Multidimensional Query Languages

See [msd, Mica, Hyp04, Hyp00]

Overview

- multidimensional query languages
- Microsoft's MDX as a case study

Multidimensional Query Languages

In the relational world, we've grown to have dominant query language (SQL).

Not an overnight standard. Plus companies play with standard.

There's no such dominant language yet for multidimensional queries.

Lots of academic proposals and proprietary "report generator" languages.

Enter MDX.

Isn’t SQL enough?

SQL has been extended in many ways, including adding support for temporal queries, adding CUBE and ROLLUP etc.

Do we really need anything else?

The Hyperion people want you to believe so. Microsoft too. See [Hyp00, Micb].

Disadvantages of SQL [Hyp00]

- many basic OLAP computations are awkward
- others are impossible (example involves “TimePeriodSales / Qtr-Sales”)
- Being optimized for non-OLAP, multidim queries will probably be too slow.
- summary tables are too painful to manage

And Microsoft says. . . [Micb]

". . . with effort, you can even duplicate some of the functionality provided by MDX in SQL".

But most of their focus is on "you should be thinking multidimensionally and SQL encourages you to think of rows and columns".

You'd need someone who was both an SQL expert and an MDX expert to really draw a believable conclusion.

Who really uses query languages?

The dream of “ordinary end-user computing” is hard to realize.

So, while SQL and other query languages were supposedly for end-users, well. . .

However, to turn a (technically minded) end-user into a query-issuing demon is much easier with a good query language. . . even if his/her business friends think he/she has joined the pencil-heads [us].

A little self-study or a night course is the barrier. . . not the need to earn an MCS.

The MDX query language

Microsoft's query language for OLE DB, late 1990s.

Hyperion has implemented it too. Probably other vendors.

"XML for Analysis" is a multi-vendor (stalled?) standard using MDX.

Usual recommended book is by Spoffard, but I haven't seen it. Web has lots of examples and tips, but it's tough to find a semantic description.

Some MDX sources

So far, the clearest explanation I've seen is in a Hyperion white paper [Hyp04]. (Plus, you get some idea of a competing system called "Report Writer" as a bonus!)

At the Microsoft Developers Network (MSDN) website, the library contains some reference pages. *Don't try to learn from there!*

MDX examples

Some are based on the Foodmart demo cubes (used by Microsoft and also Mondrian). (Web sites checked in Feb 2006.)

- from Alessandro Vernet [Ver03]
- from Russ Whitney at www.winnetmag.com [Whi01]. (Now requires registration and/or fee?)
- tests built into Mondrian demo
- from Bill Pearson's articles/tutorials at DatabaseJournal.com [Pea02] (still growing: 40 articles as of Feb. 2006)

Recent changes to MDX

Microsoft's SQL Server 2005 introduces "MDX Scripts".

At least superficially, appears quite different from the MDX that has been part of SQL Server 2000.

I have not looked into it more.

An attempt to explain MDX

I've stared at and written quite a few examples.

So, without having read a really good explanation, I think I have a true explanation (overall). There may be serious errors, and maybe some Mondrian quirks.

Mondrian comments based on v1.0, from 2004.

Currently, Mondrian 2.0 is installed.

Overview of MDX

Query first sets up a grid of display cells

Then, the grid is filled in, by formula

Note: MDX seems to be case insensitive.

Display area

The display is inherently 2d, I think. But MDX pretends it has more.

Display axes: COLUMNS and ROWS are self-explanatory.

But there are also PAGES, SECTIONS,CHAPTERS and also AXIS(i) where $0 \leq i \leq n$.

Hyperion: $n = 63$. Microsoft $n = 127$. Mondrian: COLUMNS and ROWS only.

