本文来自CSDN博客，转载请标明出处：<http://blog.csdn.net/zhangjunfangkaixin/archive/2009/09/17/4563739.aspx>

http://bidwhome.itpub.net/category/20871/35504

"?"

**这是我在学习Essbase过程中记录的笔记，全是我个人的理解，有理解不正确的地方，希望知情者指正，谢谢.**

在Essbase中使用MDX只能用于查询，尚不能来操作Cube（MS SSAS中可以使用MDX来操作Cube的），所以这里只学习MDX的Select语句。

**在学习的过程中，我觉得最有难度的地方有两个。**

一是，对OLAP多维的理解。如同学习SQL一样，SQL作用的对象是表，表的结构都是二维的，标识行列即可；但是MDX作用的对象是多维数据库，那么就得理解多维数据库的维度，层级，成员，以及如何在多维数据库中定位不同层级的成员。

二是，理解MDX查询语句的结构及术语的含义。学习SQL时，我们常提到表，字段，那么MDX中呢，想必这些术语必须理解，例如：Cube，Slice，Axis，Set，Tuple，Members

我的学习经历是这样的，首先了解MDX语句的特点及功用。

“MDX全称是Mutil-Dimensional Expressions，即多维表达式。MDX是一种和SQL类似的语言，它也可以用于查询、计算和定义一些元数据。只不过SQL是基于OLTP的，而MDX是基于OLAP的，也就是说，MDX是对多维数据进行查询的。和其它的OLAP语言不同，MDX并不完全是一种报告形式语言。但用MDX查询出来的结果仍然可以在客户端程序中以表格形式进行展现。MDX和SQL一样，也有很多不同功能的API来支持MDX，最常用到的就是XML API。”

**上面的一段话是从网上摘录来的，给MDX做了一个定义。**

**接下来学习MDX查询语句的结构及组成。**

[<with\_section>]

SELECT [<axis\_specification>

[, <axis\_specification>...]]

[FROM [<cube\_specification>]]

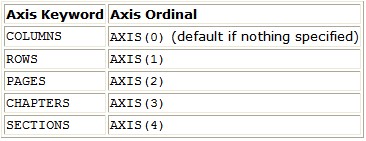
[WHERE [<slicer\_specification>]]

这个结构就是MDX查询语句的通用结构，从中要理解哪些是关键字以及关键字的顺序，WITH section、Axis、Cube、Slicer的位置。这些都是MDX的语法规定，所以要牢记各自的位置。

