Entrepôts de données : Introduction au langage MDX (Multi-Dimensional eXtensions) pour l'OLAP

(7.1)

Bernard ESPINASSE
Professeur à Aix-Marseille Université (AMU)
Ecole Polytechnique Universitaire de Marseille

Septembre 2009

- **Schéma XML d'une BD multidimensionnelle**
- **Introduction à MDX**
- **Syntaxe de base de MDX**
- **Membres et tuples dans MDX**
- **Fonctions sur les membres et les ensembles (Sets) de MDX**
- **Expressions avancées de MDX**

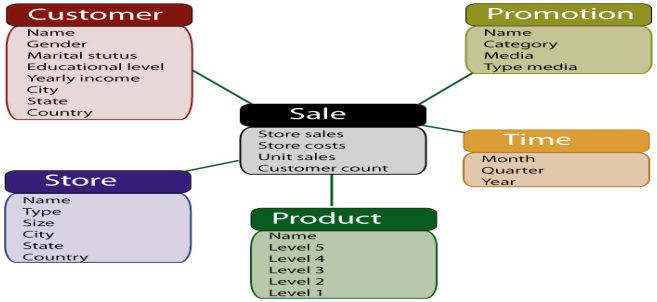
Plan

1 – Schéma XML d'une BD multidimensionnelle

- **Rappels de modélisation multidimensionnelle**
- **Schéma d'une BD multidimensionnelle en XML (Mondrian)**

Rappel de modélisation multidimensionnelle (1)

- On considère un cube « Sales » pour l'analyse de ventes d'une compagnie hypothétique :
- Ce cube contient plusieurs dimensions :
  - le temps
  - la géographie (magasin (Store), …)
  - les produits,
  - les clients
  - …
- **Exemple de cube « Sales » :**



Rappel de modélisation multidimensionnelle

- un **cube** est composé de **dimensions**
- une **dimension** peut contenir une ou plusieurs **hiérarchies** : la dimension "**Time**" contient 2 hiérarchies :

  "Year, Quarter, Month" et "Year, Week, Day"

- Une **hiérarchie** est composée de **niveaux** ("levels") correspondant à un des attributs de la base de données :
  - **hiérarchie** "**Time**" est composée des niveaux "Year", "Quarter" et "Month"
  - **hiérarchie** "**Store**" est composée des niveaux "Country", "State", "City", "Store_Name"
- Un **niveau** est composé de **membres** qui sont les valeurs d'un niveau détectées par le moteur OLAP et stockées dans les métadonnées :
  - les membres du niveau "**Country**" sont *"**France**", "Canada" et "USA"*
  - les membres du niveau "**City**" sont *"**Marseille**", "Lyon" et "Paris".*
- Une **mesure** est une quantité intéressante que l'on souhaite observer, par exemple :
  - montant des ventes
  - quantité de produit vendus

Modélisation multidimensionnelle : hiérarchies des dimensions

Soit les hiérarchies de dimensions suivantes :

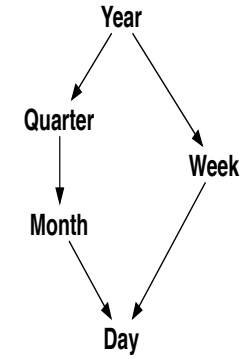
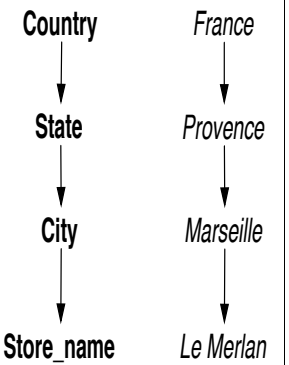
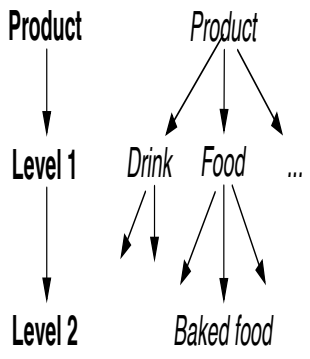
Hiérarchie Time	Hiérarchie Store ou Regions	Hiérarchie Product
		

Schéma d'une BD multidimensionnelle

- Un **schéma** = modèle logique définissant une **BD multidimensionnelle** ainsi que les **structures** associées : **cubes, dimensions, hiérarchies, niveaux** et **membres**
- Il donne aussi la **source des données** représentées dans le modèle logique
- Il est en général **en étoile**, se traduit par un **ensemble de tables relationnelles**
- **Composants majeurs d'un schéma :**
  - **cube** = collection de dimensions et de mesures dans un domaine particulier.
  - **dimension** = attribut, ou ensemble d'attributs, à travers lesquels sont observées les mesures
  - **mesure** = quantité intéressante, qu'on souhaite observer (Ex : montant des ventes, …)

Exemple :

- on peut vouloir observer la vente des produits selon leurs *couleurs*, le *sexe* du client et le *magasin* où sont vendus ces produits
- la couleur du produit, le sexe du client et le magasin de vente sont des dimensions.

Exemple de schéma XML sous Mondrian (1)

Un **schéma** peut être spécifié par un **document XML :**

- Exemple de schéma XML sous **Mondrian** (moteur OLAP) :

```
<Schema>
 <Cube name="Sales">
  <Table name="sales_fact_1997"/>
   <Dimension name="Gender" foreignKey="customer_id">
    <Hierarchy hasAll="true" allMemberName="All Genders" primaryKey="customer_id">
     <Table name="customer"/>
     <Level name="Gender" column="gender" uniqueMembers="true"/>
    </Hierarchy>
   </Dimension>
   <Dimension name="Time" foreignKey="time_id">
    <Hierarchy hasAll="false" primaryKey="time_id">
     <Table name="time_by_day"/>
       <Level name="Year" column="the_year" type="Numeric" uniqueMembers="true"/>
       <Level name="Quarter" column="quarter" uniqueMembers="false"/>
       <Level name="Month" column="month_of_year" type="Numeric" uniqueMembers="false"/>
    </Hierarchy>
   </Dimension>
   <Measure name="Unit Sales" column="unit_sales" aggregator="sum" formatString="#,###"/>
   <Measure name="Store Sales" column="store_sales" aggregator="sum" formatString="#,###.##"/>
   <Measure name="Store Cost" column="store_cost" aggregator="sum" formatString="#,###.00"/>
   <CalculatedMember name="Profit" dimension="Measures" formula="[Measures].[Store Sales]-[Measures].[Store
Cost]">
    <CalculatedMemberProperty name="FORMAT_STRING" value="$#,##0.00"/>
   </CalculatedMember>
 </Cube>
</Schema>
```