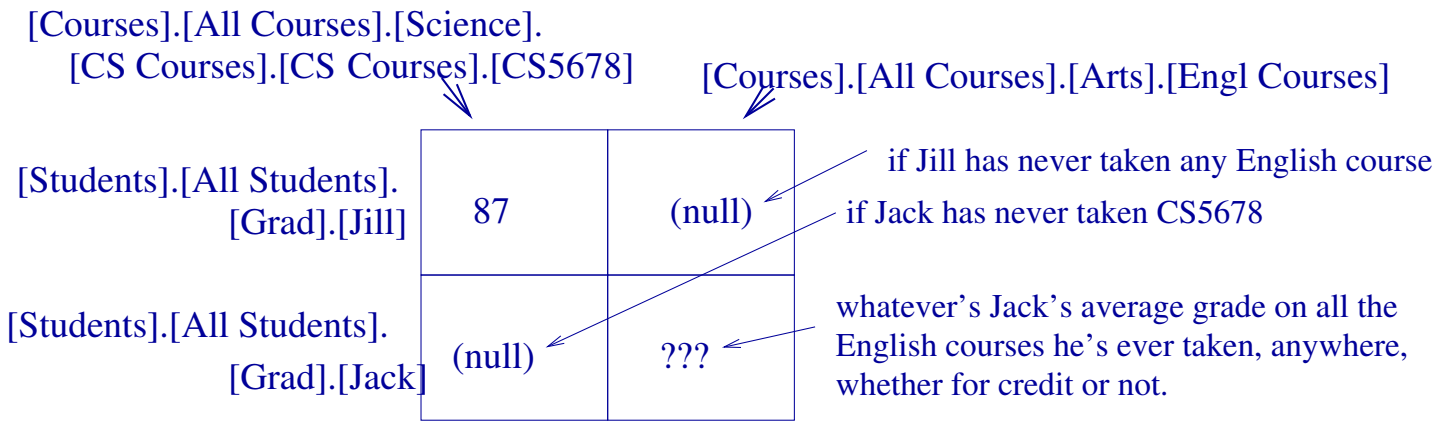
You cannot “skip” an axis, ie, specify ROWS and PAGES but omit COLUMNS.

First MDX query (For registrar's 6-d cube)

Registrar cube:

$$\text{Session} \times \text{Student} \times \text{Course} \times \text{Section} \times \text{Campus} \times \text{ForCredit} \rightarrow \text{avg}(\text{Grade})$$

```
SELECT {[Course].[All Courses].[Science].[CS Courses].[CS5678],
        [Course].[All Courses].[Arts].[Engl Courses]}  ON COLUMNS,
       {[Student].[All Students].[grad].[Jill],
        [Student].[All Students].[grad].[Jack]} ON ROWS
```



Values in the grid: How'd they get there?

Each grid cell corresponds to a context and some appropriate formula is evaluated in that context.

Eg, cell at intersection of

[Course].[All Courses].[Science].[CS Courses].[CS5678] and

[Student].[All Students].[grad].[Jill]

has the 4-d slice (Jill,CS5678) as its context.

Since we haven't specified any other formula, our main measure (avg(Grade)) is displayed. It's computed by averaging all numbers in this 4-d slice.

Syntax

A dimension value can be given by the complete path to it, in its hierarchy.

In Mondrian, you can sometimes leave out the [All *xyz*s] member.

But it would not like [Students].[Jill]. Dunno if it should.

Note the use of [and] around identifiers.

This is only required if the identifier has some “funny characters” like spaces in it.

Second MDX query (For registrar's 6-d cube)

Registrar cube:

$$\text{Session} \times \text{Student} \times \text{Course} \times \text{Section} \times \text{Campus} \times \text{ForCredit} \rightarrow \text{avg}(\text{Grade})$$

```
SELECT
 [Course].[All Courses].[Science].[CS Courses].children ON COLUMNS,
 {[Student].[All Students].[grad].[Jill],
  [Student].[All Students].[grad].[Jack]} ON ROWS
```

For columns: Note the lack of { and }, which had been used to form a set. The children function automatically returns a set.

Now, we have a column for every different CS Course.