然后学习的是以上MDX语句结构中各个位置的含义。

<with\_section>的位置用于设定自定义的计算。有两种运用，WITH SET…,WITH MEMBER…，在复杂的MDX查询语句中经常被用到。

<axis\_specification>的位置用于设定轴。这个轴是反映在MDX语句的输出报表上的。常用的就是列、行、页，如下图所示：



<cube\_specification>位置用于指定查询用的Cube，只能指定一个。

<slicer\_specification>位置用于指定限制条件，可以是多个限制条件，反映在Cube中称为切片。

**到了这一步要学习就是MDX查询语句中每个轴上的内容了。如：Set、Tuple，Member等**。

Set称为“集合”，类似数学中集合的定义，要求集合内的成员结构必须相同；

Tuple称为“元组”，类似Member，但是特殊的地方是一个Tuple是由一个或多个成员组成，而这些成员来源于不同的维度，每个维度只能出现一次；

Member称为“成员”，就是Cube结构上的最小单位。

注意：Tuple的概念比Set小，一个Set中可以包含多个Tuple。

Tuple使用( )来标记；Set使用{ }来标记；

如果一个Set中有多个Tuple，那么这多个Tuple的组成维度必须相同，且顺序相同。

例如下面的例子就是错误的。

错误01：维度不同

**{ ( [Time].[2005], [Measures].[Dollar Sales] ),**

**( [Time].[Feb, 2005],[Product].[Toys] )**

**}**

错误02：维度相同，但是顺序变了。

**{ ( [Time].[2005], [Measures].[Dollar Sales] ),**

**( [Measures].[Unit Sales], [Time].[Feb, 2005] )**

**}**

错误03：Tuple中出现了同一维度的两个成员

**( [Measures].[Unit Sales], [Measures].[Dollar Sales] )**

学习到这里，我就可以写一些简单的MDX查询语句了。

例如：

SELECT

**{[100-10], [100-20]}** ON COLUMNS,

**{[Qtr1], [Qtr2], [Qtr3], [Qtr4]}** ON ROWS

FROM Sample.Basic

**仅仅是简单的MDX查询语句是不够，要写出复杂的查询语句，必须学习MDX的一些函数。**

我个人觉得学习这些函数，首先要了解这个函数的返回值是什么，可能是数值，布尔值，集合，元组，成员等；然后再去学习函数的输入参数是什么。完成这两点之后，学习使用MDX函数就会变得很简单。

经常用到的CrossJoin ,Filter, Order等。

学习其他函数请参见Oracle Hyperion Essbase Technical Reference。

最好的学习方法就是把所有的函数都学习一遍，这样以后用到MDX解决问题的时候就可以信手拈来。

学习了函数之后，还有一块重要的东西要学习，它是WITH Section.在WITH Section部分可以定义各种复杂的计算，功能十分强大。下面的一个例子就是使用WITH来实现合计。

WITH

SET [Top3BevsPerRegion] AS

'Generate ({[Market].children}, Crossjoin ({[Market].Currentmember}, TopCount ([Product].Children, 3, ([Measures].[Sales],[Year]))))'

MEMBER [Product].[Total] AS

'SUM(TopCount ([Product].Children, 3, ([Measures].[Sales],[Year])),[Measures].[Sales])'

SET [TT] AS

'Crossjoin ({[Market].Children},{[Product].[Total]})'

SELECT

{[Year].Children} ON COLUMNS,

Hierarchize({[Top3BevsPerRegion],[TT]}) ON ROWS

FROM Sample.Basic

WHERE ([Scenario].[Actual], [Measures].[Sales])

**最后呢，当然要学习一下MDX语句执行的顺序是怎样的呢？如下**：

WITH ...

SELECT [NON EMPTY] ... [HAVING ...] on axis(0),

... on axis(1), ...

FROM ...

WHERE ...

1.解析FROM语句，选定查询的Cube；

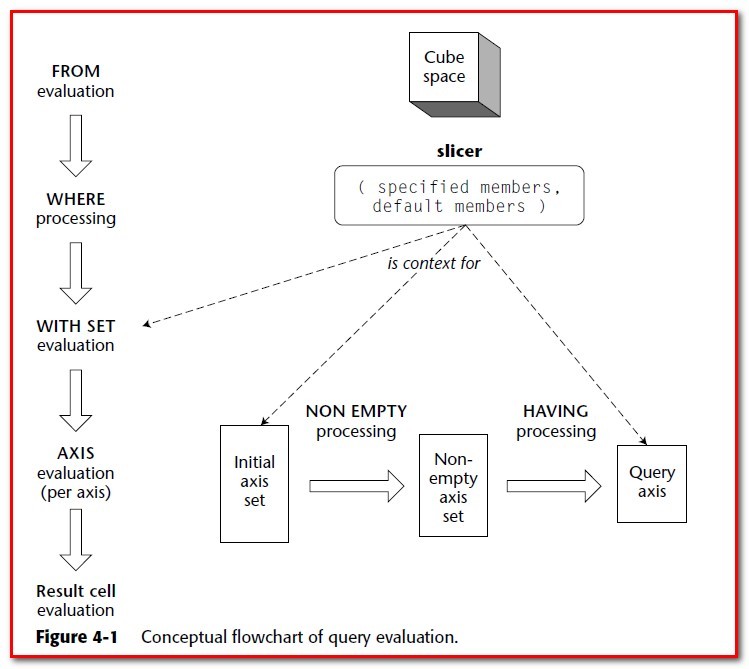
2.解析WHERE语句，对Cube做切块；

3.解析WITH语句中的命名SETS，执行预定义计算；

4.解析每一个轴（Rows，Columns，.....）上的的Tuples，计算轴上的字段；

5.解析NONEMPTY，在轴的交叉点上计算出结果。

图示：



**多维数据集**

多维数据集是联机分析处理 (OLAP) 中的主要对象，是一项可对数据仓库中的数据进行快速访问的技术。多维数据集是一个数据集合，通常从数据仓库的子集构造，并组织和汇总成一个由一组维度和度量值定义的多维结构。

