWS
FROM *[Sales]*
WHERE *(Measures.[LoQuantity], [date].[dYear].[2013])*

The slicer specification in the WHERE clause is actually a dimensional slice of the cube. Thus the WHERE clause can, and often does, extend to other dimensions (i.e., date dimension). To translate this kind of queries, the returned attributes are translated into the PROJECTION Element. If the slicer contains any condition, it will translate as new condition to the SELECTION Element. The translation of this query is similar to Query1 translation and thus no need to show it up.

3. *Drilling down and rolling up:*

The Descendants and Ancestors Functions are used to provide the ability to drill down to a lower level within the hierarchy or roll up to a higher level. These functions allows one to go to any level in the dimension hierarchy. Using DESCENDANTS it becomes possible to query the cube for information at the individual supplier nation level for AMERICA region, For example:

Query3:
SELECT *[Measures].[LoQuantity] ON COLUMNS,*
DESCENDANTS([Supplier].[suppHierarchy].[sRegion].[AMERICA],[Supplier].[suppHierarchy].[sNation]) ON ROWS
FROM *[Sales]*

To translate a query with Descendants function, the lowest level that all its members are returned is translated to the PROJECTION Element (i.e., nation), while the higher level that used to specify a value is translated to the SELECTION Element. The translation of Query3 will look like:

Query3':
***PROJECTION**:*
Attributes: Supplier.sNation
Measures: LoQuantity
***SELECTION**:*
Supplier.sRegion = AMERICA
FROM
Sales

The vice versa of Descendants is Ancestors, where a user can get the summarize information for a high level (i.e., region level) using a lower level (i.e. nation). For example:

Query4:
***SELECT** [Measures].[Lo Quantity]*
ON COLUMNS,
ANCESTORS([Supplier].[suppHiera rchy]. [s Nation].[CANADA],
[Supplier].[suppHierarchy].
[sRegion]) ON ROWS
***FROM** [Sales]*

To translate a query with Ancestors function, the highest level that all its members are returned is translated to the PROJECTION Element (i.e., nation), while the higher level that used to specify a value is translated to the SELECTION Element. The translation of Query4 is shown next:

Query4':
***PROJECTION**:*
Attributes: Supplier. sRegion
Measures: LoQuantity
***SELECTION**:*
Supplier. sNation = CANADA
***FROM** Sales*

4. Filters:
The Filter function returns the set that results from filtering according to the specified search condition. It evaluates the specified logical expression against each tuple in the specified set. The function returns a set that consists of each tuple in the specified set where the logical expression evaluates to true. The following example shows the top suppliers, defined by those who is supplied quantity exceed 30M:

Query5:
SELECT
FILTER([Supplier].[sRegion].members,
[Measures].[Loquantity] > 30000000)
ON ROWS,
[Measures].[Loquantity] ON
COLUMNS
***FROM** [Sales]*

To translate Query5, the condition on the quantity is mapped as condition on aggregation function. In other words, this condition is similar to *Having* clause in SQL format. The translation of the query is depicted next.

Query5':
***PROJECTION**:*
Attributes: Supplier. sRegion
Measures: LoQuantity
***SELECTION**:*
SUM(Loquantity) > 30000000
FROM
Sales

5. Calculated Members
An important concept when working with MDX expressions is that of calculated members. Calculated members allow one to define formulas and treat the formula as a new member. Suppose we have Query6:

Query6:
***WITH MEMBER** [Measures].[Profit] AS*
*'[Measures].[LoDiscount] **
[Measures].[LoQuantity]',
FORMAT STRING = '#.00%'

***SELECT** NON EMPTY*
([Date].[dYear].CHILDREN) ON
COLUMNS
***FROM** [Sales]*
***WHERE** ([Measures].[Profit],*
[Supplier].[sNation].[Canada])

The WITH MEMBER command defines a new measure that relates already defined measures. Thus, when we translate this command all the defined measures are extracted and mapped to Measures in the PROJECTION Element, also any returned attribute and/or condition should be mapped to its corresponding element. The translation of this query will look like:

Query6':
PROJECTION:
Attributes: Date.dYear
*Measures: LoDiscount*LoQuantity AS Profit*
SELECTION:
Supplier.sNation = Canada
FROM
Sales