Getting Fancier

Since a set-of-set is “flattened”, you can easily make a set union...

```
SELECT
 {[Course].[All Courses].[Science].[CS Courses].children,
  [Course].[All Courses].[Arts].children} ON COLUMNS,
 {[Student].[All Students].[grad].[Jill],
  [Student].[All Students].[grad].[Jack]} ON ROWS
```

Now I’ve got all CS Courses AND all Arts courses on (many) columns.

Axis dimensions

The dimensions assigned to the COLUMNS or ROWS are called axis dimensions.

You can assign several dimensions to an axis

Packing two dimensions onto the COLUMNS

```
SELECT { ([Course].[All Courses].[Science].[CS Courses].[CS5678],
          [For Credit].[All For Credits].[true]),
         ([Course].[All Courses].[Arts].[Engl Courses],
          [For Credit][All For Credits].[true])}  ON COLUMNS,
       {[Student].[All Students].[grad].[Jill],
        [Student].[All Students].[grad].[Jack]} ON ROWS
```

Now `For Credit`, `Course` and `Student` are all axis dimensions.

([Course].[All Courses].[Science].[CS Courses].[CS5678], [For Credit].[All For Credits].[true]) is called a tuple.

The context of the top-right cell in the grid is now (CS5678,Jill,true).

What will the resultant grid look like?

	[Course]..[CS5678]	[Course]..[Engl Courses]
	[For Credit]..[true]	[For Credit..[false]
...Jill	????	?????
...Jack	????	?????

Sets: How homogeneous?

Microsoft's documentation says that all tuples in a set must have the same dimensionality

First question: can you have tuples mixed with singletons in a set?

```
{ ([Course].[All Courses].[Science].[CS Courses].[CS5678],
   [For Credit].[All For Credits].[true]
  ), [Course].[All Courses].[Arts]
}
```

2004: Mondrian crashes, array-out-of bounds. Should it?

Sets: How homogeneous? (2)

Next question: can you have mixed dimensions (singletons) assigned to an axis?

```
{ [For Credit].[All For Credits].[true],
  [Campus].[All Campus],[Course].[All Courses].[Arts]
}
```

Mondrian allows it. The grid *looks* nice, but it seems that incorrect numbers are filled in the grid . . .

Specifying a range as a set

The values in each dimension have some natural ordering.

The binary : operator can build a range

Example

{ [Student].[All Students].[grad].[Jack]

:

[Student].[All Students].[grad].[Jill] }

Building a set of tuples with CrossJoin

The CrossJoin function is a handy way to combine two dimensions.

Sometimes can use infix *.

Usual use: pack several dimensions onto one display axis

Example (builds a set of $2 \times 2 = 4$ tuples for the x axis):

```
SELECT CrossJoin( {[Students].[grad],[Students].[ug].[Al]},
                  {[Campus].[SJ], [Campus].[Fton]} ) ON COLUMNS,
...
```

Functions for navigating dimensions: Members

(Best reference for the MDX functions is at MSDN.)

- members returns all the values of a given dimension. . . at all granularity levels.

  [Students].members would contain Jill, Jack, Al, grad, ug, [All Students].

Functions for navigating dimensions: Levels

- levels(*n*).members returns all values found at level *n* in a given dimension.

  Eg, [Student].levels(2).members = { [Students].[grad], [Students].[ug] }

Functions for navigating dimensions: Ancestor

- ancestor(*item, Level name*) returns the roll-up of the given dimension value to the appropriate level.

  Eg, ancestor([Student].[grad].[Jill], [Students].[kind]) returns [Students].[grad]

  (Obviously, silly to do when the item is a constant.)

  MSDN says that a level-number can be 2nd parameter, but it does not work in Mondrian.

Functions for navigating dimensions: Children

- children returns the children of a given member.

  [Student].[All Students].children is { grad, ug }

Functions for navigating dimensions: Descendants

- descendants(*Item, Level info*) can go arbitrarily far down (from current place) in the hierarchy.