**2. 维度**

是多维数据集的结构性特性。它们是事实数据表中用来描述数据的分类的有组织层次结构（级别）。这些分类和级别描述了一些相似的成员集合，用户将基于这些成员集合进行分析。

**3. 元数据**

不同 OLAP 组件中的数据和应用程序的结构模型。元数据描述 OLTP 数据库中的表、数据仓库和数据集市中的多维数据集这类对象，还记录哪些应用程序引用不同的记录块。

**4. 层次结构(Hierarchies)**

关系数据库以二维平面表的形式组织数据。这些表有一个列维度和一个行维度。在每个行和列的交点处只有一个数据元素。而多维数据库则不同，它是基于称为“多维数据集”的结构，如图所示。多维数据集按“层次结构”组织数据，而不是以表的形式组织数据。

**5. 级别(Levels)**

级别是维度层次结构的一个元素。级别描述了数据的层次结构，从数据的最高（汇总程度最大）级别直到最低（最详细）级别;

每个维度又包含源自属性的“级别”。例如，上图所示的 Source 维度有两个级别：

**Eastern Hemisphere** 和 **Western Hemisphere**。

**6. 成员(Members)**

每个级别又按层次结构组织维度的基本元素，即“成员”。每个成员表示维度中的一个数据元素。

在上图中，**Eastern Hemisphere** 级别有四个成员：**Africa、Asia、Australia 和 Europe**。

成员是维度中的一个项目，表示数据的一次或多次出现。可将维度中的成员看作基础数据库中的一个或多个记录，该记录在此列中的值属于此类别。成员是描述多维数据集中的单元数据时的最低级别的引用。

例如，上图加上了阴影，以表示 **Time.[2nd half].[3rd quarter]** 成员。

如果成员的名称中包含空格或数字，则用方括号 [ 和 ] 分隔成员的名称。尽管 Time 维度的名称仅由一个词构成，也可以使用方括号将此名称括起来。如果为 Time 维度添加了方括号，上图所示的成员也可以表示为：**[Time].[2nd half].[4th quarter]** 。

**成员名称和成员键**

可以用成员名称或成员键引用某个成员。在上一示例中，用成员在 Time 维度中的名称 4th quarter 来引用该成员。但是，如果维度不具有非唯一的成员名称，则成员名称可以重复，也可以更改渐变维度中的成员名称。 引用成员的另一种方法是引用成员键。维度使用成员键明确标识特定成员。在 MDX 中，“与”符号 (&) 用于区分成员键和成员名称。例如，以下引用使用 4th quarter 成员的成员键 Q4：

**[Time].[2nd half].&[Q4]**

使用成员键引用成员可确保在渐变维度以及具有非唯一成员名称的维度中正确标识成员。

**成员函数**

MDX 提供了许多函数，可用于检索其他 MDX 实体（如维度和级别）中的成员。因此，并不总是需要显式引用成员。例如，FirstChild 函数允许从特定维度或级别中检索所有成员。若要获得 Time 维度的第一个子成员，可以显式声明该成员，如下面的示例所示：**Time.[1st half]**

还可以使用 FirstChild 函数返回相同的成员，如下面的示例所示： **Time.FirstChild**

**7. 度量值(Measures)**

Measures 维度的成员称为“度量值”。度量值表示按多维数据集中包含的其他维度组织的数据。上图所示的多维数据集有两个度量值：Packages 和 Last。

在多维数据集中，度量值是一组值，这些值基于多维数据集的事实数据表中的一列，而且通常为数字。此外，度量值是所分析的多维数据集的中心值。即，度量值是最终用户浏览多维数据集时重点查看的数字数据。您所选择的度量值取决于最终用户所请求的信息类型。一些常见的度量值有 sales、cost、expenditures 和 production count 等。

**8. 计算成员(Calculated Members)**

计算成员是由 MDX 表达式填充的成员。它们像正常成员一样返回。MDX 提供了一个可用于创建计算成员的可靠的函数集，从而在多维数据操作方面给您提供了非常大的灵活性。