Exemple de schéma XML sous Mondrian (2)

Ce **schéma** contient un seul cube de ventes, appelé "Sales" : **<Cube name="Sales">**

- dans ce cube les **ventes** sont observées sur :
  - 2 dimensions « *Time* » et « *Gender* », et
  - 4 mesures « *Unit Sales* », « *Store Sales* », « *Store cost* » et « *Profit* »
- la **table de faits** (ici "sales_fact_1997") contient les colonnes à partir desquelles les mesures sont calculées et les références vers les tables des dimensions
- chaque **mesure** a un **nom**, une **colonne de correspondance dans la table de faits**, un **opérateur d'agrégation**
- **l'opérateur d'agrégation** est souvent "**sum**", mais aussi "**count**", "**min**", "**max**", "**avg**" et "**distinct-count**" peuvent être utilisés
- ici la mesure (<Measure>) « Profit » est **calculée** à partir des mesures « Store Sales » et « Store cost »

Exemple de schéma XML sous Mondrian (3)

Exemple : une représentation XML de la dimension "Gender" :

```
<Dimension name="Gender" foreignKey="customer_id">
  <Hierarchy hasAll="true" primaryKey="customer_id">
    <Table name="customer"/>
    <Level name="Gender" column="gender" uniqueMembers="true"/>
  </Hierarchy>
</Dimension>
```

- la **dimension "Gender"** donne le ***sexe du client***
- cette dimension a **1 seule hiérarchie** et **1 seul niveau**
- elle prend ses valeurs à partir de la colonne "**gender**" de la table "**customer**"
- la colonne "gender" a 2 valeurs **'F'** et **'M'**
- la dimension "Gender" a ainsi 2 membres **"[Gender].[F]"** et **"[Gender].[M]"**

Exemple de schéma XML sous Mondrian (4)

Hiérarchie multiple : une dimension peut contenir plusieurs hiérarchies

Ex : la dimension "Time" contient **2 hiérarchies** : "Year, Quarter, Month" et "Year, Week, Day" :

```
<Dimension name="Time" foreignKey="time_id">
  <Hierarchy hasAll="false" primaryKey="time_id">
    <Table name="time_by_day"/>
    <Level name="Year" column="the_year" type="Numeric" uniqueMembers="true"/>
    <Level name="Quarter" column="quarter" uniqueMembers="false"/>
    <Level name="Month" column="month_of_year" type="Numeric"
uniqueMembers="false"/>
  </Hierarchy>

  <Hierarchy name="Time Weekly" hasAll="false" primaryKey="time_id">
    <Table name="time_by_week"/>
    <Level name="Year" column="the_year" type="Numeric" uniqueMembers="true"/>
    <Level name="Week" column="week" uniqueMembers="false"/>
    <Level name="Day" column="day_of_week" type="String" uniqueMembers="false"/>
  </Hierarchy>
</Dimension>
```


Exemple de schéma XML sous Mondrian (6)

Mapping des dimensions et des hiérarchies avec les tables :

- une dimension est jointe avec un cube à l'aide de 2 colonnes : une dans la **table de faits** et une dans la **table de dimensions**
- **l'élément <Dimension>** a une clé étrangère (l'attribut **foreignKey**), qui est le même dans la table de faits
- **l'élément <Hierarchy>** a une clé primaire (l'attribut **primaryKey**)
- si une hiérarchie est organisée selon plusieurs tables, on peut utiliser l'attribut **primaryKeyTable** pour lever toute ambiguïté :

```
<Cube name="Sales">
  ...
  <Dimension name="Product" foreignKey="product_id">
    <Hierarchy hasAll="true" primaryKey="product_id" primaryKeyTable="product">
      <Join leftKey="product_class_key" rightAlias="product_class" rightKey="product_class_id">
        <Table name="product"/>
        <Join leftKey="product_type_id" rightKey="product_type_id">
          <Table name="product_class"/>
          <Table name="product_type"/>
        </Join>
      </Join>
    <!-- Level declarations ... ->
    </Hierarchy>
  </Dimension>
</Cube>
```

2 – Introduction à MDX

- **Origine de MDX**
- **MDX versus SQL**

Origine de MDX

- **MDX** est l'acronyme de **M**ulti **D**imensional e**X**pression
- est un **langage de requêtes OLAP** pour les bases de données multidimensionnelles
- a été inventé par Mosha Pasumansky au sein de Microsoft
- a été présenté pour la première fois en 1997 comme un volet de la spécification OLE DB for OLAP (ODBO)
- version commerciale Microsoft OLAP Services 7.0, Microsoft Analysis Services en1998
- dernière version de la spécification OLE DB for OLAP (ODBO) en 1999
- MDX peut être présenté comme une **extension de SQL** : structures de requêtes et mots clés similaires, mais avec de grandes différences.
- Bibliographie et sources du cours :
  - « MDX for Everyone », M. Pasumansky, avril 1998.
  - « Fast track to MDX », M. Whithehorn, R. Zare, M. Pasumansky, Springer, 2006.
  - « Introduction to Multidimensionnal Expressions (MDX) », auteur inconnu.
  - Cours de Kamel Aouiche, ERIC, Université de Lyon 2, 2004
  - Cours de Joseph Fong, City University of Hong Kong, 2008

MDX versus SQL (1)

- MDX est fait pour **naviguer** dans les bases **multidimensionnelles** et pour définir des **requêtes** sur tous leurs **objets** (dimensions, hiérarchies, niveaux, membres et cellules) afin d'obtenir (simplement) une **représentation** sous forme de **tableaux** croisés
- MDX ressemble à SQL par ses **mots clé SELECT**, **FROM**, **WHERE,** mais :
  - SQL construit des **vues relationnelles**
  - MDX construits des **vues multidimensionnelles** des données
- **Analogies** entre termes **multidimensionnels** (**MDX**) et **relationnels** (**SQL**) :

Multidimensionnel (MDX)	Relationnel (SQL)
Cube	**Table**
Niveau (Level)	**Colonne** (chaine de caractère ou valeur numérique)
Dimension	**plusieurs colonnes liées** ou une table de dimension
Mesure (Measure)	**Colonne** (discrète ou numérique)
Membre de dimension (Dimension member)	**Valeur** dans une colonne et une ligne particulière de la table

MDX versus SQL (2)

- **Structure générale d'une requête SQL** :

```
SELECT column1, column2, …, columnn  FROM table
```

