

# MDX: The Language of Multidimensional Analysis

## How MDX Differs from SQL

In less than a decade, the programming language MDX has become the de facto standard for reporting and analysis in business intelligence (BI) systems. It is supported by virtually every major vendor of BI servers, including Microsoft, SSAS, IBM, Hyperion, and SAP.

Why has MDX become so popular – and so important to business intelligence – in such a relatively short time? And how does it compare to SQL, which has been around a much longer time?

MDX, which originated with Panorama Software, was developed to navigate, query, and perform calculations in OLAP databases. In contrast, SQL (Structured Query Language) was designed to create, manage, and query relational databases in online transaction (OLTP) systems.

The popularity of MDX, which stands for MultiDimensional eXpressions, has come about for two primary reasons.

The first is the growth of the BI industry itself, which encompasses data warehouses, OLAP databases, and reporting and analysis tools. (OLAP stands for online analytical processing, and OLAP databases are usually referred to as “cubes.”) The growth of BI has been fueled by the recognition that business users need fast and intuitive access to company information to make more profitable business decisions.

The basis of an OLAP cube is the multidimensional database model, which consists of measures and dimensions joined through a central fact table. Multidimensional databases are also called “star schemas” because dimensions are arrayed around a fact table in a star-like structure. They are designed to give users direct access to information that is understandable, quick to find, and easy to use. Multidimensional OLAP databases are fundamentally different from the relational databases used in transaction systems. Instead of the star schema, an OLTP (online transaction processing) database has a schema that looks more like an intricate spider web, with hundreds of different tables joined together through a complex set of primary and foreign key relationships. Designers of OLTP systems worry more about ensuring that database updates and inserts are lightning-fast – only hitting the database in one place (using a technique called normalization) – than in making them easy for reporting. Data in relational tables is often labeled using coding techniques that make the field (column) names short and unique but lack descriptions that could make the tables more understandable to business users.

On the other hand, designers of OLAP systems have these goals in mind:

- ▶ **Extract information of value to business users** from the source transaction systems.
- ▶ **Transform the information so that it is easily understood**, including creating new measures, adding intelligent descriptions to code, and standardizing information (for example, putting addresses into U.S. Postal Service format). To add context to the information, OLAP designers frequently will incorporate third-party data such as consumer demographics and U.S. Census Bureau data for consumers and businesses.
- ▶ **Load the data into a multidimensional database** so that users can view measures by a wide variety of dimension characteristics, such as sales by product, region, time, and sales channel.

Dimensions often have multi-level hierarchies, enabling drill down from summary data to more detailed information. Examples of multi-level dimension hierarchies include time, with drill down from year, quarter,

month and day; geography, with drill down from state, county, city and ZIP Code; and product, with drill down from category, subcategory and product.

This emphasis on hierarchies and drill down in leads to the second reason for MDX's popularity: It is uniquely able to navigate, query, and perform calculations against multidimensional structures. These include OLAP cubes, dimensions, hierarchies, levels, attributes, members and measures. While theoretically you can query an OLAP cube using SQL, SQL has no inherent knowledge of dimension hierarchies and cannot easily navigate to the parent or children of a particular dimension member.

MDX, however, recognizes all kinds of dimension relationships, from ancestors to descendants, from members to their siblings (in a hierarchy level), and from the dimension root (the All level) to the leaf level (most granular data). If you know a particular dimension member, then MDX lets you determine all these relationships with very little coding. For example, if you know a particular product, you can query for that product's subcategory or its category with a single MDX code (Parent or Ancestor).

MDX includes a variety of functions for doing time-period comparisons, such as ParallelPeriod for viewing results from a current period to the same period a year ago. PeriodsToDate is used to analyze data from a particular period in the past to the most current period. LastPeriods lets a user select a set of periods from a current period to a period in the past (for example, the last three, six or twelve months). Other MDX time functions include YTD, QTD, and MTD, which are shorthand for the PeriodsToDate function.

The equivalent queries in SQL would takes dozens or hundreds of single-spaced lines of code and could only be written by a SQL expert. MDX has nearly 200 functions to navigate, query, filter and perform calculations in OLAP cubes. As George Spofford writes in his book, MDX Solutions, things that are straightforward in SQL are also straightforward in MDX. However, things that are complex in SQL can often be very straightforward in MDX.

Let's dig more deeply into the differences between MDX and SQL.