**9. 单元(Cells)**

包含在多维数据集中的数据元素称为“单元”。通过对多维数据集中包含的每个属性层次结构指定一个成员可以唯一地标识一个单元。

**10. 元组(Tuples)**

标识一个单元的属性的组合称为“元组”。例如，下面的元组标识了上图中值为 240 的一个单元：

**(Source.[Eastern Hemisphere].Africa, Time.[2nd half].[4th quarter], Route.Air, Measures.Packages)**

正如可以指定从关系数据库的表中检索多组列或行一样，您可以指定从多维数据集中检索一组元组。MDX 中用来指一个有序的元组集合的标识符称为“集”。下面的示例标识了上图所示的多维数据集中的一个元组集：**{ (Time.[1st half].[1st quarter]), Time.[2nd half].[3rd quarter]) }**

对于复杂的 MDX 语句，集标识符和集表达式会变得很长，难以理解。而且，许多集表达式还会在不同的语句中重用。您可以创建带有别名的集（称为“命名集”），以使 MDX 查询更容易理解。

元组标识多维数据集中的单元。一个元组由多维数据集中每个层次结构中的一个成员组成（显式或隐式引用）。如果特定层次结构中的成员没有在元组中显式引用，则该层次结构中的默认成员将隐式包含在元组中。 在 MDX 中，元组根据其复杂性依照语法进行构造。如果元组只由一个层次结构中的一个成员组成（通常称为“简单元组”），则下列语法是可以接受的：

**Time.[2nd half]**

如果元组由多个层次结构中显式引用的多个成员组成，则元组所表示的成员必须括在圆括号内，如下面的示例所示。

**(Time.[2nd half], Route.nonground.air)**

由一个成员组成的元组也可以括在圆括号内，但圆括号不是必需的。元组常常编组成集，以便在 MDX 查询中使用。

**元组函数**

有一些返回元组的 MDX 函数，它们可在任何接受元组的地方使用。

**11. 集(Sets)**

集是零个、一个或多个元组的有序集合。集最常用于定义 MDX 查询中的查询轴和切片器轴，因此可以只有一个元组，在某些情况下，也可以为空。下面的示例显示了具有两个元组的集：

**{ (Time.[1st half], Route.nonground.air), (Time.[2nd half], Route.nonground.sea) }**

一个集可以包含同一个元组的多次出现。下面的集是可以接受的：

**{ Time.[2nd half], Time.[2nd half] }**

集是指以元组表示的一组成员组合，或指集中的元组所表示的单元中的值，取决于使用集的上下文。

在 MDX 语法中，将元组用大括号括起来以构造集。

**重要事项**

由一个元组构成的集不是元组。MDX 将它们解释为集。某些 MDX 函数接受元组作为参数，而如果传递单个元组集，则会产生错误。元组和由单个元组构成的集不可互换。

**集函数**

显式键入元组并将它们括在大括号内并不是检索集的唯一方法。MDX 支持许多返回集的函数。冒号运算符允许您使用成员的自然顺序创建集。例如，下面的示例中显示的集包含第一季度到第四季度的元组：

**{[1st quarter]:[4th quarter]}**

如果不使用冒号运算符创建集，可以通过指定下列元组创建相同的成员集：

**{[1st quarter], [2nd quarter], [3rd quarter], [4th quarter]}**

冒号运算符起到包含作用。生成的集中包含冒号运算符两侧的成员。

其他返回集的 MDX 函数可单独使用，也可作为以逗号分隔的成员列表的一部分使用。例如，下列所有 MDX 表达式都是有效的：

**{Time.Children}**

**{Time.Children, Route.nonground.air}**

**{Time.Children, Route.nonground.air, Source.Children}**

**命名集**

命名集是为其创建了别名的集。命名集最常用于复杂的 MDX 查询中，以使这些查询更易阅读，更便于维护。

**12. 表达式(Expressions)**

在 MDX 中，表达式是标识符、值、函数和运算符的组合，Analysis Services 可以通过计算表达式来检索某个对象，如集、成员或类似字符串或数字的标量值。

**13. 语句(Statements)**

MDX 语句是表达式和关键字的组合，Analysis Services 可以通过运行 MDX 语句来执行某个操作，如查询多维数据集或计算子多维数据集的单元。例如，可以使用 MDX SELECT 语句从 Analysis Services 实例中检索多维数据。