- **Structure générale d'une requête MDX** :

```
SELECT  axis1 ON COLUMNS, axis2 ON ROWS  FROM cube
```

- **FROM** spécifie la **source** de données :
  - en **SQL** : une ou plusieurs tables
  - en **MDX** : un cube
- **SELECT** indique les **résultats** que l'on souhaite récupérer par la requête :
  - en **SQL** :
    - une vue des données en 2 dimensions (lignes (rows) et colonnes (columns))
    - les lignes ont la même structure définie par les colonnes
  - en **MDX** :
    - nb quelconque de dimensions pour former les résultats de la requête.
    - terme d'axe pour éviter confusion avec les dimensions du cube.
    - pas de signification particulière pour les rows et les columns, mais il faut définir chaque axe : axe1 définit l'axe horizontal et axe2 définit l'axe vertical

3 – Syntaxe de base de MDX

- **Structure générale d'une requête MDX**
- **Spécification de Membres, Tuples et Sets dans MDX**
- **Spécification d'un axe dans MDX (simple et en énumération)**
- **Spécification de filtres (Slicers) dans MDX : clause WHERE**
- **Insertion de commentaires en MDX**

Structure générale d'une requête MDX

Un prototype de requête MDX est donné par la syntaxe suivante :

```
SELECT [<axis_specification>
[, <spécification_des_axes>...]]
FROM [<spécification_d_un_cube>]
[WHERE [<spécification_de_filtres>]]
```


Membres

- Un **membre** = une **instance** d'un **niveau** d'une **dimension**, spécifié entre **crochets** [..]
  Ex : [Food], [Drink] sont des membres de la dimension "Products" au niveau 1
- Les **membres** = items accessibles dans les hiérarchies et peuvent être référencés de différentes façons :
  [1997]
  [Time].[1997]
  [Product].[Food]
  [Product].[Food].[Baked Goods]
  [Product].[All Products].[Food].[Baked Goods]
- Les **enfants** d'un **membre** = membres du niveau immédiatement en dessous de celui-ci
- Ex. d'utilisations de membres dans des requêtes simples :

```
SELECT [Time].[1997] ON COLUMNS
FROM [Sales]
SELECT [Product].[Food] ON COLUMNS
FROM [Sales]
SELECT [Product].[Food].[Baked Goods] ON COLUMNS
FROM [Sales]
SELECT [Product].[All Products].[Food].[Baked Goods] ON COLUMNS
FROM [Sales]
```


Tuples, cellule et mesure

- Un **tuple** = suite de plusieurs **membres** entre **parenthèses** séparés par une **virgule** :
  Ex : ([Time].[1997], [Product].[Food])
  on peut omettre les parenthèses s'il s'agit d'un tuple ne contenant qu'un seul membre.
- Un **tuple** donne la liste des membres qui identifient une ou plusieurs **cellules** dans un cube.
- Une **cellule** est l'intersection des dimensions d'un cube de données :
  Ex :

```
SELECT ([Time].[1997], [Product].[Food]) ON COLUMNS
FROM [Sales]

SELECT ([Product].[All Products].[Food].[Baked Goods], [1997]) ON COLUMNS
FROM [Sales]
```

- Dans un tuple, les **mesures** sont traitées comme une dimension particulière, nommée [***Measures***] :

```
SELECT ([Measures].[Unit Sales], [Product].[All Products].[Food].[Baked Goods]) ON
COLUMNS
FROM [Sales]
```

Sets (1)

- Un **set** = un **ensemble ordonné** de **tuples**
- Un **set** peut être vu comme une **plage de valeurs**
- Un set **commence** par une accolade "**{**", dans laquelle sont énumérés les **tuples** séparés par des **virgules**, et se termine par une accolade appariée "**}**"

Ex1 :

```
SELECT
{
  ([Measures].[Unit Sales], [Product].[All Products].[Food].[Baked Goods]),
  ([Measures].[Store Sales], [Product].[All Products].[Food].[Baked Goods])
}
ON COLUMNS
FROM [Sales]
```

ce set contient :

- **2 mesures différentes** (Units sales et Store Sales) et
- **le même membre** (Baked Goods) :

Sets (2)

- **Ex2** : un set qui comporte **2 mesures et 2 membres différents** de la **même dimension** sur **2 niveaux différents** ([Food] et [Baked Goods]) :

```
SELECT
{  ([Measures].[Unit Sales], [Product].[Food]),
   ([Measures].[Store Sales], [Product].[Food].[Baked Goods]) }
ON COLUMNS
FROM [Sales]
```

- **Ex3** : un set qui a la **même mesure** et **2 différents membres contigus** ([Food] et [Drink]) :

```
SELECT
{  ([Measures].[Unit Sales], [Product].[Food]),
   ([Measures].[Unit Sales], [Product].[Drink]) }
ON COLUMNS
FROM [Sales]
```

- **Ex4** : un set qui ne contient **qu'un seul membre** ([1997]) :

```
SELECT
{  ([1997]) } ON COLUMNS
FROM [Sales]
```


Spécification d'un axe en MDX (1)

Plusieurs spécifications possibles pour un **même axe** en MDX :

- un **set** suivi du mot clef « **ON** » suivi d'un **nom d'axe spécifique**
- fait référence à un numéro d'ordre s'il y a plus de 2 axes de restitution, ou simplement aux noms d'axes explicites « **COLUMNS** » et « **ROWS** »

Ex: unités vendues "[Measures].[Unit Sales]" par an pour les produits "Drink" et "Food" :

```
SELECT
{
   ([Measures].[Unit Sales], [Product].[Food]),
   ([Measures].[Unit Sales], [Product].[Drink]) } ON AXIS(0),
{
   ([Time].[1997]),
   ([Time].[1998]) } ON AXIS(1)
FROM [Sales]
```

ou

```
SELECT
   { ([Measures].[Unit Sales], [Product].[Food]),
   ([Measures].[Unit Sales], [Product].[Drink]) } ON COLUMNS,
{
   ([Time].[1997]),
   ([Time].[1998]) } ON ROWS
FROM [Sales]
```


Spécification d'un axe en MDX (2)

- Une façon simple est de définir un axe est de présenter sur l'axe **tous** les **membres** de d'une **dimension** :

```
<dimension name>.MEMBERS
```

- Si l'on veut voir apparaître tous les membres de la dimension à un certain niveau de cette dimension :

```
<dimension name>.<level name>.MEMBERS
```

par exemple la requête :

```
SELECT
   Years.MEMBERS ON COLUMNS,
   Régions.Continent.MEMBERS ON ROWS
FROM Sales
```

donne le résultat :