  Descendants([Course].[Science], 2) skips the kids (Discipline level), but gives the grandchildren. eg EE2222, CS5678, . . .

  Descendants(X, 1) is the same as X.children

  Level info can also be a hierarchy level name, eg [Course].[Discipline].

- descendants(*Item*) gives the descendants at all levels.

Example query

```
SELECT CrossJoin( Descendants( [Students].[grad] ),
                  Campus.members
                ) ON COLUMNS,
       { [Courses].[CS5678], Courses.level(2).members} ON ROWS
FROM junk
```

- Columns for (Jill,SJ), (Jill, Fton), . . . (grad,[All Campuss])
- Rows for CS5678, Arts, Science, Engg, . . .
- One grid cell is “Jill’s avg(grade) in Science courses at SJ”

Slicer Dimensions

Recall, any cell in the grid corresponds to a slice of the cube. And the value for that cell is computed (somehow) based on the measures within that slice.

We can dice away (and not have to display) parts of the cube we don't want, using a slicer that specifies the context that we want.

MDX keyword is WHERE and should be followed by a single tuple.

Example: Only consider For-Credit courses in Fton

```
SELECT Descendants( [Students].[grad] ) ON COLUMNS,
       Courses.level(2).members ON ROWS
FROM junk
WHERE ([For Credit].[true],[Campus].[Fton])
```

Now the 2-d slice for each cell in the grid has specified [For Credit], [Campus], [Student] and [Course] values.

(Slice still has unconstrained [Session] and [Section]).

The “Only Use a Dimension Once” rule

When you assign a dimension to ROWS or COLUMNS, you use it.

When you constrain a dimension in a slicer tuple, you use it.

Logically, it does not make sense to use a dimension more than once. So don’t!

Multiple Measures

MDX permits several measures per cube. Syntactically, they are organized into a [Measures] dimension.

So, which measure gets used by default when the grid values gets filled in?

I'm hazy on this. I don't think the different measures should get aggregated. I think that the cube schema provides a default.

A common thing: use WHERE to slice just the desired measure. Eg,

SELECT . . .

WHERE ([Measures].[grade])

Assigning a measure to a display axis

If you assign members from [Measures] to a display axis (eg, *x*), the formula used to fill in the grid values in these columns reflects the specified measure.

Eg, suppose we have another measure counting F's

```
SELECT {[Measures].[grade], [Measures].[Fcount]} ON COLUMNS,
       Courses.level(2).members ON ROWS
FROM junk
WHERE ([For Credit].[true],[Campus].[Fton])
```

The two columns will contain different values.

Calculating sets

So far, meaningful calculations seem to occur only after the grid is laid out.

Now, we'll let you calculate how the grid is laid out, based on slice data.

Important: grid axes are calculated independently of one another, but using the slicer tuple.

Calculating sets: “non empty”

```
SELECT {[Measures].[grade], [Measures].[Fcount]} ON COLUMNS,
       non empty [Course].members ON ROWS
FROM junk
WHERE ([For Credit].[true],[Campus].[Fton])
```

This query omits rows for courses that have never been offered for credit in Fredericton.

Any omitted course has a slice

([For Credit].[true], [Campus].[Fton], that course) that is empty of measure values.

This ROW determination ignores the specified COLUMNs.

Iterative functions: Set in, set out

Functions generate, filter, topcount, order, . . . take a set and return a set.

Functional Programming Concept	related MDX
filter	filter
map	generate
take *n*	head *n*
(take *n*).reverse	tail *n*
(take *n*).sort	topcount *n*
sort	order

Iterative functions: Set in, value out

Functions sum, avg, min, max, . . . collapse a set to a single value (in an obvious way). Mostly used with “calculated members”, to be discussed later.

Functional programmers can think “fold”.

Iterative functions: Filter

Syntax: filter(*Set, Predicate*)

Iterates through the elements in *Set.*

Predicate is evaluated in the context established by the slicer and the current member of *Set*.