**14. 脚本(Scripts)**

MDX 脚本是一个用分号分隔的语句集，Analysis Services 可以通过运行 MDX 脚本在指定的作用域和上下文中执行许多操作

**15. 存储过程(Stored Procedures)**

MDX 以用户定义函数的形式提供扩展性。若要创建这些用户定义函数，可以使用支持 Microsoft .NET Framework 或 COM 接口的任何编程语言。若要使用户定义函数可对多维数据运行，在创建函数时应注意使函数能按 MDX 语法接受参数并返回值。可以从 MDX 表达式、语句和脚本中调用用户定义函数。

当引用或调用存储过程时，应在括号前面指定函数名称。可以在括号内指定表达式（称为“参数”），以提供要传递给参数的数据。调用函数时，必须为所有参数提供参数值，并且必须按照用户定义函数中定义的相同参数顺序指定参数值。

**注意**

Microsoft SQL Server 2005 Analysis Services (SSAS) 中使用“存储过程”这一术语表示这些类型的函数。在 Analysis Services 的早期版本中，这些类型的函数称为“用户定义函数”。

**存储过程的类型**

Analysis Services 支持 COM 程序集和 CLR 程序集。建议使用 CLR 程序集，因为 CLR 程序集具有增强的安全性。如果服务器上安装了 Microsoft Office Excel，也可以使用 Excel 功能。

**MDX 的基本结构**

MDX 的基本结构有三种： Members 、 Tuple 、 Set 。

1.         Members ：指的是维度树上的一个节点，这里有一点需要指出，量度也是一个特殊的维度，所以对于普通维度上的 Member 可以有几下几种表示方法： [Customer] 或 [Time].[1996] 等，对于特殊的维度——量度而言，也可以表示一个 Member ，如： [Measures].[ unit sales] 等。

Member 的表示方法就是用中括号的形式，“ [……] ”。

2.         Tuple ：是由若干个 Members 组成，每一个维度上最多只能有一个 Member ，对于一个 Tuple 而言至少有一个维度，多则不限，同时对于没有列出来的那就表示为默认的 Member 。 Tuple 的表示方法是小括号“ () ”，因为其又有 Member 构成，所以通常的格式为“ ([…],[…][…],…) ”；示例：

a)         ([Regin ].[USA])

b)        ([product].[ computers],[time].[2008]) 。

3.         Sets ：同一维度上若干个 Members 的集合，或者是若干个 Tuples 的集合，但这里有一个地方需要注意，那就是如果是若干个 Tuple 组成的集合是，各个 Tuple 里的 Member 之间存在着一定的对应关系。集合的表示方法用大括号“ {} ”，所以可能的表示方法为：

a)         {[time].[2008],[time].[2009],[time].[2000]} ，这里 Set 是由同一维度的若干个 Member 组成。

b)        {([computer],[usa ]),([ mobile],[china])} ，这个 Set 是由两个 Tuples 组成，这里大家可以看到，在第一个 Tuple 当中，第一个 Member 是名为 computer 的产品，所以后面的 Tuple 的第一个 Member 也必须是一个产品，所以我们这里看到的是 mobile ，第一个 Tuple 里第二个 Member 是一个国家，所以第二个 Tuple 的第二个 Member 也必须是一个国家名，依次类推。

**MDX 查询语句**

一个标准的 MDX 查询语句就是由我们前面介绍的 MDX 的三个基本对象构成，也就是 Member 、 Tuple 、 Set 。

一个标准的 MDX 查询的语法如下：

SELECT Set ON COLUMNS,

       Set ON ROWS

FROM Cube

WHERE Tuple

示例：

SELECT

{[time].[ 1997],[time].[1998]}

ON COLUMNS,

{([product].[drink],[customer].[gender].[F]),( [product].[food],[customer].[gender].[M])}

ON ROWS

FROM

[Sales]

WHERE

([Measures].[ StoreSales ])