	1994	1995	1996	1997
N. America	120,000	200,000	400,000	600,000
S. America	-	10,000	30,000	70,000
Europe	55,000	95,000	160,000	310,000
Asia	30,000	80,000	220,000	200,000

Spécification d'un axe en MDX (3)

Cette requête MDX :

```
SELECT
  Years.MEMBERS ON COLUMNS,
  Régions.Continent.MEMBERS ON ROWS
FROM Sales
```

	1994	1995	1996	1997
N. America	120,000	200,000	400,000	600,000
S. America	-	10,000	30,000	70,000
Europe	55,000	95,000	160,000	310,000
Asia	30,000	80,000	220,000	200,000

Son résultat de la requête est une table avec 2 axes :

- l'axe **horizontal** est (i.e. **columns**) contient tous les membres de la **dimension** '**Years**'
- l'axe **vertival** est (i.e. **rows**) contient tous les membres du **niveau** '**Continent**' de la **dimension** '**Regions**'

- **Remarque :** la mesure considérée (auxquelles correspondent les valeurs du résultat) n'est pas précisée, c'est une **mesure par défaut** (**default measure**) : '**sales**' (valeurs de ventes)

Spécification d'un axe en énumération dans MDX

- Certaines **dimensions** ou **niveaux** ont **plus de 1000 membres** !
- On peut souhaiter **ne pas considérer tous les membres** de la dimensions ou du niveau
- Dans MDX on peut **spécifier une liste de membres à considérer** ainsi :

  { dim.member1, dim.member2, … , dim.membern}

  Ex : *on souhaite considérer seulement les ventes sur les 2 années 1996 et 1997 :*

```
SELECT
  { Years.[1996], Years.[1997] }  ON COLUMNS,
    Regions.Continent.MEMBERS     ON ROWS
FROM Sales
```

Remarques :

- il est recommandé de spécifier des noms de **membres** entre crochet **[ ]** avec aucun symbole blanc, point, …
- ici [1996] et [1997] sont des membres et pas des valeurs
- un **axe** est spécifié dans une expression entre accolades **{ }**
- dans cette requête la **mesure** (measure) pas explicitement définie (mesure par défaut)
- dans MDX la **mesure** est traitée de la même façon qu'une **dimension**
- quand une dimension a le niveau '**All**', le **premier membre** du **premier niveau** sera choisi par **défaut**

Spécification de filtres (Slicers) dans MDX (1)

- Dans la requête précédente on considère que **2 dimensions** : **Regions** et **Years**
- Supposons qu'on s'intéresse non plus aux ventes de tous les **produits,** mais seulement aux **ventes d'ordinateurs**, on définie alors le **nouvel axe** :

  { Products.[Product Group].**Computers** }

- On pourrait ajouter cet axe à notre requête, mais elle contiendrait alors **3 axes**, et tous les outils OLAP ne pourraient le visualiser, aussi on préfère utiliser une opération de **Slice (filtre) :**
- Dans MDX l'opération de **Slice** est traitée par une clause **WHERE** :

```
SELECT
    { Years.[1996], Years.[1997] } ON COLUMNS,
    Regions.Continent.MEMBERS ON ROWS
FROM Sales
WHERE ( Products.[Product Group].[Computers] )
```


Spécification de filtres (Slicers) dans MDX (2)

- La clause **WHERE** a la syntaxe :

  WHERE (member-of-dim1, member-of-dim2, …, member-of-dimn)

- Dans un **Slice** en MDX on peut avoir **plusieurs membres**, mais ils doivent **appartenir à des dimensions différentes** (pas de Slice sur 2 produits différents, par ex. computer et printers)
  - **Ex**: **Slide** sur un produit (computer) et un client particulier (AT&T) :

```
SELECT
  { Years.[1996], Years.[1997] }  ON COLUMNS,
  Regions.Continent.MEMBERS       ON ROWS
FROM Sales
WHERE
(
  Products.[Product Group].[Computers],
  Customers.[AT&T]
)
```

Ventes de computer à AT&T pour les années 1996 et 1997 pour tous les continents.

Spécification de filtres (Slicers) dans MDX (3)

- On peut souhaiter voir plutôt que les ventes (sales mesure par défaut) de computer, le **nombre d'unité expédiées**
- On doit juste **ajouter la mesure** « **units** » dans le **Slice** :

```
SELECT
     { Years.[1996], Years.[1997] } ON COLUMNS,
    Regions.Continent.MEMBERS ON ROWS
 FROM Sales
WHERE
(
   Products.[Product Group].[Computers],
   Customers.[AT&T],
   Measures.[Units]
)
```

Unités de computer expédiées à AT&T pour les années 1996 et 1997 pour tous les continents.

Insertion de commentaires

- Les commandes MDX peuvent être commentées de trois façons différentes :
  - // Commentaire en fin de ligne
  - -- Commentaire en fin de ligne
  - /* Commentaire sur plusieurs lignes
    */
- Les commentaires peuvent être imbriqués comme le montre l'exemple ci-dessous :

```
/* Commentaire sur
    plusieurs lignes /* Commentaire imbriqué */
*/
```


4 – Membres et tuples dans MDX

- **Emboitement (Nest) de tuples dans MDX**
- **Membres calculés dans MDX**
- **Membres NULL et Cellules EMPTY**

Emboitement (Nest) de tuples dans MDX (1)

- Les **axes** peuvent contenir des **membres** ou des **tuples**
- On veut voir les **ventes** de **computers** et **printers** en **Europe** et **Asia** sur les années **1996** et **1997**. On peut le faire avec un requête MDX avec **3 axes** :

```
SELECT
   { Continent.[Europe], Continent.[Asia] }        ON AXIS(0),
   { Product.[Computers], Product.[Printers] }  ON AXIS(1),
   { Years.[1996], Years.[1997] }                    ON AXIS(2)
FROM Sales
```

- Mais le résultat de cette requête n'est pas une table à 2 dimensions, mais un cube à 3 dimensions, **plus difficile à interpréter**
- **Solution** : réécrire la requête en considérant 3 dimensions, **avec seulement 2 axes** et permettant avoir le résultat suivant :

		1996	1997
Europe	Computers		
	Printers		
Asia	Computers		
	Printers		

Emboitement (Nest) de tuples dans MDX (2)

- Les **lignes (ROWS)** de cette table contiennent 2 dimensions : **Computer** et **Printers**
- La **dimension Product** est **emboité** (nested) dans la **dimension Regions** conduisant aux **4 combinaisons** des membres de dimensions {Europe, Asie} et {Computers, Printers}
- Chaque **combinaison de membres** venant de **dimensions différentes** est un **Tuple**
- La requête MDX est alors :

```
SELECT
  { Year.[1996], Year.[1997] } ON COLUMNS,
  {
   ( Continent.Europe, Product.Computers ),
   ( Continent.Europe, Product.Printers  ),
   ( Continent.Asia, Product.Computers ),
   ( Continent.Asia, Product.Printers  )
  } ON ROWS
 FROM Sales
```

Donne les ventes de computers et printers pour l'Europe et l'Asie pour 1996 et 1997.