Often use *Dim*.currentmember

(There is also a prevmember for tricky code.)

Filter example: ignore credit courses nobody's failed in Fton.

```
SELECT [Student].members ON COLUMNS,
       filter([Course].members,
       not isempty([Measures].[Fcount])) ON ROWS
FROM junk
WHERE ([For Credit].[true],[Campus].[Fton])
```

This is almost like "not null". Context of evaluation is (Fcount,[For Credit].true, Fton, *the course*).

Filter example: ignore credit courses only a few failed in Fton.

```
SELECT [Student].members ON COLUMNS,
       filter([Course].members, ([Measures].[Fcount]) > 10) ON ROWS
FROM junk
WHERE ([For Credit].[true],[Campus].[Fton])
```

Context weirdness

filter([Course].members, ([Measures].[Fcount])>10) is equivalent to

filter([Course].members, ([Measures].[Fcount],[Course].currentmember)>10)

Then what does

filter([Course].members, ([Measures].[Fcount],[Course].prevmember)>10) do?

Uses the natural ordering on Courses. . . keep a course only if the course numbered before it had high failures.

Since Course ordering is meaningless, this is nonsense..

Mondrian has a bug and/or I misunderstand MDX

Mondrian accepts such queries but gives me surprising results.

First issue: what about members that are *first* and so have no predecessor? Mondrian does not seem to have the “isValid()” function.

An invalid slice should have no measures, so I tried the equivalent of

filter([Course].members, not isEmpty([Measures].[Fount],[Course].prevmember) and ([Measures].[Fcount],[Course].prevmember)>10)

And the selected values changed! (Still same number of them, but different).

Use of prevmember

If the dimension has a sensible ordering, (Eg, Session), then you might be able to compare equivalent periods.

If there are 3 Sessions per year, then

prevmember.prevmember.prevmember

could help you compare one session against the same session the year before.

currentmember.lag(3) is equivalent.

eg see courses whose failures have increased lots

filter([Course].members, ([Measures].[Fcount]) -

([Measures].[Fcount],[Session].currentmember.lag(3))

Sorting dimension values

Use order to sort things.

eg, sort courses according to their failures last term.

order([Courses].members, ([Measures].[Fcount],[Session].prevmember), BDESC)

“BDESC” is specifies some kind of descending sort.

order with just 2 parameters defaults to ascending sort.

Sorting dimension values non-numerically

What if I just want to put courses in alphabetic order? Ensure the 2nd parameter has a string (not numeric) type.

MDX allows “Properties” to be attached to dimension values, and they can be strings. Eg, registrar has a property “email address” for each student (but not for aggregates).

```
order([Student].[student].members, [Student].currentmember.properties("email address"))
```

orders people alphabetically according to email address.

Iterative functions: topcount

topcount is just a shorthand for combining “order” and “head”.

topcount([Student].[kind].members, 2, [Student].currentmember)

is, I think, equivalent to

head(order([Student].[kind].members,[Student].currentmember), 2)

The Hyperion white paper suggests the 3rd parameter to topcount is optional, but Mondrian wants it.

Iterative functions: generate

Generate lets you apply some operation to each member of a set.

Eg, for every student, form tuples of the student and his worst three sessions.

```
generate([Student].members,
  CrossJoin( [Student].currentmember,
             topCount([Sessions].level(2).members, 3,
                      ([Measures].[Fcount])
                     )
           )
        )
```

Example as part of a query

```
SELECT {[Measures].[grade]} ON COLUMNS,
...goop above.. ON ROWS
FROM junk
WHERE ([For Credit].[true],[Campus].[Fton])
```

Note that the slicer makes each student's worst semesters be determined only from Fton failures (that were for credit).

Named sets and calculated members

The WITH keyword is used to create named sets and new dimension values.

These items precede the SELECT keyword.