这个 MDX 查询就表示：查询时间为 1997 和 1998 两年的 drink 类产品，女消费者和 food 类产品男消费者的 Store Sales 。

**MDX 表达式**

**children**

它的作用是列出指定分类下的所有项，如

1.         [product type].[food].children ，这就表示列出产品类型为 food 的所有产品。

2.         [region ].[country].children ，这就表示列出所有的国家名称。

该函数不能用在 Tuple 中

**CurrentMember**

当前某个 Member ，举例：

某个产品销售额贡献度计算，对于某一个产品或一种类型的产品而言它的销售额贡献度指的是它的销售额与所有产品销售额的比值，此时，如果我们要用 MDX 来计算这个贡献度的值是可以采用 CurrentMember 来实现，如：

此时我们的 MDX 表达可以是：

([product].currentMember ,[ Measures].[stores sales]) / ([product].[ all products],[Measures].[stores sales])

同时因为默认情况下指的就是 currentMember ，所以上面的 currentMember 可以去掉：

([Measures].[ stores sales]) / ([product].[ all products],[Measures].[stores sales])

所有 MDX 表达式的写法是由若干个 Tuple 组成 ，比如上面的这个表达式就是由两个 Tuple 的相除构成。

**prevMember 和 nextMember**

这两个元素分别指的是当前 Member 的上一个 Member 和当前 Member 的下一个 Member 。

利用这两个元素我们可以来实现实际当中常见的同期比（前期比），所谓的同期比指的 是：

（当前 member- 当前 member 的前一个 member ） / 当前 member 的前一个 member

对应到 MDX 表达式我们可以用 prevMember 和 nextMember 来实现，如：

(([time].currentMember ,[ Measures].[store sales])-([time].prevMember ,[Measures].[storessales ]))

/

([time].prevMember ,[ Measures].[store sales])

当然因为 currentMember 可以省略，所以上面的写法也可以是：

(([ Measures ].[store sales])-([time].prevMember ,[Measures].[storessales ]))

/

([time].prevMember ,[ Measures].[store sales])

**Parents 、 FirstChild 、 Descendants**

下面的这张图就显示一个树

这张图中 all 这个节点是 97 和 98 的 parents ， 97 和 98 也是 all 的 children ，同时 97 是 all 的 firstChild

如果我们要取到 97 下的所有的月份，那么我们就需要用 Descendants 来实现，它的语法是：

Descendants( [time].[97],month) ，这里的 month 是一个 level 的名称。所有它的格式为： Descendants([ 一个 Member], 一个 level 的名称 )

比如在食品销售当中，我们要知道哪一类的食品更受消费者欢迎，这样我们需要只知道各类食品的销量，比如食物（ food ）、饮料 (drink) 等的具体销量 (unit sales) ，然后再计算出所有食品总的销量（ total sales ） , 如果采用 MDX 我们可以很容易的实现，采用 MDX 后的计算哪种食品更受欢迎的表达式如下：

([food].currentMember ,[ Measures].[unit sales])

/

([food].currentMember.parent ,[ Measures].[total sales])

因为 currentMember 可以省略，所以上面的写法又可以是：

( [ Measures].[unit sales])

/

([food].currentMember.parent ,[ Measures].[total sales])

1、 百分比的格式化

with member [时间.时间].[<%=subStr1 %>] as 'Sum(([时间.时间].[全部]<%=subStr4 %> : [时间.时间].[全部]<%=subStr5 %>))'

member [入库机构.入库库存机构].[合计] as '<%=hjdeptnames %>'

member [material num.器材编码].[合计] as '[material num.器材编码].[所有器材编码]'

member [出库机构.出库库存机构].[小计] as '[出库机构.出库库存机构].[所有出库库存机构]'

member [Measures].[满足率] as '[Measures].[发料数量]/[Measures].[申请数量]',FORMAT\_STRING = "#.#%"

select

NON EMPTY Hierarchize(Crossjoin({[时间.时间].[<%=subStr1 %>]},

Union(Crossjoin({<%=StrWuzi%>},{[Measures].[申请数量],[Measures].[发料数量], [Measures].[发料单价],[Measures].[发料小计]}),

Crossjoin({[material num.器材编码].[合计]},{[Measures].[申请小计],[Measures].[发料小计],[Measures].[满足率]})

)

)) ON COLUMNS,

NON EMPTY Crossjoin({<%=deptnames %>, [入库机构.入库库存机构].[合计]}, {[出库机构.出库库存机构].[所有出库库存机构].[计划与物资管理部].Children,[出库机构.出库库存机构].[小计]}) ON ROWS

from [nySQerp]

