CAL: A Generic Query and Analysis Language for Data Warehouses

Ganesh Viswanathan & Markus Schneider
University of Florida, Gainesville, FL 32611, USA
{gv1, mschneid}@cise.ufl.edu

Abstract

Data warehouses (DWs) and OLAP systems are essential tools in organizational decision making. However, users of these systems are often provided with complex data models and system-specific GUIs and dashboards to analyze the data and generate reports. Available DW query languages like MDX and Oracle OLAP support only numeric data and moreover require skilled DW developers to design queries. This can strain the analyst and restrict effective data analysis. In this paper, we provide a solution to this problem by introducing a powerful textual query language based entirely on the abstract datacube metaphor for multidimensional analysis. This language called the Cube Analysis Language (CAL) captures the full dimensionality of the data while providing an extensible query interface for the analyst. CAL includes three basic components: the Cube Definition Language *(*CDL*), to design and develop the cube, the* Cube Manipulation Language *(*CML*), to manipulate and alter the cube, and the* Cube Query and Analysis Language *(*CQAL*), to navigate through cubes and aggregate data in the DW for analysis. CDL, CML and CQAL are designed to be explicit user-level concepts that work easily with existing OLAP languages and logical DW designs. To demonstrate its functionality we have implemented the CAL query processor for Mondrian (an open-source OLAP tool) and tested it using an example business sales dataset on Postgres.*

1 Introduction

In recent years, several new and innovative front end applications have been developed for performing business intelligence (BI) and online analytical processing (OLAP). These include several GUI tools that help to analyze, and query databases and spreadsheets as a means to gather information for organizational decision making. Data warehouses serve as the foundation repositories of massive amounts of data that are collected from several heterogeneous sources over a large period of time in an organization. These large repositories store valuable information that can be mined using OLAP query tools for knowledge gathering. However, there is the lack of a generic user interface to enable analyst users to query data from the data warehouse by just using an abstract cube interface. The existing query languages popularly used in multidimensional databases such as Oracle OLAP and MDX are based on unique logical implementation models, and can often be inappropriate for the user of a general purpose data warehouse. In recent years, MDX has become the defacto standard for textual OLAP languages. However, it is not often used in applications or industry due to several problems. First, it requires the user to have good knowledge of the underlying implementation details for performing unions and crossjoins. Second, it does not provide a global view of the schema but requires the user to have the multidimensional schematic of data before querying. Thirdly, MDX can often get complicated due to several levels of ordered crossjoins in the query. Thus, often the query itself cannot be written or validated directly, instead requiring a MDX parser and implementation for testing its validity. We found several of the existing MDX implementations to be rather different in their levels of functionality and support, leading to many errors and incompatible syntax issues.

In this paper, we aim to solve these problems associated with existing query languages for data warehouses, by introducing the Cube Analysis Language (CAL). CAL is an abstraction over standard SQL aggregate extensions and MDX, and helps the user to design, build and manipulate data cubes in a fully multidimensional environment. CAL also includes a query and analysis component that can be seamlessly extended to support complex analysis operations. These are later translated into logical components and implemented using existing technologies such as Relational OLAP or Multidimensional OLAP.

The rest of this paper is organized as follows. Section 2 discusses related work and summarizes the available query language approaches for data warehouses. In Section 3, we introduce a sample product-sales data warehouse example that is used in the rest of the paper. Section 4 presents the Cube Analysis Language (CAL) for datacube construction, manipulation and querying. Section 5 provides a comparison of CAL and MDX and motivates the need for a generic user-level textual analysis interface for data warehouses. In Section 6 we introduce our working prototype for CAL and conclude the paper in Section 7 along with a discussion of future work.

2 Related Work

There have been a few proposals for OLAP query languages intended for multidimensional and statistical databases. For example, the syntax and definition of a Data Mining Query Language (DMQL) for the DBMiner System is presented in [2]. DMQL supports concept hierarchies on the dimensions of the cube. The language provides a framework for mining through multiple levels of data using characteristic, discriminant, classification, and association rules. Since the language design is too specific, it is not complete in a data warehouse context. However, DMQL serves as an interesting querying platform for OLAP.