Membres calculés dans MDX (1)

- MDX permet d'étendre le cube en définissant d'autres **membres de dimensions**, membres qui sont **calculés à partir de membres existants**
- Le syntaxe pour les membres calculés est de mettre la construction suivante **WITH** en face de l'instruction SELECT :
  - **WITH MEMBER parent.name AS 'expression'**
- Le parent du nouveau membre est de **dimension mesure** (measures), le nom est **Profit** et l'expression est : Profit = Sales – Cost :

```
WITH
  MEMBER Measures.Profit AS 'Measures.Sales – Measures.Cost'
SELECT
   Products.MEMBERS ON COLUMNS,
   Year.MEMEBERS ON ROWS
FROM Sales
WHERE ( Measures.Profit )
```

Donne les profits des produits au cours des années

Membres calculés dans MDX (2)

- Les membres calculés sont traités de la même manière que des membres ordinaires
- Ils peuvent être utilisés :
  - **dans la définition d'axes**
  - **dans une clause WHERE** : on peut définir des membres calculés avec d'autres membres calculés
- Ex : le membre calculé `ProfitPercent` (profit pourcentage du coût) :

```
ProfitPercent = Profit / Cost
```

```
WITH
  MEMBER Measures.Profit AS 'Measures.Sales – Measures.Cost'
  MEMBER Measures.ProfitPercent AS 'Measures.Profit / Measures.Cost',
FORMAT_STRING = '#.#%'
SELECT
  { Measures.Profit, Measures.ProfitPercent } ON COLUMNS
FROM Sales
```

- L'instruction `FORMAT_STRING` défini le format du membre calculé

Membres calculés dans MDX (3)

- Supposons que l'on veuille comparer la compagnie entre 1997 et 1998, on peut :
  - construire une requête qui a un axe {[1997], [1998]} et regarde les paires de nombres pour chaque mesure
  - **définir un membre calculé** dans le niveau Year, parallèle à 1997 et 1998, qui contiendra la différence entre eux :
- **Ex :**

```
WITH MEMBER Time.[97 to 98] AS 'Time.[1998] – Time.[1997]'
SELECT
  { Time.[97 to 98] } ON COLUMNS,
  Measures.MEMBERS ON ROWS
FROM Sales
```

Membres calculés dans MDX (4)

Supposons qu'on veuille voir comment les profits ont évolués entre 97 et 98 :

```
WITH
   MEMBER Measures.Profit AS 'Measures.Sales – Measures.Cost'
   MEMBER Time.[97 to 98] AS 'Time.[1998] – Time.[1997]'
SELECT
  { Measures.Sales, Measures.Cost, Measures.Profit } ON COLUMNS,
  { Time.[1997], Time.[1998], Time.[97 to 98] } ON ROWS
FROM Sales
```

On obtient la matrice 3x3 :

	Sales	Cost	Profit
1997	300	220	80
1998	350	210	140
97 to 98	50	-10	60

Membres NULL

Soit la requête suivante :

```
WITH MEMBER Measures.[Sales Growth] AS '(Sales) – (Sales, Time.PrevMember)'
SELECT
  { [Sales], [Sales Growth] } ON COLUMNS,
  Month.MEMBERS ON ROWS
FROM Sales
```

Calcule pour chaque mois la croissante des ventes comparée au mois précédent.

- Pour le premier mois, il n'y a pas de mois précédent dans le cube : au lieu de retourner une erreur, MDX a la notion de **membre NULL** représentant des membres qui n'existent pas
- La sémantique d'un **membre NULL** est :
  1. When the cell contains as one of its coordinates NULL member, the numeric value of the cell will be zero :
     Ainsi `(Sales, [January].PrevMember) = 0`
  2. Any member function applied to NULL returns NULL.
  3. When NULL member is included in set, it is just ignored.
     Ainsi `{[September], [January].PrevMember} = {[September]}`

Cellules EMPTY

- La notion **cellule EMPTY** est liée à celle de membre NULL
- Quand une cellule est hors du cube alors sa valeur est vide, et la valeur calculée pour cette cellule est 0 (zéro).
- On peut avoir des cellules vides pas seulement hors du cube.
- Il est important de distinguer :
  - le fait qu'une donnée existe et sa valeur est 0, et
  - le fait qu'une donnée n'existe pas résultant d'une cellule EMPTY, cellule dont la valeur sera évaluée à 0.
- Pour cela MDX a le prédicat **`IsEmpty`** qui peut être appliqué à une cellule et retourne la valeur booléenne TRUE ou FALSE.

5 – Fonctions sur les membres et ensembles (Set) de MDX

- **Fonctions sur les membres et les dimensions**
- **Opérations sur les ensembles (Sets) dans MDX**
- **Fonctions sur les Sets (Head, Tail, Subset, Topcount, Order, Filter)**
- **Fonctions sur les Sets (CrossJoin)**

Fonctions sur les membres et les dimensions (1)

- On a besoin de fonctions pour **naviguer** dans les **hiérarchies** de **dimensions**
- Les axes d'interrogation définissent les sous-espaces dimensionnels de la requête
- Par exemple écrire `[WA].Parent` (Etat du Washington) est équivalent à écrire `[USA]`
- Si l'on veut connaître les **coordonnées** d'une **cellule** dans le sous-espace de la requête :

```
<dimension name>.CurrentMember
<level name>.CurrentMember
```

- Ex : on veut calculer le *pourcentage des ventes dans chaque ville relative son état*, on utilisera pour trouver l'état d'une ville donnée la fonction **`Parent`** :

```
WITH
  MEMBER Measures.PercentageSales AS
'(Regions.CurrentMember,   Sales) /
(Regions.CurrentMember.Parent, Sales)',FORMAT_STRING = '#.00%'
SELECT
    { Sales, PercentageSales } ON COLUMNS,
    Regions.Cities.MEMBERS ON ROWS
FROM Sales
```


Fonctions sur les Membres de dimensions (2)

Fontion	Signification	Remarque
Parent	Donne le parent du membre de dimension considéré	Déplacement vertical sur une hiérarchie, et on change ainsi de niveau
FirstChild	Donne le premier fils du membre considéré	
LastChild	Donne le dernier fils du membre considéré	
FirstSibling	…	Déplacement horizontal à l'intérieur d'un même niveau
LastSibling	…	
NextMember	Donne le membre suivant du membre considéré	
PrevMember	Donne le membre précédent fils du membre considéré	
FirstSibling	…	

Ex:

- [Aug].PrevMember retourne [Jul]. Les 2 membres appartiennent à [Qtr 3]
- [Oct].PrevMember retourne [Sep]. On traverse la hiérarchie en changeant de parents de [Qtr 4] à [Qtr 3].