3.3 The Transformation Rules

When a user requests access to a particular resource, the request is validated against the permitted resource list assigned to that user. If the requested resource produces a valid match, the user request is allowed to execute as originally written. Otherwise, the query will either be rejected outright or modified according to a set of flexible transformation rules. In other words, in our framework the AEM takes the responsibility for authentication and authorization. AEM is first extracts the user's credentials and validates them against a valid list. If the user is authenticated, the requested resources in the query are then verified against the accessible resources assigned to that user. If the requested resource is not permitted, the query is rewritten in order to provide the valid resources. To do this, AEM is based on a *query rewriting* technique that rewrites queries containing unauthorized data access to ensure that the user only receives the data that he/she is authorized to see. Rewriting is accomplished by adding or changing specific conditions within the query according to a set of concise but robust transformation rules. In case of query modification, the user is informed by a warning message that telling him/her that the query is modified during security concerns. However, in this case, the user should not know what he/she is restricted from, since this may be used as external information in some case to infer the protected data. Due to the space limitations, we just give a summarization for the transformation rules in terms of its three possible outcomes *Execute*, *Modify*, and *Reject*. The formalization of these rules along with their proofs can be found in [19].

- The query is allowed to execute without modification in two situations. The first situation when a hierarchy level say L_i is restricted and there is an exception *E*, here if any upper level exists in the *Selection* or *Projection* query element, OR if the L_i value or any value from the levels below it exists in the *Condition* element AND this value is equal to the exception value *Ev* or any value under it, the query is then executed without modification. In the second situation, when a specific value of L_i is restricted and there is an exception *E*, here the query is also executed without modification if the prohibited value *Lv* or any value under it exists in the *Condition* element AND it is equal to the exception value *Ev* OR any value under it.

  For example, suppose that we have the following policy: An analyst Alice is invited to analyse the Furniture Sales data. However, Alice is restricted from viewing the suppliers' quantities except for the Canadian suppliers. If Alice submits the query depicted next. The query will be executed without any modification because Alice is asking to view the quantity sold for a more detailed child level of her exception (e.g., Quebec is province in Canada).

Query7:
PROJECTION:
Attributes: Date.dYear
Measures: LoQuantity
SELECTION:
Supplier.sProvince = Quebec
FROM
Sales

- However, the query is modified in one situation, when a hierarchy level say L_i is restricted and there is an exception E, here the query is modified if level L_i or any value from the levels below it exists in the query *Selection* element only, then we add the exception *E* as a new condition, OR if the exception value *Ev* belongs to the values under *Lv*, then we replace the prohibited level in the *Condition* element by the exception *E*.

  For example, suppose that the previous policy is changed as follows: Alice is now restricted from viewing the suppliers' quantities except for the suppliers of Montreal city. Assume that Alice resends Query7. It will be modified by condition (e.g., *Supplier.sProvince = Quebec*) in the Condition element with the exception (e.g., *Supplier.sCity = Montreal*). In this example, Alice gets only the values that she is allowed to see. The modified query will look like Query7'.

Query7':
***PROJECTION**:*
Attributes: Date.dYear
Measures: LoQuantity
***SELECTION**:*
Supplier.sCity = Montreal
FROM
Sales

- Ultimately, the query is rejected in two situations. The first situation when a level L_i is restricted, and there is no exception, then if the level L_i or any value from a lower level exists in the *Selection* element only, OR if the level L_i or any value from the levels below it exists in the *Condition* element the query is rejected. The query is also rejected when a specific value P is restricted, and there is no exception or if P or any value under it exists in the *Condition* element.
  For example, if Alice's exception is no more exited and she resubmit Query7, the query will be rejected.

4 Experiments

In this section, we investigate the efficiency of the MDX query translation. Specifically, we provide initial performance results for the MDX conversion middleware. However, in the second part of this section, we investigate the queries checking times to demonstrate that translation times are acceptable comparing to the queries execution times.