The specification for SumQL, a query language for Summary Databases (SDB) is provided in [5]. SumQL helps the user to pose aggregate queries over SDBs by taking advantage of the semantic properties of the summary data model. SumQL is an SQL-like query language that includes constructs to reflect SDB concepts such as dimensions, hierarchies over categories and aggregation. However, SumQL is provided as a language for *summary databases*, and it does not explicitly include data warehousing and OLAP functions such as slicing, dicing, etc., and hence the user has to design complex queries for handling such operations.

MultiDimensional eXpressions (MDX) [4] is a powerful language for multidimensional data querying and is now supported by numerous BI and OLAP system vendors. However, MDX can be complicated for a unskilled user of a data warehouse, since it does not specify conventional OLAP operations such as slice or dice explicitly, but instead forces the user to perform join data across several dimensions (using multiple crossjoins among its members) to arrive at the required aggregation state. MDX has been widely adopted and is supported by Microsoft, Microstrategy, Whitelight, SAS, SAP and other popular analysis services vendors.

Another multidimensional query language called the Cube Query Language (CQL) is presented in [1], that is designed for statistical and scientific databases (SSDB), particularly the CROSS-DB system [3]. CQL is an extension of SQL that supports multidimensional querying and processes queries in a two-step fashion, with a data querying (GET) step followed by a data presentation (SHOW) step. It enables the use of *features* to further refine the selection process. It allows selection predicates not only on classification nodes, but also on feature values, that characterize the single items. However, CQL does not provide a generic interface for conventional data warehouse operations.

3 $\mathcal{B}ig\mathcal{C}ube$ Data Model and Example

We introduced the $\mathcal{B}ig\mathcal{C}ube$ data model for multidimensional analysis in [7]. Here, we provide a query language for $\mathcal{B}ig\mathcal{C}ube$ called the Cube Analysis Language (CAL) that provides a complete new syntax for multidimensional data modeling, navigation and analysis using the abstract hypercube metaphor. This allows for a clear and user-centric data warehouse implementation. In this Section, we present a Product Sales example that is used in the rest of the paper to demonstrate CAL functionality.

Consider an example Sales dataset from a supplier, storing order information for the sale of products, to several customers. Products are categorized by their name and brand. Each product also has type and color information. The cost price for each product is maintained by the supplier, along with its available quantity and the manufacturing location, as part of the inventory. The invoice issued by the seller contains the selling price and quantity of products ordered by the customer. Once an order is place, the order commit time is recorded for each day, month, week and year. The store location is also recorded for shipping purposes and includes city, zone, state and country information. Apart from this, the supplier also records the profit resulting from the sales of a certain quantity of products, to capture and track their overall sales.

In $\mathcal{B}ig\mathcal{C}ube$, *sales* represents an enterprise data warehouse containing order information from the sale of products to customers. Based on user requirements, we identify three *subjects* for analysis (based on user requirements), namely, Inventory, Invoice, and Sales. Inventory is quantified by the following measures: cost_price, manufactured_at and unit_ sales. Similarly, Invoice and Sales have their own measures. The *perspectives* for data analysis include products, time and location. Time has the members days, weeks, months and years ordered into hierarchies, as determined by the user. Similarly, location also includes its constituent hierarchies. This illustrates the conceptual structure of the *sales* data warehouse.

4 CUBE ANALYSIS LANGUAGE (CAL)

The Cube Analysis Language consists of three components: *CDL* or *Cube Definition Language* to design and construct the cube structures, *CML* or *Cube Manipulation Language* to alter and modify cube schema, and *CQAL* or *Cube Query and Analysis Language* to navigate through cube instances and query it for analysis. The basic arithmetic operators in CAL include addition (+), subtraction (-), multiplication (x), division (/), power, and parentheses for grouping. CAL also supports regular expressions in its syntax and uses the *dot notation* for accessing the components of the cube.