Fonctions sur les Membres de dimensions (3)

- On veut définir un **nouveau membre calculé Sales Growth** montrant la **croissance** ou le **déclin** des ventes comparées avec le **précédent mois, trimestre** ou **année** :

```
WITH MEMBER Measures.[Sales Growth] AS '(Sales) – (Sales, Time.PrevMember)'
```

- On peut alors utiliser ce nouveau membre dans la requête :

```
SELECT
  { [Sales], [Sales Growth] } ON COLUMNS,
  Month.MEMBERS ON ROWS
FROM Sales
```

- Autre exemple : voir une chute des ventes pour différents produits et comment ces ventes ont augmenté pour chaque produit dans le dernier mois :

```
SELECT
  { [Sales], [Sales Growth] } ON COLUMNS,
  Product.MEMBERS ON ROWS
FROM Sales
```


Les ensembles (Sets) dans MDX

- MDX propose des fonctions classiques de théorie des ensembles, pour des **ensembles (set) de membres ou de tuples**
- Dans un « set », la seule restriction est que tous les éléments ont la même structure
- Pour un **set de membres** :
  - tous les membres doivent venir de la même dimension(même si ils appartiennent à différents niveaux) :
    - le set **{ [1997], [August] }** est valide,
    - le set **{ [August], [Sales] }** n'est pas valide.
- Pour un **set de tuples** :
  - la dimensionnalité doit être la même et les membres correspondants des tuples doivent venir de la même dimension :
    - le set **{ ([1997], [USA]), ([August], [WA]) }** est valide,
    - le set **{ ([August], [WA]), ([1997], [USA]) }** n'est pas valide, car l'ordre des dimensions dans le tuple est inversé.

Opérations sur les ensembles (Sets) dans MDX

Ces fonctions ont en **entrée** et en **sortie** des **sets de membres** ou des **sets de tuples**

Intersection : `INTERSECT( set1, set2 )`

- Rarement utilisée

Intersection : `UNION( set1, set2 )`

- Très souvent utilisée pour combiner plus de 2 ensembles , parfois on veut l'union d'ensembles avec membres, etc. On peut aussi utiliser les expressions MDX plus concises suivantes :

  { set1, set2, …, setn } ou { member1, set1, set2, member2, …, memberk, setn }

- Ex: on veut voir sur les axes résultats sur l'Europe, USA, tous les états des USA, toutes les villes dans l'état WA et pour l'Asie (**scénario de Drilldown typique**), on peut alors définir un set :

  { [Europe], [USA], [USA].Children, [WA].Children, [Asia] }

- L'équivalent avec l'opérateur **UNION** sera :

  UNION({ [Europe], [USA] }, UNION([USA.Children], UNION([WA].Children, { [Asia] })))

- Certaines implémentations de MDX dispose de l'opérateur + (plus) pour faire l'union de 2 sets : `set1 + set2 + … + setn`

Fonctions sur les Sets (1)

- Fonctions principales qui opèrent sur un set et qui retournent un set :

Fonction	Syntaxe	Description
Head	Head(<< Set >> [,<< Numeric Expression >>])	Eléments de tête d'un set
Tail	Tail(<< Set >> [,<< Numeric Expression >>])	Derniers éléments d'un set
Subset	Subset(<< Set >>, << Start >> [,<< Count >>])	Sous-ensemble d'éléments d'un set
TopCount	TopCount(<< Set >>, << Count >> [,<< Numeric Expression >>])	Les premiers éléments ayant la plus grande valeur de la mesure
Order	Order (<< Set <<, {<<String Expression>> | <<Numeric Expression >>} [, ASC | DESC | BASC | BDESC])	Tri des éléments d'un set
Filter	Filter(<< Set >>, << conditions >>)	Les éléments d'un set qui satisfont le filtre

Fonctions sur les Sets (2) :

- L'exemple suivant montre l'utilisation des fonctions **Topcount**, **Tail** et **Head** :

```
SELECT
{
   ([Measures].[Unit Sales])
}
ON COLUMNS,
{
      (Head([Time].Children, 2)),
      (Tail([Time].Children, 3)),
      (Topcount([Time].Children,1,[Measures].[Unit Sales]))
}
ON ROWS
FROM [Sales]
```

- La requête suivante, avec la fonction **Filter** n'affiche que les cellules dont la mesure "[Unit Sales]" est supérieur ou égale (≥) à une certaine valeur :

```
SELECT
{
   ([Measures].[Unit Sales])
}
ON COLUMNS,
{
      Filter([Time].Children, ([Measures].[Unit Sales]) > 70000)
}
ON ROWS
FROM [Sales]
```


Fonctions sur les Sets : CrossJoin (1)

- La fonction **CrossJoin** permet de croiser des « sets » et ainsi réaliser des tableaux croisés
- Etant donné un set A et un set B, **CrossJoin construit un nouveau set contenant tous les tuples (a,b) où a est dans A et b dans B** :
- **Ex 1 :**

```
SELECT
{
CrossJoin
(
{([Time].[1997].[Q1]), ([Time].[1997].[Q2])},
{([Measures].[Unit Sales]), ([Measures].[Store Sales])}
)
}
ON COLUMNS,
{
([Product].[Drink].Children)
}
ON ROWS
FROM [Sales]
```

Fonctions sur les Sets : CrossJoin (2)

- **Ex 2 :**

```
SELECT
{
 CrossJoin
 (
  {([Time].[1997].[Q1]), ([Time].[1997].[Q2])},
  {([Measures].[Unit Sales]), ([Measures].[Store Sales])}
 )
}
ON COLUMNS,
{
  ([Product].[Drink].Children),
  ([Product].[Food].[Baked Goods])
}
ON ROWS
FROM [Sales]
```