For the following results, all testing was conducted on a 12-core AMD Opteron server with a CPU core frequency of 2100 MHz, L1/L2 cache size of 128K and 512K respectively, and a shared 12MB L3 cache. The server was equipped with 24 GB of RAM, and eight 1TB Serial ATA hard drives in a RAID 5 configuration. The supporting OS was CentOS Linux Release 6.0. All OS and DBMS caches were cleaned between runs. The machine is coupled with a column store database management systems (MonetDB) [33]. This system have been shown to perform more than an order of magnitude faster than traditional row-oriented database systems (row-stores) for large, read-intensive data repositories such as those found in data warehouses, decision support, and business intelligence applications that support analytical workloads. The reason behind this performance advantage is straightforward: column-stores are much more I/O efficient for read-only queries since they only have to read from disk those attributes actually referenced (directly or indirectly) by a query [33].

We begin by defining the 10 MDX queries listed in Appendix A. These queries cover the five main forms of MDX queries discussed previously. All queries are translated into the intermediate IR representation before undergoing the checking process. Figure 3 depicts the translation time. In all cases, the translation process takes a small amount of time, in the range of 27-56 milliseconds. In fact, these times are acceptable comparing to the queries execution times.

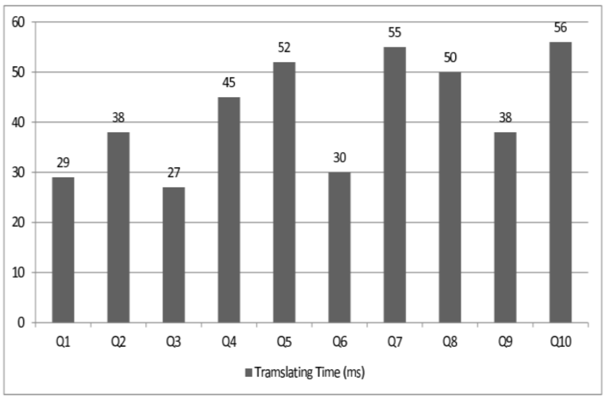

Figure 3. Translating Time of 10 MDX Queries.

To demonstrate the validity of our work, we utilized a popular benchmark (namely, APB-1 benchmark, release I) [34]. In this round, the APB database generator (with a channel of 40) was used to load two fact tables: Actvars, with approximately 86 million records, and Planvars with approximately 62 million records. The fact tables are joined to four dimension tables: Product, Customer, Time, and Channel, each housing up to 36000 records. Again, we focus on the translation and also the execution times. In this case, we consider 10 analytic queries, with each providing sophisticated restrictions on the associated dimensions (i.e., some queries have only one condition, others have up to four conditions). The queries themselves are executed against two fact tables (e.g., Actvars and Planvars tables). The full query set can be found in [34]. APB queries are needed to be translated into our algebra. Figure 4 shows the translation time. As seen, the translation costs are quite small, in the range of 38-75 milliseconds.

In terms of the checking results, we have isolated the APB queries into two query classes based on the target fact table. For example, Figure 5 shows the ratio of processing cost to query execution time against the Actvar fact table, while Figure 6 shows the results for the Planvars fact table. It is worth to mention that the query execution time is still listed for queries that violate the policy and were not candidates for re-writing, in order to give a better sense of the relative balance between checking and execution.

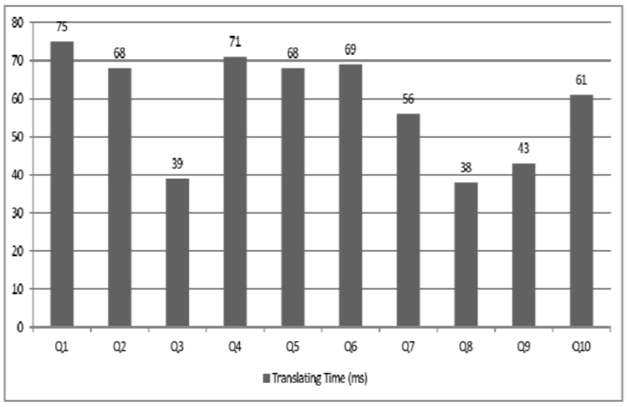

Figure 4. Translating Time of APB Queries.