4.1 Cube Definition Language (CDL)

The Cube Definition Language (CDL) helps to design and construct data cube in a conceptual manner. The underlying type system presents an object-oriented view of the cube structure that helps to model the multidimensional dataset with high expressiveness. Each cube contains a set of subjects and perspectives. Subjects have associated measures and perspectives have associated categories of members forming data hierarchies as presented in [7]. The list of members is unique for each perspective. The list of subjects and perspectives is unique for each cube. Cube creation is achieved using the CREATE clause, modification of cube structure is achieved using ALTER, and DROP is used for deletion as illustrated by the examples below.

```
CREATE CUBE Sales (
  SUBJECT Invoice   (Selling_Price   decimal(20,2),
                     Sales_Quantity  int)
  SUBJECT Inventory (Cost_Price      decimal(20,2),
                     'Unit Sales'    int,
                     Manufactured_At char)
  SUBJECT Sales     (Profit          profit_t,
                     'Unit Sales'    int)
       SHARED (Sales.'Unit Sales', Invoice.Quantity)
  PERSPECTIVE  Product (PName    char,
                       Brand    char)
       ATTRIBUTE    (PColor    char,
                     PType     char)
  PERSPECTIVE  Time (Day       int,
                     Month     char,
                     Week      int,
                     Year      int)
    HIERARCHY (Day,Month,Year) AS Time_H1
    HIERARCHY (Day,Week,Year) AS Time_H2
  PERSPECTIVE  Location (City  point,
                  County       region,
                  State        char,
                  Region_1     char,
                  Region_2     char,
                  Country      char),
    HIERARCHY (City,County,Country) AS Loc_H1
    HIERARCHY (City,State,Region_1,Country) AS Loc_H2
    HIERARCHY (City,State,Region_2,Country) AS Loc_H3 );
```

During the logical design phase (Section 6), for relational implementation the attributes for the product dimension become part of separate tables linked to product tables by foreign keys. Thus each product (identified by name) would have unique color and type attributes. Members and measures in a cube can be defined using alpha-numeric data types or spatial types as supported by the underlying database system or extensibility mechanism. In the above example, *map generalization* is applied on the Location perspective during rollup of city (point) values (spatial data) to state and higher levels (alphanumeric data).

The ALTER ADD and ALTER MODIFY clauses can be used to add new components and modify existing components of a cube respectively. Each perspective defines a new axis on the multidimensional cube. However, since combination of perspectives determines the set of measures in the cells of the cube, the cube itself changes and has to be reconstructed (except for ALTER RENAME). The following examples illustrate the working of the ALTER clause.

```
ALTER Sales ADD SUBJECT Supplier (ShipDate datetime);
ALTER Sales.Location  ADD HIERARCHY
    (City,Country,Continent) AS Loc_H4;
ALTER Sales DROP PERSPECTIVE Supplier;
```

The USING clause provides a mechanism to create duplicates of the cube structure. When used with the CREATE CUBE clause, all the data values are also conceptually duplicated by default. With a NULL suffix, the subjects, perspectives and associated hierarchies in the newly created cube (its structure) match those of the source but the values are null as in the first query below.

```
CREATE CUBE Sales_Orders USING Sales NULL;
CREATE SUBJECT Sales_Orders.Order_Invoice
    USING Sales.Invoice;
```

The DROP CUBE statement drops the entire cube along with all its set of subjects and perspectives.

```
DROP CUBE Sales_Orders;
```