当然也可以格式化成其他的。比如：#,###.00，￥#,##0.00等

2、 为字段设置颜色

WITH MEMBER [Measures].[Profit] AS

'([Measures].[Store Sales] - [Measures].[Store Cost])',

FORMAT\_STRING = Iif([Measures].[Profit] < 100000, '|#|style=green',

'|#|style=red')

SELECT {[Measures].[Store Sales], [Measures].[Profit]} ON COLUMNS,

{[Product].CurrentMember.Children} ON ROWS

FROM [Sales]

3、 对union的运用

with member [时间.时间].[<%=subStr1 %>] as 'Sum(([时间.时间].[全部]<%=subStr4 %> : [时间.时间].[全部]<%=subStr5 %>))'

member [时间.时间].[<%=subStr1 +"入库"%>] as 'Sum(([时间.时间].[全部]<%=subStr4 %> : [时间.时间].[全部]<%=subStr5 %>))'

member [时间.时间].[<%=subStr1 +"出库"%>] as 'Sum(([时间.时间].[全部]<%=subStr4 %> : [时间.时间].[全部]<%=subStr5 %>))'

member [库存机构.库存机构].[合计] as '[库存机构.库存机构].[所有库存机构]'

member [material num.器材编码].[小计] as '[material num.器材编码].[所有器材编码]'

member [时间.时间].[<%=beginTime%>] as '[时间.时间].[全部]<%=subStr4 %>'

member [时间.时间].[<%=endTime%>] as '[时间.时间].[全部]<%=subStr5 %>'

member [Measures].[期初结余] as '[Measures].[当日结余数量]'

member [Measures].[期末结余] as '[Measures].[当日结余数量]'

select

NON EMPTY Union(

Union(

Union(

Crossjoin({[时间.时间].[<%=subStr1 %>]},{[Measures].[单价]}),

Crossjoin({[时间.时间].[<%=beginTime%>]},{[Measures].[期初结余]})

),

Union(

Crossjoin(

{[时间.时间].[<%=subStr1 +"入库"%>]},

{[Measures].[收料数量], [Measures].[入库数量小计]}

),

Crossjoin(

{[时间.时间].[<%=subStr1 +"出库"%>]},

{[Measures].[发料数量], [Measures].[出库数量小计]}

)

)

),

Crossjoin({[时间.时间].[<%=endTime%>]},{[Measures].[期末结余]})

)ON COLUMNS,

NON EMPTY Union(

Crossjoin(

{[库存机构.库存机构].[所有库存机构].Children},{[material num.器材编码].[所有器材编码].Children,[material num.器材编码].[小计]}

),

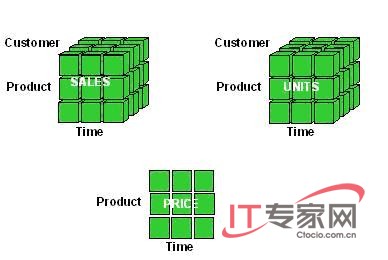
Crossjoin({[库存机构.库存机构].[合计]},{[material num.器材编码].[小计]})

) ON ROWS

from [zancun]

　这一系列文章旨在使你从OLAP的基础概念到更高级的特性——例如使用OLAP分析函数的大型库创建你自己的计算，以及更具数据库管理员类型的主题——例如监控立方体数据加载和聚合过程。

　　在下面的示例中，只有一个时间维度，尽管它显示了三个时间。在图片中的这三个度量有不同的形状，或维度。Sales和Units都是以Customer、Product和Time为维度的。Price只能以Product和Time为维度，它不使用Customer维度，因为价格不随客户的不同而不同。



**层级**

　　 维度层级是可选的，但是[**OLAP**](http://whatis.ctocio.com.cn/searchwhatis/359/6025859.shtml)系统常见的。一个层级是一个逻辑结构，它将一个维度的成员分组以用于分析。例如：

