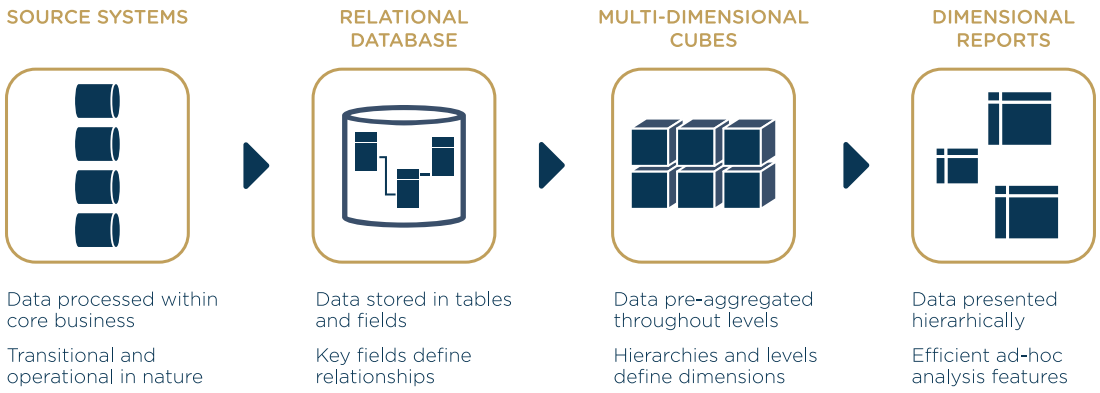
DATASOURCE CONSULTING
an EXL company

WHITE PAPER:

ENHANCING YOUR ENTERPRISE REPORTING ARSENAL WITH MDX

INTRODUCTION

In the trenches, we constantly look for techniques to provide more efficient and effective reporting and analysis. For those that utilize cubes, multidimensional databases or dimensionally modeled relational (DMR) models, multi-dimensional expressions (MDX) can be leveraged for not only solving complex report requirements, but also providing deeper insights to report consumers. In a typical enterprise-reporting environment, the reporting application issues queries to a relational database via Structured Query Language (SQL). While SQL is common and reliable, it is constrained to only two dimensions, rows and columns. A report requirement may demand a SQL query, or multiple queries and/or sub-queries, to read, filter, and group multiple tables. As complexity grows within the report requirement, the report developer is challenged to build an efficient query, or set of queries, in order to meet the report delivery service-level-agreement (SLA) and to minimize impact on the source system, such as memory usage, i/o and table availability. To help address these challenges, many Business Intelligence (BI) programs turn to dimensional models, cubed data sources, and multi-dimensional reporting. The graphic below illustrates a typical BI architecture that leverages multi-dimensional cubes and reports:



Enterprise reporting vendors, such as Cognos and SAS, have evolved their multi-dimensional reporting tools. Their applications, built for delivering Online Analytical Processing (OLAP) style analytics, now include functionality to simplify the customization of multi-dimensional reports. This means there are more ways to access, manipulate, and display data residing in a multi-dimensional source, such as a cube.

The presentation of an OLAP model has typically been constrained to a hierarchical view geared towards drill down analysis. By including multi-dimensional expressions (MDX) based functionality developers can break this traditional mold and present highly customized reports. Prior to the emergence of MDX, on-the-fly report customization by end users, was by and large delivered with relational reporting techniques. The availability of MDX illustrates how vendors are enhancing the use of their multi-dimensional reporting applications.

WHAT IS MDX

MDX surfaced in the late 90s as part of Microsoft's OLE DB for OLAP (ODBO) specification. Recognized as an industry standard for processing multi-dimensional data, ODBO is an application-programming interface (API) designed for exchanging data between an OLAP server and a Windows platform.

Microsoft also introduced XML for Analysis (XMLA) in 2000, which helped solidify MDX as a de facto standard for OLAP systems. Today, MDX based functionality is utilized by a wide range of enterprise reporting vendors, such as Business Objects (BO), Cognos, Microstrategy, Proclarity, and SAS, to facilitate and enhance their multi-dimensional analysis and reporting applications.