4.2 Cube Manipulation Language (CML)

Once the data cubes have been created, CML commands are used to populate the cubes with member and measure values. The data loading processing in a data warehouse is often complex and consists of several Extraction, Transform and Load (ETL) phases. CML commands provide an alternative textual interface to achieve these operations. The INSERT statement is used to add new data into a cell of the data cube. Inserting new members can lead to several empty cells (for measures) in the data cube. Inserting the measure values in cells is achieved by explicitly specifying a combination of perspectives, using the PERSPECTIVE clause following the INSERT INTO ... VALUES clause.

```
INSERT INTO Sales.Inventory
  VALUES (200.50, 50, 'Orlando')
  PERSPECTIVE Location.City = 'Boston' AND
           Time.Year = 2011 AND
           Product.Name = 'TiVo';
```

The UPDATE-SET statement helps to make changes to data values that already exist in the cube. Updating members may or may not affect the measure values in the cells of the cube.

```
UPDATE Sales
SET Inventory.Quantity = 200
WHERE (Location.City='Boston' AND
    Time.Year=2011 AND Product.Name='TiVo');
```

The DELETE statement helps to remove data (member or measure value) from the cube. The example shows the removal of all data related to the sales of iPhone at Boston on the first of every month.

```
DELETE FROM Sales
WHERE (Location.City='Boston' AND Time.Day=01
    AND Product.Name='iPhone');
```

The TRUNCATE command clears the entire cube by removing all the measure values. Truncate can also be used on the perspectives of a cube, but since this removes all the members from that perspective, the cells of the cube become null.

```
TRUNCATE Sales.Invoice;
```

4.3 Cube Query and Analysis Language (CQAL)

The Cube Query and Analysis Language (CQAL) helps in navigating and querying data cubes and performing analysis. The primary command in CQAL is the GET clause which is used to retrieve measure values from the data cube (similar to SELECT in SQL for relations). The *period* or *dot operator* can be used to implicitly drill-down or roll-up on the levels of the perspective (wherever hierarchies have been defined). The period operator references the cube components using their names in the data warehouse multidimensional space. The AS clause at the end of the query helps to store results and build multi-level queries. Backticks are used whenever spaces are present in the names of cube components. The ON <axis_number> operator helps to format the resultant multidimensional data cube along its axes. Axis numbers in CAL start from 1 which refers to the columns in the report. However, since only two-dimensional reports are convenient for visualizations, automatic joins of members are achieved from left to right as specified in the query. The TOP n operator can be used with GET to obtain the top n results from the query.

Slicing is a simple form of selection applied on OLAP cubes. During slicing one or more perspectives can be removed from the cube using a set of equality constraints on its perspectives. Members along different perspectives can also be filtered by slicing. Consider a query to show the inventory of all electronics products at Boston in 2011. SLICE allows both the dot notation and filtering via equality as shown below.

```
GET Sales.Inventory
  SLICE Product.PType = 'electronics'
  SLICE Location.City = 'Boston'
  SLICE Time.Year.2011
FROM Sales;
```

The DICE clause allows for the creation of subcubes that are based on a combination of members to yield a small set of measure values. DICE may not reduce the number of perspectives, just yielding a smaller cube. Unlike SLICE, DICE also allows filtering of multiple members from within the same perspective.

```
GET Inventory.Cost_Price NOT NULL
  DICE (Time.Year > 2011) AND
  (Location.County LIKE 'S%') FROM Sales;
```

This query returns a smaller data cube (subcube) that shows only sales measure vales (both profit and quantity) from counties having a starting with 'S' and for the period after 2011.

ROLLUP is an aggregation (generalization) operator helps the user to rollup across the levels of a hierarchy by specifying the current level (state) in the hierarchy, and the final level or the number of levels to rollup for generalization. Since the hierarchy chosen is also explicitly specified, aggregation proceeds correctly even if multiple aggregation paths exist for the same perspective. If the rollup reaches the apex cube, the highest level values are returned. Referring to the CREATE CUBE statement in Section 4.1, we can see that both queries shown below perform the same rollup operations and return similar results. The syntax of the rollup operator is: ROLLUP perspective.hierarchy (currrent_ level, final_ level);

```
GET Sales.'Unit Sales'
    ROLLUP Location.Loc_H2 (City, Country)
    FROM Sales;
```