Fonctions sur les Sets : CrossJoin (3)

- **Ex 3 :**

```
SELECT
Crossjoin
(
{[Measures].[Unit Sales], [Measures].[Store Sales]},
{[Time].[1997].[Q1]}
)
ON COLUMNS,
Crossjoin
(
{[Promotion Media].[All Media].[Cash Register Handout], [Promotion Media].[All
Media].[Sunday Paper, Radio, TV],
[Promotion Media].[All Media].[TV]},
{[Gender].Children}
)
ON ROWS
FROM [Sales]
```


Fonctions sur les Sets : CrossJoin (4)

- Dans l'exemple précédent il y a des cellules vides, dû au fait qu'aucun croisement n'est possible sur le 2 axes
- Pour n'afficher que les cellules pleines, utilisez l'opérateur **Non Empty CrossJoin** :

```
SELECT
Crossjoin
(
{[Measures].[Unit Sales], [Measures].[Store Sales]},
{[Time].[1997].[Q1]}
)
ON COLUMNS,
Non Empty Crossjoin
(
{[Promotion Media].[All Media].[Cash Register Handout], [Promotion Media].[All
Media].[Sunday Paper, Radio, TV],
[Promotion Media].[All Media].[TV]},
{[Gender].Children}
)
ON ROWS
FROM [Sales]
```

- **Remarque** : on ne peut pas mettre plus d'une fois la même hiérarchie dans plusieurs axes indépendants d'une requête.

6 – Expressions avancées de MDX

- **Analyse comparative (ParallelPeriod)**
- **Calcul cumulatif**
- **Expressions conditionnelles**

Analyse comparative (ParallelPeriod)

- L'analyse comparative consiste à présenter à côté d'une mesure, la même mesure sur une période parallèle qui lui est antérieure
- On utilise la fonction **ParallelPeriod** :

ParallelPeriod(["niveau"[, "expression numérique" [, "membres"]]])

- où "niveau" représente le niveau, la période, sur lequel on veut "remonter dans le temps", "l'expression numérique" donne le nombre de période, et "membre" permet de fixer un membre sur la dimension temps.
- **Ex**: *la mesure "[Unit Sales]" est affichée en parallèle sur le trimestre en cours et antérieur :*

```
WITH
MEMBER [Measures].[Unit Sales Q-1]
as (  ParallelPeriod( [Time].[1997].[Q1], 1)  )
SELECT
{ ([Measures].[Unit Sales]), ([Measures].[Unit Sales Q-1]) } ON COLUMNS,
{  ([Time].Children)  } ON ROWS
FROM [Sales]
```


Calcul cumulatif (1)

- Il arrive souvent qu'on ait besoin de calculer des cumuls sur une période de temps
- Ex: ***calcul du cumul de la mesure "[Unit Sales]" durant l'année 1997.***
  - Dans un premier temps, on doit retrouver tous les mois de l'année 1997
  - On peut envisager un accès par membre, mais l'expression correspondante pourrait être longue et dépendante des trimestres :

```
SELECT
{
([Measures].[Unit Sales])
}
ON COLUMNS,
    {
        ([Time].[1997].[Q1].Children),
        ([Time].[1997].[Q2].Children),
        ([Time].[1997].[Q3].Children),
        ([Time].[1997].[Q4].Children)
    }
ON ROWS
FROM [Sales]
```


Calcul cumulatif (2)

- Une façon plus élégante et moins fastidieuse est d'utiliser la fonction **Descendants** avec 2 paramètres :
  - 1° paramètre : retourne un ensemble de descendants d'un membre sur un niveau ou d'une distance spécifiée
  - 2° paramètre (optionnel) : permet d'inclure ou d'exclure des descendants d'autres niveaux
- **Ex 1** : *calcul des descendants d'une distance égale à 2 mois à partir de l'année 2007 :*

```
SELECT
  {
  ([Measures].[Unit Sales])
  }
ON COLUMNS,
  {
     Descendants([Time].[1997],2)
     --L'appel ci-dessous donne le même résultat
     --Descendants([Time],[Time].[Month])
  }
ON ROWS
FROM [Sales]
```


Calcul cumulatif (3)

- **Ex 2 :** *calcul des descendants du niveau mois du membre dont la valeur est le 2ième semestre :*

```
SELECT
  {
  ([Measures].[Unit Sales])
  }
ON COLUMNS,
  {
     Descendants([Time].[1997].[Q2],[Time].[Month])
  }
ON ROWS
FROM [Sales]
```

Calcul cumulatif (4)

La fonction **Descendants** peut avoir un 3ième paramètre optionnel, appelé **drapeau** :

Drapeau	Description
Self	Renvoie les membres descendants à partir d'un niveau donné ou à une distance donnée. Le membre donné en entrée est renvoyé si le niveau donné est le niveau de ce membre.
After	Renvoie les membres descendants de tous les niveaux subordonnés d'un niveau ou à une distance donnée.
Before	Renvoie les membres descendants à partir de tous les niveaux entre un membre et un niveau donné, ou à une distance donnée. Le membre donné en entrée est renvoyé mais pas les membres à partir niveau ou de la distance donnée.
Before_And_After	Renvoie les descendants membres à partir de tous les niveaux subordonnés d'un niveau d'un membre donné. Le membre donné en entrée est renvoyé mais pas les membres à partir niveau ou de la distance donnée.
Self_And_After	Renvoie les descendants membres à partir d'un niveau ou à une distance donnée, et à partir tous les niveaux subordonnés du niveau ou de la distance donnée.
Self_And_Before	Renvoie les descendants membres à partir d'un niveau ou à une distance donnée, et à partir de tous les niveaux entre le membre et le niveau donné ou le distance donnée, incluant le membre donné en entrée.
Self_Before_After	Renvoie les descendants membres à partir des niveaux subordonnés d'un niveau d'un membre donné. Le membre donné en entrée est également renvoyé.
Leaves	Renvoie les descendants feuilles entre un membre et un niveau donné ou à une d'une distance donnée.