SQL vs. MDX

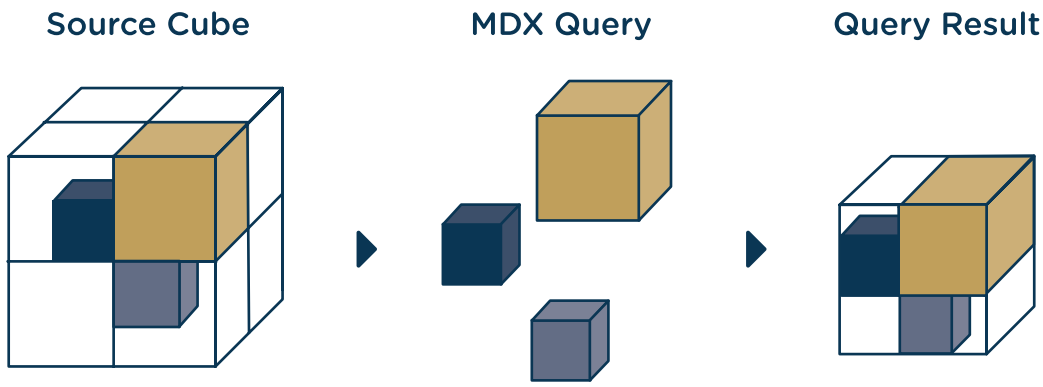
The primary difference between SQL and MDX is MDXs ability to query multiple dimensions whereas standard SQL can only query two dimensions. While SQL has the ability to query a multi-dimensional data source, it was designed to query a relational source. Semantically, the two syntax languages look very similar and are structured with comparable execution paths. The graphic on the following page compares the two query syntaxes at a high-level.

DATA SOURCES **EXACT QUERY** **QUERY TYPE**	 **RESULT SET** **SQL**	 **CELL SET** **MDX OR SQL**
APPLICATION PROGRAMMING INTERFACE	Object Linking and Embedding (OLE) Database (DB)	Object Linking and Embedding (OLE) Database (DB) for Online Analytical Processing (OLAP)
EXAMPLE QUERY	**SELECT** Sale_month, sum(sales) as 'sales' **FROM** Finance_FACT **WHERE** state = 'Colorado' **GROUP BY** sale_month	**SELECT** { [measures].[sales] } on COLUMNS { [dates].[JAN 2009] } on ROWS **FROM** [FinanceCube] **WHERE** [[state].[Colorado] }
SYNTAX DESCRIPTION **SELECT**	Defines the fields to return	Identifies the dimensions and members to return
FROM	Identifies the table and fields to query	Defines multi-dimensional source (cube) to query
WHERE	Identifies which dimensional fields to include or exclude	Returns a slice of the cubes data

Even though the two syntaxes share the keywords, SELECT, FROM, and WHERE, as noted in the above graphic, their meaning and usage differs. As SQL puts a resulting data set in columns of a query, MDX delivers dimensions as an axis within a resulting cubed data set. Microsoft defines an axis as an edge or dimension within the resulting cube. This differentiation, between originating dimensions and the resulting axes, simplifies the relationship between the source and the resulting data set. This parallels how SQL is used to query a relational source returning one or more tables within the same structure as the queried tables.

With regard to MDX, data resides in a multi-dimensional data source as value points within one or more cubes. A user query will extract additional cubes (sub-cubes of the source cube) comprised of a set of value points used to generate an additional result set cube, essentially creating a multi-dimensional shell, or subject-related skeleton of the query's result set, including queried measures where applicable. This means a placeholder will be created for each potential dimensional combination, that is uncompressed within the resulting cube.

The graphic below illustrates what happens when MDX is executed against a source cube:



MDX fundamentals

The MDX data model is based on dimensions, made up of individual members, and cubes as well as four of its most common components, tuples, sets, axes and slicers. These components form the syntactical building blocks of MDX and help illustrate its fundamental capabilities.

Tuples - Tuples and sets are similar to members and dimensions in that a group of tuples can form a set (whereas dimensions are comprised of a group of individual members). Tuples are essentially multi-dimensional members with only one member from each dimension. A simple tuple comes from a single member whereas a tuple combining a time related member and a geography-based member results in the following tuple:

([Time].[JAN 2009], [Geography].[Colorado])

Each cell member is identified as a tuple for each dimension in the cube. A tuple also represents a slice of the cube (whereas a simple tuple is the most granular slice).