It should be clear that the checking time is again quite small, in the range of 39-600 milliseconds. This implies that little or no I/O is required during the checking phases once the DBMS process has started. This is the case since the framework can pre-load with the appropriate metadata. For the queries, however, the benchmark database is extremely large and we cannot expect them to be preloaded into memory without enormous hardware resources. In other words, there are strong limits on the amount of optimization that can be performed during the execution phase. We also note that, in terms of this specific test case, the conditions of Query 1 and Query 8 do not violate the policy constraints and thus the checking times for each of them are less than for other queries.

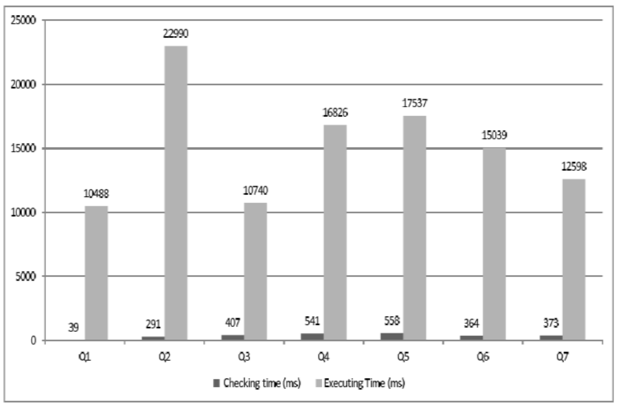

Figure 5. Performance for APB Queries against Actvars table.

As a final point, we re-iterate that our framework is efficiently practical. Specifically, it does not impact query performance in a meaningful way. As such, it is possible to plug-in any standards-compliant DBMS server. For test purposes, in fact, this can be an advantage as it provides more intuitive test results and underscores the potential for integration with standard database servers.

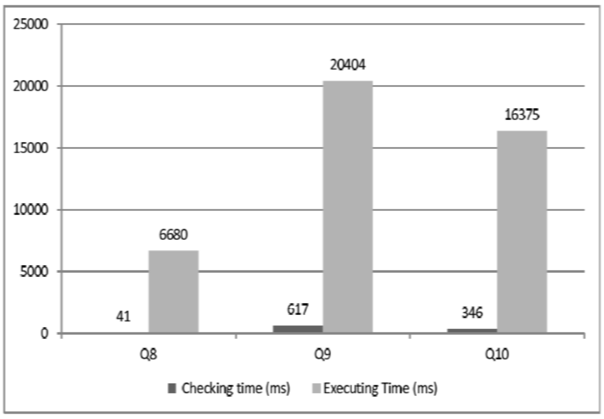

Figure 6. Performance for APB Queries against Planvars table.

5 Conclusion and Future Work

In this paper, we have discussed an integrated security framework for querying Multidimensional data with MDX. We began by describing a set of transformation rules that can be used to provide data access functionality for DBMSs that use the popular MDX query language. We then presented the general translation process for MDX queries being mapped into an intermediate representation (IR). We specifically discussed how the IR could support five general forms of MDX queries. To demonstrate the validity of our work, we have provided initial test results indicating that the process takes a small amount of time, which are acceptable comparing to the queries execution times. Then we coupled our framework with MonetDB a popular column-store database systems and used a well-known benchmark APB to demonstrate that the framework can translate various types of queries.

As a future work, we are planning to extend the functionality of the current research in twofold. First, at present the framework is supporting five main MDX queries, the next goal is to cover all forms of MDX functions types. Second, to validate whether our framework are properly designed using model testing driven so the common security patterns are well applied and assessed. Our hope is that this will eventually lead to provide more secure systems, through refinement of this work.

Acknowledgment

The authors are grateful to the Applied Science Private University, Amman-Jordan, for the full financial support granted to cover the publication fee of this research article.

References

[1] Bender, Gabriel, Lucja Kot, and Johannes Gehrke. "Explainable security for relational databases." *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014.

[2] Stern, Daniel A. "Balanced Design in Information Systems Security Planning."*The International Conference in Information Security and Digital Forensics*. The Society of Digital Information and Wireless Communication, 2014.

[3] Bhatnagar, Neerja. "Security in Relational Databases." *Handbook of Information and Communication Security*. Springer Berlin Heidelberg, 2010. 257-272.