Calcul cumulatif (5)

- Requête renvoyant tous les membres dont le niveau est le descendant hiérarchique du niveau 1 (trimestre) de la dimension Time :

```
SELECT
    {
    ([Measures].[Unit Sales])
    }
ON COLUMNS,
    {
        Descendants([Time],1, After)
    }
ON ROWS
FROM [Sales]
```

- Requête donnant tous les membres dont le niveau est le supérieur hiérarchique du niveau 2 (mois) de la dimension Time :

```
SELECT
    {
    ([Measures].[Unit Sales])
    }
ON COLUMNS,
    {
        Descendants([Time],2, Before)
    }
ON ROWS
FROM [Sales]
```


Calcul cumulatif (6)

- Requête renvoyant tous les membres du niveau 1 (trimestre) et ceux des niveaux qui les suivent hiérarchiquement (mois) :

```
SELECT
    {
    ([Measures].[Unit Sales])
    }
ON COLUMNS,
    {
        Descendants([Time],1, Self_and_After)
    }
ON ROWS
FROM [Sales]
```


Calcul cumulatif (7)

- **Ex** : *Calcul du cumul de "[Unit Sales]" par intervalle d'un mois de l'année 1997 :*

```
WITH
Member [Measures].[Accumulated Unit Sales]
as
    Sum(YTD(), [Measures].[Unit Sales])
    -- ici comme on ne spécifie pas de paramètre pour YTD, il
utilisera [Time].currentMember
SELECT
    {
    ([Measures].[Unit Sales]), ([Measures].[Accumulated Unit
Sales])
    }
ON COLUMNS,
{
Descendants([Time],[Time].[1997].[Q4].[12])
}
ON ROWS
FROM [Sales]
```

- Si on voulait avoir le cumul par intervalle d'un trimestre, il suffit de reculer d'un niveau les descendants de la dimension "[Time]"

Expressions conditionnelles (1)

• Elles permettent de réaliser des **tests conditionnels** à l'aide du mot clé **IIF**.

• Dans l'exemple qui suit, on construit plusieurs membres dont :

- **[Measures].[Q-1]** : donne [Unit Sales] au trimestre précédent
- **[Measures].[Evolution]** : donne l'évolution de [Unit Sales] entre deux trimestres consécutifs
- **[Measures].[%]** : donne l'évolution en pourcentage
- **[Measures].[Performance]** : renvoie une chaîne de caractères qui nous permet d'appliquer des règles de gestion liées à des conditions

Expressions conditionnelles (2)

```
WITH
MEMBER [Measures].[Q-1] AS
( ParallelPeriod([Time].[1997].[Q1],1, [Time].CurrentMember), [Measures].[Unit Sales] )
MEMBER [Measures].[Evolution]
AS
( ([Time].CurrentMember, [Measures].[Unit Sales]) - (ParallelPeriod([Time].[1997].[Q1],1,
[Time].CurrentMember), [Measures].[Unit Sales]) ), solve_order = 1
MEMBER [Measures].[%] AS
( [Measures].[Evolution] /  (ParallelPeriod([Time].[1997].[Q1],1,[Time].CurrentMember),
[Measures].[Unit Sales]) ), format_string = "Percent", solve_order = 0
MEMBER [Measures].[Performance] AS
IIF([Measures].[Q-1]<>0,
   IIF([Measures].[%] < 0, "-",
      IIF([Measures].[%] > 0 and [Measures].[%] < 0.01, "+",
      IIF([Measures].[%] > 0.01 and  [Measures].[%] < 0.05, "++", "better performance") )),"null")
SELECT
  { ([Measures].[Unit Sales]),
    ([Measures].[Q-1]),
    ([Measures].[Evolution]),
    ([Measures].[%]),
    ([Measures].[Performance]) } ON COLUMNS,
    Descendants([Time], 1) ON ROWS
FROM [Sales]
```


Résumé sur MDX (1)

Instruction simple MDX :

```
SELECT axis [,axis]
FROM cube
WHERE slicer [,slicer]
```

La clause SELECT est utilisée pour définir les dimensions des axes (axis) et la clause WHERE pour spécifier les dimension de filtrage (slicer), et CUBE est le nom du cube considéré

Les dimensions des axes dans la clause SELECT
Si le cube est vu comme une structure à n dimensions, cette clause spécifie les arêtes du cube a retourner. La structure de base de la clause SELECT est :

```
SELECT set ON axis_name,
 set ON axis_name,
 set ON axis_name
```

où axis_name peut être COLUMNS ou ROWS

Les filtres (Slicers) dans la clause WHERE
Les dimensions de filtrage contiennent les seuls membres avec lesquels le cube est filtré ou « sliced »

Résumé sur MDX (2)

Le contenu des axes :

Un **membre** (Member) = est un item dans une dimension et correspond à une élément spécifique de donnée, Ex. :

[Time].[1997]

[Customers].[All Customers].[Mexico].[Mexico[

[Product].[All Products].[Drink]

un **tuple** = une collection de membres de différentes dimensions, Ex. :

([Time].[1997], [Product].[All Products].[Drink])

(1997, Drink)

(1997, [Customers].[All Customers].[Mexico].[Mexico])

Un **set** = une collection de tuples, Ex. :

{[Time].[1997], [Time].[1998], [Time].[1999]}

{1997, 1998, 1999}

{1990:1999} (The colon(☺ is an inclusive range.)

{(1997, Drink), (1998, Drink)}

Résumé sur MDX (3)

La hiérarchie des données :

Les données au sein d'un cube sont organisées avec les relations suivantes :

Dimensions
Hierarchies
Levels
Members

Ex. :

Product	*Dimension*
Function	*Hierarchy*
Product Family	*Level*
Drink	*Member*
Food	*Member*
Non-Consumable	*Member*

Résumé sur MDX (3)

MDX Functions and Expressions:

Function refers to a specific operation being performed on some set of data (Sum(), TopCount()). Expression describe syntax in which the function is placed after the cube parameter ([1997].children, [Products].DefaultMember).

The .Members Expression

This expression is used to retrieve a set of enumerated members from a dimension, hierarchy, or level. For example:

dimension.Members
hierarchy.Members
level.Members

The CrossJoin() Function:

The CrossJoin() function is used to generate the cross-product of two input sets. If 2 sets exists in 2 independent dimensions, the CrossJoin operator creates a new set consisting of all of the combinations of the members in the two dimensions as follows: crossjoin (set1, set2)

The TopCount() and BottomCount() Functions:

These expressions sort a set based on a numerical expression and pick the top index items based on rank order as follows:

TopCount (set, index, numeric expression)
BottomCount (set, index, numeric expression)

The Filter() Function

The Filter() function is used to filter a set based on a particular condition as follows:

Filter (set, search condition)

The Order() Function

The Order() function provides sorting capabilities within the MDX language as follows:

Order (set, string expression [, ASC | DESC | BASC | BDESC]) or
Order (set, numeric expression [, ASC | DESC | BASC | BDESC])