Sets - Sets are a group of tuples within a pre-defined order. A set may contain the same tuple more than once, many tuples, or no tuples. Syntactically, there are many ways to represent a set, which in its most basic form is represented with curly brackets:

{ ([Time].[JAN 2009], [Geography].[Colorado]) }

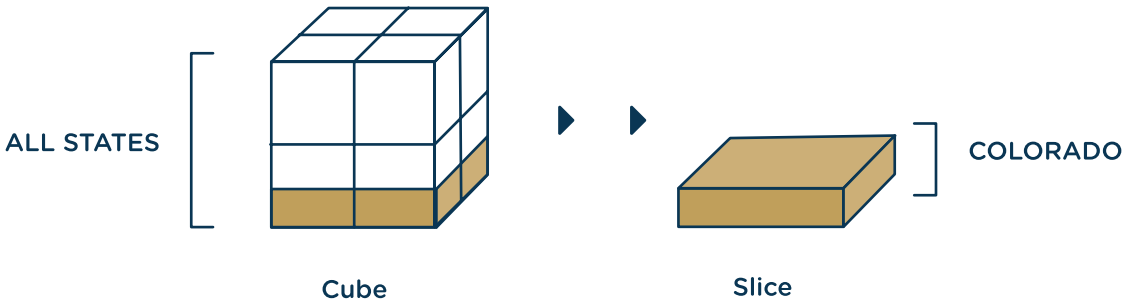
Sets are a significant building block within MDX as tuples are combined to customize multi-dimensional report data.

Axes - As mentioned, an axis is specific to the MDX query's resulting data set and refers to the dimensional structure requested within that query. The axis specification is defined right after the SELECT clause within the ON COLUMNS and ON ROWS statements below in bold:

SELECT
{ [measures].[sales] } on COLUMNS,
{ [date].[JAN 2009] } on ROWS
FROM [FinanceCube]
WHERE { [state].[Colorado] }

Slicers - The slicer specification, defined within the WHERE clause, is a way of filtering a query for a specific subset within the multi-dimensional source. The specification must include the WHERE keyword followed by a member, tuple, or set. The slicer is identified in bold:

SELECT
{ [measures].[sales] } on COLUMNS,
{ [date].[JAN 2009] } on ROWS
FROM [FinanceCube]
WHERE { [state].[Colorado] }



The fundamentals of an MDX query, as applied to enterprise reporting, represent how a developer can present an end-user with report customization options. The basic example above (for Sales specific to the month of 'JAN 2009' within 'Colorado') could be integrated into a dynamic enterprise report with prompts for 'Month' and 'State'. It is this type of fundamental technique, among other MDX based techniques for customizing report data, that enterprise report vendors are exposing more readily within their multi-dimensional reporting applications.

CONCLUSION

By equipping report developers with the techniques discussed above vendors are not only simplifying development of multi-dimensional reports, but also increasing the value of storing data in multi-dimensional sources. MDX and these techniques have been available but constrained to sophisticated developers, very fluent with MDX as a querying language as well as XML scripting and OLAP APIs.

By introducing these techniques into their multi-dimensional reporting applications, vendors are allowing novice developers to incorporate MDX based functionality into their reports. This enhanced functionality allows developers to satisfy a wider range of report requirements and utilize multi-dimensional sources for more than just ad hoc analysis. This enhanced functionality is blurring the lines between relational and multi-dimensional reporting. These techniques allow developers to embed relational reporting functionality into their multi-dimensional reports. This allows BI programs to use multidimensional sources, such as OLAP cubes, for more than just analysis.

By equipping developers with these techniques, vendors are increasing the usability of their tools and providing the industry with further justification for utilizing multi-dimensional sources.

References

Spofford, George (2001), MDX Solutions With Microsoft SQL Server Analysis Services Volitich, Dan (2008), IBM Cognos 8 Business Intelligence MSDN SQL Server Developer Center (2009), Comparison of SQL and MDX Retrieved November 24th, 2009, from:
http://msdn.microsoft.com/en-us/library/aa216779%28SQL.80%29.aspx SAS® 9.2

OLAP Server: MDX Guide (2009), Basic MDX and Cube Concepts
Retrieved November 24th, 2009 from: http://support.sas.com/documentation/cdl/en/mdxag/59575/HTML/default/a002253136.htm