## MDX Originated with Panorama Software

When first conceived of by Panorama Software in the mid-1990s, MDX was an extension of SQL. In fact, its original name was Multi-Dimensional eXtensions, says Mosha Pasumansky, a former Panorama developer who is credited with being the primary author of the language. While MDX and SQL have significant differences, they can be thought of as "kissing cousins." MDX actually started life as a series of macros on top of SQL in order to reduce the amount of SQL coding required to query Panorama's OLAP server. In a classic case of form follows function, Panorama first defined the functionality needed to query its OLAP server, then developed the MDX language to support that functionality.

MDX became Multi-Dimensional eXpressions after the language and the Panorama OLAP server were acquired by Microsoft in 1996. Microsoft expanded the language and released it in the 1997 OLE DB for OLAP specification, now an industry standard. The OLAP server became OLAP Services with the release of SQL Server 7.0, and later was renamed Analysis Services when data mining algorithms were added to the server in SQL Server 2000. MDX later became the basis for the industry standard XML/A (analysis) interface (for Web services), which made MDX a de facto standard.

After the OLAP server and MDX were acquired by Microsoft, Panorama continued development of its own MDX-based NovaView query tool. Panorama continues to work closely with Microsoft to make sure that

NovaView takes full advantage of on-going enhancements to Analysis Services. More recently Panorama forged a partnership with the SAP BW/NetWeaver BI development team to optimize MDX queries to the SAP data warehouse, while also helping SAP enhance its implementation of MDX.

NovaView generates MDX queries as part of the user interface for both Microsoft Analysis Services and SAP BW/NetWeaver BI, without casual users being aware that MDX exists. NovaView, however, makes the advanced functionality of MDX easily available to power users, analysts, and report authors.

## MDX and SQL Share Superficial Similarities

Given that MDX started out as SQL macros, it's not surprising to see similarities in the two languages. You can see this in the SELECT statements for both languages.

The basic MDX query has this form:

```
SELECT <axis specification> on columns, <axis specification> on rows FROM <cube>
```

The basic SQL query has nearly identical syntax:

```
SELECT <column 1>, <column 2>, <column n> FROM <table>
```

In SQL, the SELECT keyword selects columns from one or more tables, which are specified in the FROM clause and linked together using JOIN conditions. Multi-table joins can be quite complex. MDX requires no joins because dimensions and measures are linked through a cube's metadata.

In MDX, the SELECT keyword places sets of dimension members on one, two, three or more axes. Axis 0 can be referenced using the term "columns" and axis 1 can be referenced using the term "rows." This, however, is for convenience and an MDX query functions work just fine using either "0" or "columns." Technically an MDX query could have three, four, or more axes, but most query tools can represent ONLY a two-dimensional crosstab or grid. The FROM clause specifies a particular OLAP cube.

Both languages have an optional WHERE clause. In SQL, the WHERE clause is used to restrict (filter) the rows. In MDX, the WHERE clause points to a specific slice of an OLAP cube.

Most users will never have to write a full MDX query using SELECT, FROM and WHERE. That's because BI tools like NovaView generate the MDX queries in the background. MDX expressions in BI tools can be used to create calculated members, sets, filters, and exceptions.

MDX and SQL share similar functions for counting and calculations, such as AVG, COUNT, MIN, MAX, SUM, STDEV, and VAR. But, as described earlier, that's where the similarities end. You can see the differences in the following SQL and MDX queries. Both queries are written to calculate the percentage of total sales that each city in the database contributes to its parent state.

Here's the full SQL query:

```
SELECT c.state,c.city,sum(c.sales_revenue) sales_City
into #city
FROM sales_revenue_table c
group by c.state,c.city
order by c.state,c.city
```

```
SELECT s.state,sum(s.sales_revenue) as sales_state
into #state

FROM sales_revenue_table s
group by state

select c.state,c.city,c.sales_City as 'City Sales',
s.sales_State as 'State Sales', round(100* cast(c.sales_City as
float) / cast(s.sales_State as
float),2) as 'Percent State Total'
from #city c,
     #state s
where c.state = s.state
order by c.state, c.city
```

Here's the same query written as an MDX expression in NovaView:

```
[Measures].[Sales Revenue} / ([Measures].[Sales Revenue,
    Ancestor( [Geography].[City], [Geography].[State] ))
```

Without a detailed discussion of the code syntax, you can see that the SQL query is much more complex than the same query written in MDX. MDX is taking advantage of the relationship between cities and states in the geography dimension in a single expression. The MDX Ancestor function is asking for the ancestor of the city at the state level. The SQL query is using multiple SELECT statements, with GROUP BY and ORDER BY clauses, to perform the same calculation.