DRILLDOWN is a specialization operator helps the user to drill down the levels of a hierarchy by specifying the current level (state) in the hierarchy, and the final level or the number of levels to step down for specialization. If the drilldown reaches the base cube, the lowest level values are returned. Both examples shown below perform drilldown to the same *city* level but along two different hierarchies, thus yielding different aggregate (measure) results in the cube. The syntax of the drilldown operator is: DRILLDOWN perspective.hierarchy (current_ level, final_ level);

```
GET Sales.'Unit Sales'
    DRILLDOWN Location.Loc_H3(Country, City)
    FROM Sales;
```

The DRILLTHROUGH clause helps to retrieve the base data by drilling through the base cubes into the relational databases. In the following example, we get the sales data for each city (which is the lowest category level of the location perspective) using the existing representation across the axes.

```
GET Sales.‘Unit Sales’
   DRILLTHROUGH Location AS "data1.dbfs";
```

The DRILL ACROSS clause involves access across data cubes using the type checking on the perspectives or subjects of the associated cubes. The following example helps to obtain sales data for all cities by using two structurally similar cubes, each having a distinct set of location values.

```
GET Sales.Quantity
  DRILLACROSS Sales.Location.City,
       Sales_Orders.Location.City;
```

Statistical Aggregation Operators: CQAL includes a collection of predicates for many aggregation operators that are supported based on the available data types and operations in the underlying logical implementation model (Section 6). Some examples of predicates for aggregations for conventional alphanumeric data include sum, count, min, max, and avg. The following query shows the quantity of sales of electronics products in the UK, from 2010. This example uses a combination of many available CAL operators as an illustration of the flexibility of the query language for multidimensional data analysis.

```
GET COUNT(Sales.‘Unit Sales’)
  SLICE Product.PType = ‘electronics’
  DICE Time.Year > 2010 AND Location.City LIKE ‘London’
  ROLLUP Location.Loc_H2 (City, 3)
AS C_SALES_elecUK2010;
```

5 Advantages of CAL over MDX

Multi-Dimensional eXpressions (MDX) has become popular in recent years and is supported by several major BI vendors. However, in these systems MDX is not used directly for querying but instead as a internal link in the query handler system both due to its complexity and lack of extensibility and support for new complex data types. Thus dashboards and graphical front-ends are the only tools used commonly for data warehouse development and multidimensional data analysis. CAL provides a new textual interface (for multidimensional data) that is similar in its ease-of-use to SQL (for relational datasets). In this section, we provide a comparison of syntax and semantics between CAL and MDX, highlight some of the shortcomings of MDX and motivate the need for a new abstraction over existing OLAP languages.

First, MDX requires building sets and tuples and the creation of explicit cross joins and unions between members. This is achieved using many different parentheses that often make the code unreadable. In CAL, the underlying $\mathcal{B}ig\mathcal{C}ube$ data model takes care of type checking and query validation. CAL does not provide for explicit user-level cross-joins. Instead the order of aggregation is according to the hierarchy explicitly specified by the user.

Further, MDX allows axes to be specified in order and does not allow alteration or deletion of a previous axis. This is often a big restriction when the DW structure is being altered by an aggregation query. In CAL, the query parser dynamically reorders query results with internal axis ids which are different from those specified by the user. This helps when multi-level CAL queries are built with several sub-cubes that used in higher level computations. The link between the internal physical axis id and the logical user specified axis number is maintained by the system. Thus, CAL does not restrict the number of axes or a skip in their specification in the query.