[4] Ferrari, Alberto, Marco Russo, and Chris Webb. Microsoft SQL Server 2012 Analysis Services: The BISM Tabular Model. Pearson Education.

[5] Oehler, Karsten, et al. *IBM Cognos TM1*. McGraw-Hill, 2012.

[6] Mondrian OLAP server. http://community.pentaho.com/projects/mondrian

[7] Ruiz, Jose R. *Oracle Essbase 11 Development Cookbook*. Packt Publishing Ltd, 2012.

[8] Johari, Rahul, and Pankaj Sharma. "A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection."*Communication Systems and Network Technologies (CSNT), 2012 International Conference on*. IEEE, 2012.

[9] Tajpour, Atefeh, et al. "Effective Measures for Evaluation of SQL Injection Detection and Prevention Tools." *JCIT: Journal of Convergence Information Technology 8.14 (2013): 13-28.*

[10] Bertino, Elisa, and Ravi Sandhu. "Database security-concepts, approaches, and challenges." *Dependable and Secure Computing, IEEE Transactions on2.1 (2005): 2-19.*

[11] Elmasri, R. Navathe. Fundamentals of database systems. Pearson, 2014.

[12] Eduardo Fern´andez-Medina, Juan Trujillo, Rodolfo Villarroel, and Mario Piattini Developing secure data warehouses with a uml extension. *Inf. Syst., 32(6):826–856, September 2007.*

[13] Altamimi, Ahmad, and Todd Eavis. "PICM: A practical inference control model for protecting OLAP cubes." *Web Applications and Networking (WSWAN), 2015 2nd World Symposium on. IEEE, 2015.*

[14] Altamimi, Ahmad, and Todd Eavis. "OSSM: The OLAP Security Specification Model. " *Databases Theory and Applications. Springer International Publishing, 2014. 26-37.*

[15] Kimball, Ralph. The data warehouse lifecycle toolkit. *John Wiley & Sons, 2008.*

[16] Rizvi, Shariq, et al. "Extending query rewriting techniques for fine-grained access control." *Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM, 2004.*

[17] Wang, Qihua, et al. "On the correctness criteria of fine-grained access control in relational databases." *Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007.*

[18] Altamimi, Ahmad. "Securing OLAP Cubes". *Lambert Academic Publishing, Germany, 2015. ISBN 978-3-659-39290-0*

[19] Altamimi, Ahmad, and Todd Eavis. "Securing Access to Data in Business Intelligence Domains." *International Journal on Advances in Security Volume 5, Number 3 & 4, 2012.*

[20] Xiaolei Qian, "View-based access control with high assurance," *in Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on , vol., no., pp.85-93, 6-8 May 1996*

[21] Bender Gabriel, Kot Lucja, and Gehrke Johannes. "Explainable Security for Relational Databases." *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, 2014.*

[22] Toledo R., Nunez A. Tanter E., and Katz S. "Aspectizing Java Access Control," *in Software Engineering, IEEE Transactions on , vol.38, no.1, pp.101-117, Jan.-Feb. 2012*

[23] Stahl Thomas, and Markus Völter. "Model-driven Software Development." *Technology, Engineering, Management. Chichester, England: J. Wiley & Sons, 2013.*

[24] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. "Model-Driven Software Engineering in Practice." *Synthesis Lectures on Software Engineering 2012 1:1, 1-182*

[25] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. “Query Rewriting and Optimization for Ontological Databases.” *ACM Trans. Database Syst. 39, 3, Article 25, 2014.*

[26] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. “A framework for recommending OLAP queries.” *In Proceedings of the ACM 11th international workshop on Data warehousing and OLAP(DOLAP '08). ACM, New York, NY, USA, 73-80.*

[27] Saida Aissi, Mohamed Salah Gouider, Tarek Sboui, and Lamjed Ben Said. “Personalized recommendation of SOLAP queries: theoretical framework and experimental evaluation.” *In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). ACM, USA, 2015.*

[28] George Spofford, Sivakumar Harinath, Chris Webb, Dylan Hai Huang, Francesco Civardi: “MDX-Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase.” *Wiley, 2006, ISBN 0-471-74808-0*

[29] Mosha Pasumansky, Mark Whitehorn, Rob Zare. “Fast Track to MDX.” Springer, 2006, ISBN 1-84628-174-1

[30] Carl Nolan. "Manipulate and Query OLAP Data Using ADOMD and Multidimensional Expressions." *Microsoft. Retrieved 2008.*

[31] Alfredo Cuzzocrea, Ladjel Bellatreche, and Il-Yeol Song. “Data warehousing and OLAP over big data: current challenges and future research directions.” In *Proceedings of the sixteenth international workshop on Data warehousing and OLAP* (DOLAP '13). ACM, USA, 2013.