This MDX query is the type of expression that would be used in a BI client tool such as NovaView. NovaView would take the expression and automatically create the full MDX SELECT, FROM, WHERE statement. In fact, NovaView has a user interface function that would provide the identical results without having to write an MDX expression.

## MDX and SQL Query Different Data Sources

The primary difference between MDX and SQL is in the types of data sources they query.

MDX navigates multidimensional OLAP cubes. Theoretically, cubes can have dozens of dimensions, but rarely have more than 15 to 20 to make them easier to work with and more understandable. Cubes consist of dimensions, hierarchies and levels, attributes, and measures. The result of an MDX query against an OLAP cube is a subcube, a subset of the original cube.

SQL navigates across two-dimension tables (each with columns and rows) in OLTP relational databases. There may be hundreds of tables in an OLTP system, most of them irrelevant for analysis and reporting. Relational databases are designed for transaction speed, not for understandability. The SQL query writer has to know which tables contain the information he or she is interested in and how to join them.

Much about SQL is getting columns from multiple tables to line up in rows using JOIN conditions, which can be quite complex. The result of a SQL query is another table with columns and rows. Queries against OLTP systems for reporting can severely affect processing speed, and most IT organizations don't permit SQL queries during daily operations. Some organizations may replicate a relational database for reporting, but it will maintain all the querying difficulty of its operational twin.

Rows in SQL are defined by the columns, each of which can be of different data types and meanings. In MDX, axes are structured independently. The columns do not define the rows and vice versa. It's the intersection of the axes that provides the relevant data.

Another important distinction between MDX and SQL is in the use of the WHERE clause. In SQL, the WHERE clause filters the result set by variables from one of more tables, for example restricting results to a specific year or ZIP Code. The WHERE clause in MDX performs a similar filtering function, except the WHERE clause points to specific coordinates in cube space. All the cube dimensions that are not specified on the axes in an MDX query are parked in what are called slicer dimensions. An MDX query references every dimension in a cube, usually by the default member (which in most cases is the [All] level). However, a dimension can be sliced, or restricted, by one of its members. It's this slicing that's specified in an MDX WHERE clause.

## SQL Has No Knowledge about Dimension Hierarchies

SQL has little knowledge of position. In a SQL query, rows are independent of each other and it doesn't matter where they are located in a table. MDX, on the other hand, is very position-aware, knowing which dimension members are parents, ancestors, children, siblings, previous members, or descendants of each other.

You use these and other MDX functions to return sets of related dimension members, such as the descendants of a particular month, which are all the days of that month. MDX "knows" that September has 30 days, October has 31, and February has 28 (except in leap years and MDX also knows that!), so that the days of a month don't have to be hard-coded in a query.

SQL doesn't make it easy to compare data across time periods. In a typical OLTP system, the only reference to time you will see is the specific date of a transaction. In an OLAP cube you will see dates organized into hierarchies, with year drilling down to quarter, quarter to month and month to days (dates). It's very difficult to write a SQL query that will return sales for this month compared to the same month the previous year. MDX does this in a simple expression using the ParallelPeriod function. It's the relationships among OLAP cube objects that give MDX much of its power.

## SQL Does Things that MDX Can't

SQL is a multi-purpose language for constructing and querying relational tables. In addition to querying relational tables, SQL is a data definition language (DDL) for creating relational databases and tables, altering them, populating them, and updating them. In these respects, SQL is self-contained. A relational database administrator can do virtually everything required of her using SQL.

MDX has almost no data definition capabilities beyond creating calculated members and named sets. You can't create an OLAP cube using MDX. You can, however, create a multidimensional, relational

database using SQL. SQL is commonly used to extract, transform, and load data from online transaction processing systems into data warehouses.

SQL and MDX can be thought of as “kissing cousins.” SQL is used to query relational (twodimensional) databases, while MDX targets multidimensional OLAP cubes. The two languages share the same DNA: MDX was originally created as extensions of SQL. Today MDX has been redesigned as an object-oriented language that works against the Microsoft ADOMD.Net object model, or is mapped to other multidimensional structures in OLAP servers such as SAP NetWeaver BI.

At the end of the day, it’s not an either MDX or SQL question. MDX and SQL work hand-in-hand in BI applications, each doing its part to support self-service reporting and analytics. Organizations that master both languages will have a competitive advantage over their less knowledgeable competitors.

For more information about Panorama NovaView, please contact us at [info@panorama.com](mailto:info@panorama.com) or call:  
US/Canada: +1-877-709-5848 | Europe: +44-207-887-6300 | Israel: +972-3-645-9777  
[www.panorama.com](http://www.panorama.com)