Third, from the user's standpoint one the main problems with MDX is the mixing of structure and the values of the cube. For e.g., Time represents a cube dimension type whereas 2011 represents the value for the Year level in Time. This separation is not clear in a MDX query. In CAL, the underlying $\mathcal{B}ig\mathcal{C}ube$ data model makes the clear separation using explicit data-typing, and the user is always allowed to start from the pre-defined cube *structure* and lead onto the *values* as seen in the example below.

```
SELECT { [Time].[2006] : [Time].[2011] } on columns,
       { [Product].[Tools] : [Product].[Home Audio] }
       on rows
FROM   Sales
WHERE ([Location].[Gainesville], [Measures].[Unit Sales])
```

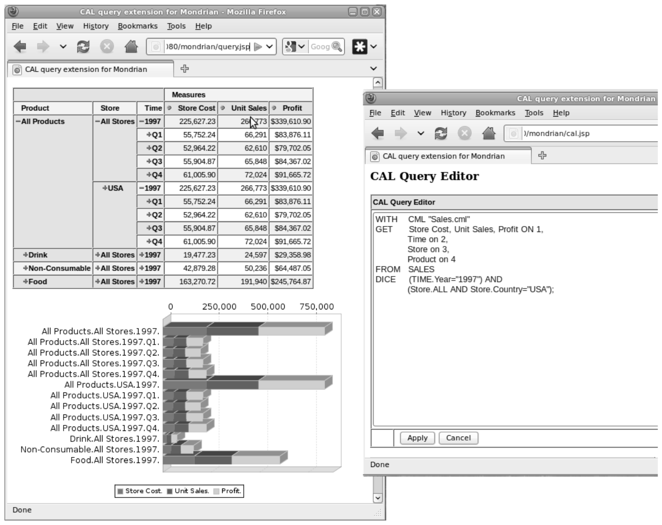
The equivalent CAL query shown below illustrates an example of how the syntax is simpler and more intuitive for analysis.

```
GET    Time.[2006:2011] ON 1,
       Product.[Tools:‘Home Audio’] ON 2
FROM   Sales
DICE   Location=‘Gainesville’ AND Sales.‘Unit Sales’;
```

Further, MDX only allows implicit slice (using WHERE clause) across dissimilar dimensions. However in CAL, DICE can be used for choosing members even from the same perspective. The system internally handles the two cuboids that result from “each slice” and merges them together into one single data cuboid. Also, CAL is not case or line-oriented, and spaces in member or measure names are handled using enclosing *backtick* characters as in SQL.

6 Logical Implementation

The Cube Analysis Language is based on the $\mathcal{B}ig\mathcal{C}ube$, which is a generic data model for


CAL query extension for Mondrian - Mozilla Firefox
File Edit View History Bookmarks Tools Help
)80/mondrian/query.jsp
CAL query extension for Mondrian
0 250,000 500,000 750,000
All Products.All Stores.1997.
All Products.All Stores.1997.Q1.
All Products.All Stores.1997.Q2.
All Products.All Stores.1997.Q3.
All Products.All Stores.1997.Q4.
All Products.USA.1997.
All Products.USA.1997.Q1.
All Products.USA.1997.Q2.
All Products.USA.1997.Q3.
All Products.USA.1997.Q4.
Drink.All Stores.1997.
Non-Consumable.All Stores.1997.
Food.All Stores.1997.
Store Cost. Unit Sales. Profit.
Done
CAL query extension for Mondrian
File Edit View History Bookmarks Tools Help
)/mondrian/cal.jsp
CAL query extension for Mondrian
Apply
Cancel
Done

			Measures		
Product	Store	Time	Store Cost	Unit Sales	Profit
−All Products	−All Stores	−1997	225,627.23	26[illegible]773	$339,610.90
		+Q1	55,752.24	66,291	$83,876.11
		+Q2	52,964.22	62,610	$79,702.05
		+Q3	55,904.87	65,848	$84,367.02
		+Q4	61,005.90	72,024	$91,665.72
	+USA	−1997	225,627.23	266,773	$339,610.90
		+Q1	55,752.24	66,291	$83,876.11
		+Q2	52,964.22	62,610	$79,702.05
		+Q3	55,904.87	65,848	$84,367.02
		+Q4	61,005.90	72,024	$91,665.72
+Drink	+All Stores	+1997	19,477.23	24,597	$29,358.98
+Non-Consumable	+All Stores	+1997	42,879.28	50,236	$64,487.05
+Food	+All Stores	+1997	163,270.72	191,940	$245,764.87