　　· 一个Time维度可能有一个描述了月份怎样分组以显示一个季度和季度怎样分组以显示一个整年的层级。

　　· 一个Organization维度可能有一个使得你可以很容易地确认一个特定经理直属报告的层级。

　　如果需要的话每一个维度都可以有多个层级。例如，时间维度可以有一个层级显示Julian日历，而另一个层级显示一个财务日历。



**OLAP基本概念概述（组图）**

作者: 戴羽, 　出处:IT专家网,　责任编辑: 李书琴,

2008-05-21 10:46

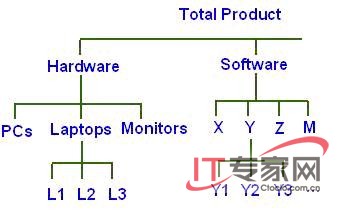
　　这一系列文章旨在使你从OLAP的基础概念到更高级的特性——例如使用OLAP分析函数的大型库创建你自己的计算，以及更具数据库管理员类型的主题——例如监控立方体数据加载和聚合过程。

　　· 在级别间导航： 维度的层级使得可以向下钻取到更低的级别或导航(向上钻取)到更高的级别。在时间维度的成员“2005”上向下钻取会将你导航到从2005 Q1到2005 Q4。在一个日历年份层级中，对2005 Q1向下钻取会将你导航到从05年1月到05年3月。这类的关系使得用户更加容易地导航大型多维数据。

　　· 从子值到父值的聚合：父亲显示了它的孩子的聚合。较低级别的数据值聚合进更高级别的数据值。维度被组织为层次化以便不同聚合级别的数据可以一起有效地操纵以用于分析和显示。

　　· 从父值到子值的分配：聚合的相反是分配，它经常被计划、预算和类似的应用程序所使用。这里，层级的角色是确认特定维度成员的孩子和子孙以用于“从上至下”的预算分配(在其它使用中)。

　　· 将成员分组以用于计算： 共享和索引计算利用了层级关系(例如，每一个产品所占总收益的百分比)，或者产品收入在某一范畴的百分比，或者一个零售区[**域**](http://whatis.ctocio.com.cn/searchwhatis/194/7352194.shtml)的成本占地理区域的百分比)。



　　在这个例子中，你可以进行下面的产品层级：

　　· 在层级中从最低到最高的每一个级别中导航。

　　· 从最高级别导航到最低级别。

　　· 聚合层级中从最低级别获取的数据(各个产品)到最高级别(总产品)。

**级别**

　　 每一个级别显示了层级中的一个位置。底层级别上的级别包含了聚合它下面级别的值。不同级别的成员有一个一对多的父子关系。一个层级一般包含几个级别，而一个单独的级别可以包含进不只一个的层级。

　　如果Sales度量的数据是存储在Product级别的，那么产品维度的更高级别使得销售数据可以正确地聚合进Subcategory、Category，以及所有的Products级别。

　　如果在一个维度上建有多个层级，那么可能一个层级会显示在不只一个的层级中或可能只存在于一个层级中。

层级的类型 在一个多维模型中，有两个基本的层级类型：

　　· 基于等级

　　· 基于值

　　大多数层级是基于等级的，包括在上一节中显示的产品维度层级和这一节中显示的时间维度层级。在时间层级示例中，在层级中有日、月、季度和年份级别。

　　销售一般也有一个基于级别的结构，如下面的例子所示：

　　Representative > Area > Region > Country > Continent > World

　　其它的维度可能有不是很严格的基于级别的层级，例如：

　　在公司图表中有一个清晰的层级，但是总裁的所有直属报告可能不是在同一级别。在这个例子中，两个VP(副总裁)和总裁的Admin(行政助理)都是总裁的直属报告，但是他们不是在同一级别。VP不是Admin级别，Admin也不是VP级别。

WITH

MEMBER MEASURES.DIMENSIONNAME AS [Date].[Calendar].CURRENTMEMBER.DIMENSION.NAME

MEMBER MEASURES.HIERARCHYNAME AS [Date].[Calendar].CURRENTMEMBER.HIERARCHY.NAME

MEMBER MEASURES.LEVELNAME AS [Date].[Calendar].LEVEL.NAME

SELECT

{MEASURES.DIMENSIONNAME, MEASURES.HIERARCHYNAME, MEASURES.LEVELNAME}

ON Columns,

[Date].[Calendar].MEMBERS

ON Rows

FROM [Adventure Works]