Example of syntax to create a named set

WITH SET *foo* AS ' *some set-valued expression*' SELECT . . .

The quotations are necessary. The SELECT part can now refer to *foo*.

Values in the grid

OK, so we've set up a grid. How about the values in it?

Recall, they are computed, by formula, from the cell's *context* established by the axis dimension and the sliced dimension.

The default formula is to aggregate the main measure, I think.

But you can change this. We've already seen you can specify an alternative measure member for a row or a column.

Now, we'll learn how to make up our own measures, and more.

Calculated members

Use WITH MEMBER [*Dim*].*name* AS '*formula*' to create a calculated member. Usually, but not always, a member of [Measures].

Any cell/tuple whose context involves a calculated member, uses that *formula* to determine its value.

And here's the problem. There may be more than one calculated member in a context....which *formula* do we use??

Solution involves SOLVE_ORDER and is deferred

Example

```
WITH MEMBER [Measures].[grade change] as
  '([Measures].grade,Session.currentmember) -
   ([Measures].grade,Session.prevmember)'
SELECT CrossJoin( {[Measures.[grade change]},
                  [Session].members
                ) ON COLUMNS,
       [Student].members ON ROWS
```

For every real and aggregate student, we compute the grade change from last session.

Hmmm. . . what about the very first session? No previous!

Example fixup

Fixup probably based on EXCEPT operator and also [Session].firstmember

Okay, now to tackle SOLVE_ORDER, which seems quite hairy and I'm not sure I understand all ramifications.

MSDN's longest MDX discussion is with this topic.

Silly Example, motivating SOLVE_ORDER

```
WITH MEMBER [Student].[M1] AS '1+2'
     MEMBER [Course].[M2] AS '3+4'
SELECT { [Student].[M1]} ON COLUMNS,
       { [Course].[M2] } ON ROWS
```

Well, in the 1×1 grid, do we see 3 or 7?

Solve order

When a calculated member is created, can specify SOLVE_ORDER = n to go with it.

$n \in [-8181, 65535]$ but is normally small & positive.

Of multiple competing values, the one with the largest solve order has the lasting effect.

It seems that the other formulas are evaluated earlier, and that their earlier values can, before being overwritten with the final version, be used in computing other values. All very subtle.

Illustration of subtlety

(This may well be wrong.)

Cell A is computed by a formula with SOLVE_ORDER 2 and another with SOLVE_ORDER 4.

Cell B is computed by a formula with SOLVE_ORDER 3 and the formula will use the value in cell A.

Therefore, the value used by B is not the final version of cell A.

See the MSDN references or the Hyperion white paper. This problem cannot arise if you have only one calculated member.

Examples

The references for this lecture have many different different examples.

Many are based on the FoodMart example. Fortunately, Mondrian has the FoodMart cubes for your playing. And Mondrian has about 19 FoodMart queries all ready for play. [Some don't work, on purpose, and others don't work, by accident].

To play with these cubes, you need to know the DW Schema.

In Mondrian, this is expressed in more-or-less self-evident XML.

FoodMart schema, some dimensions and (made up) example values

Definitions shared between cubes are specified first, in the XML.

- Store eg [Store].[All Stores].[USA].[CA].[Alameida].[Store 13]
- Time eg [Time].[1997].[Q1].[Jan]
- Product eg [Product].[All Products].[Drink].[Alcoholic Beverages].[Beer and Wine].[Beer].[Good].[Good Imported Beer]
- Warehouse eg [Warehouse].[Canada].[BC].[Victoria].[Can02]

Cube “Sales”

Dimensions are Store, Store Size in SQFT, Store Type, Time, Product, Promotion Media, Promotions, Customers, Education Level, Gender, Marital Status, Yearly Income, Has bought dairy.

Measures are Unit Sales, Store Cost, Store Sales, Sales Count, Customer Count.