CAL Query Editor

CAL Query Editor

```
WITH    CML "Sales.cml"
GET     Store Cost, Unit Sales, Profit ON 1,
        Time on 2,
        Store on 3,
        Product on 4
FROM    SALES
DICE    (TIME.Year="1997") AND
        (Store.ALL AND Store.Country="USA");
```

Figure 1: CAL query editor and results of OLAP analysis using the Cube Markup Language (CML) and Cube Analysis Language (CAL) extensions on Mondrian[6].

multidimensional data cubes. Thus, CAL can work with both popular logical design approaches for data warehouses, namely, Relational OLAP (ROLAP) and Multidimensional OLAP (MOLAP). To demonstrate the functionality of this query language in user-centric multidimensional analysis, we have implemented a CAL parser for Mondrian [6], which is an Open Source OLAP server from Pentaho. We have provided XMLA datasource access from a relational PostgreSQL database and extended JPivot, a JSP tag library capable of rendering OLAP tables and charts (see Figure 1) in order to display the results in graphical format. Consider a query '*to display the store cost, unit sales and the profit generated from the sales of products, during 1997, in stores inside the USA and the gross total over all stores (all locations)*' from our sample dataset. Using CAL, this query can be built using a simple set of CQAL statements and its result can be graphically analyzed in Mondrian as shown in Figure 1.

7 Conclusions and Future Work

This paper presents a major improvement in user-centric data warehouse design by introducing a novel text-based *Cube Analysis Language* for multidimensional analysis. CAL consists of three components, namely the *Cube Definition Language* (*CDL*), the *Cube Manipulation Language* (*CML*) and the *Cube Query and Analysis Language* (*CQAL*) that help to build, manipulate and query the data cube respectively. CAL helps the user to communicate in an abstract datacube environment and provides an extensible interface to perform multidimensional navigation and data analysis operations. These DW operations are internally validated and implemented using the underlying logical data model (ROLAP or MOLAP). We also demonstrate the functionality of CAL by implementing a working prototype over Mondrian (an open source OLAP tool) and a relational Postgres database. In the future, we plan to provide support for spatial data and moving objects and their aggregations in the data warehouse using CAL, and also complete a MOLAP server capable of handling uncertainty and various ragged hierarchies during data analysis.

References

[1] A. Bauer and W. Lehner. The Cube-Query-Languages (CQL) for Multidimensional Statistical and Scientific Database Systems. In *Proceedings of the 5th Intl. Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 263–272. World Scientific Press, 1997.

[2] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query language for relational databases. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, Montreal, Canada, 1996.

[3] W. Lehner, T. Ruf, and M. Teschke. CROSS-DB: A Feature-Extended Multidimensional Data Model for Statistical and Scientific Databases. In *Proceedings of the 5th Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 253–260, New York, USA, 1996. ACM.

[4] Microsoft Corporation. Multidimensional Expressions (MDX) Reference, Accessed: 2nd March 2011. msdn.microsoft.com/en-us/library/ms145506.aspx.

[5] T. Pedersen, A. Shoshani, J. Gu, and C. Jensen. Extending OLAP querying to external object databases. In *Proceedings of the 9th Intl. Conf. on Information and knowledge management (CIKM)*, pages 405–413. ACM New York, USA, 2000.

[6] Pentaho Analysis Services: Mondrian Project, Accessed: 2nd March 2011. http://mondrian.pentaho.org/.

[7] G. Viswanathan and M. Schneider. BigCube: A Metamodel for Managing Multidimensional Data. In *Proceedings of the 19th International Conference on Software Engineering and Data Engineering (SEDE)*, pages 237–242, 2010.