[32] Alfredo Cuzzocrea. “Aggregation and multidimensional analysis of big data for large-scale scientific applications: models, issues, analytics, and beyond.” *In Proceedings of the 27th International Conference on Scientific and Statistical Database Management, 2015.*

[33] Boncz Peter, Zukowski Marcin, and Nes Niels. “Monetdb/x100: Hyper pipelining query execution.” 2005.

[34] OLAP Council: APB-1 OLAP Benchmark, Release ii, 1998. http://www.olapcouncil.org/research/bmarkly.htm.

Appendix A

• **Query1**:

```
SELECT {[Measures]·[loquantity],
[Measures]·[lodiscount]} ON COLUMNS,
{[customer]·[cregion]·CHILDREN} ON ROWS
FROM[Sales]
```

• **Query2**:

```
SELECT{[Measures]·[loquantity],
[Measures]·[lodiscount]} ON COLUMNS,
{[customer]·[cregion]·[ AMERICA]} ON ROWS
FROM [Sales]
```

• **Query3**:

```
SELECT{[date]·[dyear]·[1998]} ON COLUMNS,
{[customer]·[c nation]·[ CANADA],
[customer]·[cnation]·[ UNITEDSTATES]}ON
ROWS
FROM [Sales] WHERE ([Measures]·[lo discount])
```

• **Query4**:

```
SELECT{[Measures]·[lodiscount]} ON
COLUMNS, {[part]·[pcategory]·[ MFGR#11]} ON
ROWS
FROM [Sales]
WHERE ([date]·[dyear]·[1998],
[customer]·[cnation]·[ CANADA])
```

• **Query5**:

```
SELECT [Measures]·[loquantity] ON COLUMNS,
{DESCENDANTS([supplier]·[sregion - snation -
scity]·[sregion]·[ AMERICA],
[supplier]·[sregion - snation - scity]·[snation])} ON
ROWS FROM [Sales]
```

• **Query6**:

```
SELECT {[Measures]·[lo quantity]} ON
COLUMNS, {CROSSJOIN([part]·[p
brand1]·CHILDREN, [date]·[dyear]·[1998])} ON
ROWS FROM [Sales]
```

• **Query7**:

```
SELECT NON
EMPTY(FILTER({[customer]·[cregion]·CHILDRE
N},[customer]·[cregion]·CurrentMember·Name <>
' AMERICA')) ON COLUMNS,
NON EMPTY(FILTER({[date]·[dyear]·
CHILDREN}, [date]·[dyear]·CurrentMember·Name
```

> '1997')) ON ROWS FROM [Sales] WHERE
[Measures]·[lotax]

• **Query8**:
WITH MEMBER [Measures]·[profit] AS
'[Measures]·[lodiscount] *[Measures]·[loquantity]'
SELECT NON
EMPTY([date]·[dyear]·CHILDREN) ON
COLUMNS FROM [Sales] WHERE
([Measures]·[profit], [supplier]·[s nation]·[
Canada])

• **Query9**:
SELECT {[Measures]·[lo quantity] } ON
COLUMNS, FILTER
({[supplier]·[sregion]·MEMBERS}, [Measures]·[lo
quantity] > 30000000) ON ROWS
FROM [Sales]

• **Query10**:
SELECT {[Measures]·[lo discount], [Measures]·[lo
quantity]} ON COLUMNS,
FILTER(CROSSJOIN({[date]·[d
year]·[1995]:[date]·[d year]·[1997]}, {[part]·[p
mfgr]·CHILDREN}), [Measures]·[lo quantity] >
4670000) ON ROWS
FROM [Sale]