Some of the other dimensions are

- Customers eg, [Customers].[All Customers].[Canada].[BC].[Burnaby].[Alexandra Wellington]
- Gender eg, [Gender].[M]
- Promotions eg, [Promotions].[All Promotions].[Bag Stuffers]

Aside: how did I discover some members?

Some are fictional. For others, I asked with an MDX query, eg

```
select head( [Promotions].levels(2).members,1) on ROWS
from Sales
```

Cube “Warehouse”

Dimensions: Store, Store Size in SQFT, Store Type, Time, Product, Warehouse

Measures: Store Invoice, Supply Time, Warehouse Cost, Warehouse Sales, Units Shipped, Units Ordered, Warehouse Profit

Cube “Store”

Dimensions: Store, Store Type, Has coffee bar, Grocery sqft

Measures: Store Sqft, Grocery Sqft

My advice is to play with this cube, because it is much smaller than many of the others.

Cube “HR”: bad to play with

Dimensions: Time, Store, Pay Type, Store Type, Position, Department, Employees with Time being not the global one.

Measures: Org Salary, Count, Number of Employees, Employee Salary, Avg Salary

Some sample values for the dimensions are:

[Time].[1997].[Q1].[1], [Pay Type].[All Pay Types].[hourly], [Position].[All Position].[Middle Management].[HQ Information Systems],[Department].[All Departments].[1]

Cube “Warehouse and Sales”

Has most things in Warehouse and in Sales.

Mondrian 1.0 notes (dunno about 2.0)

Several of the sample queries in Mondrian don't work because they try to use a `val` function (converts strings to numbers) on things that are already numbers.

Other sample queries don't run in Mondrian because the `item` function does not seem to work.

The [Measures] dimension reacts badly when you ask for its members.

Mondrian notes (2)

When you have a query syntax error or Mondrian has an internal error, it reacts very rudely, with a nasty Java traceback. By scrolling back in the traceback, I've found one *can indeed* (usually) guess what has gone wrong.

The foodmart example is quite realistic [seems big to me]. Several of the queries keep the machine humming for minutes. I am concerned about overloading my database machine.

Please avoid running expensive queries when you expect several users (ie, around the time that assignments are due).

Running Mondrian

Mondrian is installed as a Web service on pizza.unbsj.ca.

Just hit `http://pizza.unbsj.ca:8080/mondrian` with a browser!

Use the Ad-hoc query interface. You can start with some 19 queries; choose from the drop-down menu then click “Show query”. Edit the query to your taste, in the textbox, then click “Submit MDX query”.

Hint: use a text editor with your query. Then just copy and paste to the Mondrian text box.

Running Mondrian with XML/A

Use the “XML for Analysis Tester”. For queries, choose built-in for a SELECT query. Between the tags for XML element STATEMENT, you’ll see the text of an MDX query. Replace it with your own query.

Or write your own SOAP client.

Other Mondrian demos

The custom tags look really promising. . .

Three Cheers for Julian Hyde et al.

I may’ve mentioned lots about bugs, but an open-source OLAP system is wonderful for a course like this, or for an “implementing OLAP” course.

References

[Hyp00] Hyperion. Analytical processing: A comparison of multidimensional and SQL-based approaches. online as a white paper, http://www.hyperion.com, 2000.

[Hyp04] Hyperion. Introducing MDX to Report Writer users. online as a white paper, http://www.hyperion.com, 2004.

[Mica] Microsoft. Analysis services: MDX. online at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/olapdmad/agmdxbasics_04qg.asp.

[Micb] Microsoft. MDX overview: Comparision of SQL and MDX.

[msd] Microsoft OLE DB for OLAP.

[Pea02] Bill Pearson. MDX essentials (series). online at http://www.databasejournal.com, 2002.

[Ver03] Alessandro Vernet. OLAP: MDX examples. online at http://www.scdi.org/∼avernet/misc/olap-mdx, 2003.

[Whi01] Russ Whitney. MDX by example. online at http://www.winnetmag.com/Articles/Print.cfm?ArticleID=22994, 2001.