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

(7.2)

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

Décembre 2015

- **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

Bibliographie et sources du cours

Ouvrages :

- Vaisman A., Zimányi E., « Data Warehouse Systems: Design and Implementation », Springer-Verlag, 2014, ISBN 978-3-642-54654-9.
- M. Whitehorn, R. Zare, M. Pasumansky, « Fast track to MDX », Springer, 2006.
- Golfarelli M., Rizzi S., « Data Warehouse Design : Modern Principles and Methodologies », McGrawHill, 2009.
- …

Cours :

- Cours de K. Aouiche, Université de Lyon 2, 2009.
- Cours de A. Vaisman A., E. Zimányi, Université Libre de Bruxelles, 2014.
- Cours de P. Marcel, Université de Tours, 2012.
- Cours de M. Herschel, Herschel, Université Paris Sud, 2012.
- …

1 – Introduction à MDX

- **Origine de MDX**
- **MDX versus SQL**
- **Un exemple d'entrepôt**
- **Rappel des catégories d'opérations OLAP**

Origine de MDX

- **MDX**, acronyme de **M**ulti **D**imensional e**X**pression, est un **langage de requêtes OLAP** pour les bases de données multidimensionnelles
- inventé en 1997 par **Mosha Pasumansky** au sein de Microsoft, version commerciale Microsoft OLAP Services 7.0 & Analysis Services en1998, dernière spécification OLE DB for OLAP (ODBO) en 1999
- **MDX est fait pour** naviguer dans les bases multidimensionnelles, et définir des requêtes sur tous leurs objets (*dimensions, hiérarchies, niveaux, membres et cellules*)
- Une requête MDX retourne un **rapport à plusieurs dimensions** consistant en **un ou plusieurs tableaux 2D**
- Utilisé par de **nombreux outils de BI** commerciaux ou non
- Langage très **complexe** et **puissant** générant des **requêtes plus compacte** que les **requêtes SQL équivalentes**

MDX versus SQL (1)

- La **syntaxe** de MDX ressemble à celle de SQL par ses **mots clé SELECT**, **FROM**, **WHERE,** mais leurs **sémantiques** sont différentes :
  - 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
  - **MDX :** **SELECT** axis1 **ON COLUMNS,** axis2 **ON ROWS FROM** cube
- **Clause FROM** spécifie la **source** de données :
  - **en SQL :** une ou plusieurs tables
  - **en MDX :** un cube

MDX versus SQL (2)

- **La clause 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

MDX query example:

Soit la requête MDX suivante : *(Q=Quarter)*

```
SELECT  {Paris, Berlin} ON ROWS
        {[Q1], [Q2].CHILDREN} ON COLUMNS
FROM    CubeSales
WHERE   (MEASURES.SalesAmount,
         Time.[2014],
         Product.Product)
```

Agrégation de la mesure « SaleAmount » avec la fonction SUM function

Sélection de la dimension Time (2014 seulement)

Sélection de la dimension Product (all product)

Résultat :

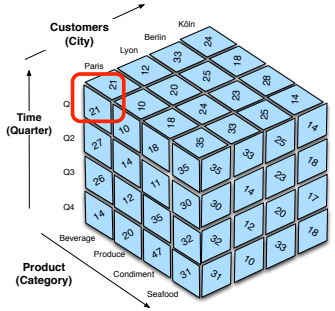
	Q1 2014	April 2014	May 2014	June 2014
Paris	12,567	3,360	5,450	4,570
Berlin	12,567	3,360	5,450	4,570
…	…	…	SalesAmount values	…

Simplified Data Warehouse:

- **Measures** : ***Unit Price, Quantity, Discount, SalesAmount, Freight***
- **Dimension** : **Time**
  - **hierarchy** : *Year > Quarter > Month >* with **members** :
    - Year: *2010, 2011, 2012, 2013, 2014*
    - Quarter: *Q1, Q2, Q3, Q4*
    - Month: *January, February, March, …*
- **Dimension** : **Customer**
  - **hierarchy** : *Continent > Country > State > City* with **members** :
    - City: *Paris, Lyon, Berlin, Köln, Marseille, Nantes …*
    - State: *Loire atlantique, Bouches du Rhône, Bas Rhin, Torino…*
    - Country: *Austria, Belgium, Danmark, France, ...*
    - Continent level: *Europe, North America, Sud America, Asia*
- **Dimension** : **Product**
  - **hierarchy** : *Category > Subcategory > product* with **members** :
    - Category : *Food, Drink …*
    - Food category: *Baked_food …*
    - …

Un exemple de cube « Sale »

- Ce cube « sale » a : *(inspired from Vaisman & Zimányi)*
  - **3 dimensions** : Time, Customer, Product
  - **1 measure** : SalesAmount



La cellule en haut à gauche a la valeur 21 et correspond à la vente de *beverage* à Paris durant le premier *trimestre* (Q1) :
(Product.Category.Beverage ; Time.Quarter.Q1 ; Customer.City.Paris)

Rappel des catégories d'opérations OLAP

3 catégories d'opérations élémentaires :

Restructuration : concerne la représentation, permet un changement de points de vue selon différentes dimensions : opérations liées à la structure, manipulation et visualisation du cube :

- Rotate/pivot
- **Switch**
- **Split, nest, push, pull**

Granularité : concerne un changement de niveau de détail : opérations liées au niveau de granularité des données :

- **roll-up,**
- **drill-down**

Ensembliste : concerne l'extraction et l'OLTP classique :

- **slice, dice**
- selection, projection et jointure (drill-across)

2 – 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 (1)

- **Syntaxe générale d'une requête MDX (forme de Backus-Naur):**

  SELECT [<specification d'un axe>
  [, <spécification d'un axe>...]]
  FROM [<spécification d'un cube>]
  [WHERE [<spécification d'un filtre (slicer)>]]

- **Parenthèses en MDX :**

{ } : Ensemble des éléments servant à la création d'une dimension du résultat de la requête

( ) : Sélection de tuples dans la clause WHERE

[ ] : Représentation d'espaces, de caractères spéciaux et d'interprétation non numérique de chiffres.

REMARQUE: Dans MDX les **[ ]** sont optionnels, excepté pour un nom avec des caractères « espace » , avec des chiffres, ou qui sont des mots-clés MDX, quand ils sont requis.

Structure générale d'une requête MDX (2)

SELECT - description des axes du cube résultat

- **Chaque dimension du résultat :**
  - est associée à un **rôle** correspondant à sa représentation dans le tableau retourné par la requête MDX :

    Ex : ON COLUMNS, ON ROWS, ON PAGES, ON SECTIONS, ON CHAPTERS

  - sur un ou plusieurs **niveaux** de la hiérarchie :

    Ex1 : {Paris, Berlin} de la dimension Lieu, niveau Ville

    Ex2 : {[1er trimestre], [2nd trimestre].CHILDREN} de la dimension Temps, niveaux trimestre et mois.

Structure générale d'une requête MDX (3)

FROM - Spécification du/des cube/s de départ

- Ensemble de cubes nécessaires à la création du cube résultat
- Si plusieurs cubes nécessaires, cela implique une **jointure multidimensionnelle** : chaque paire de cubes doit alors posséder au moins une dimension concordante.

WHERE - Restriction sur le/s cube/s de départ

- **Restrictions sur le/s** cube/s de départ de la clause FROM.
- Spécification des restrictions par une **liste de noeuds de la hiérarchie d'une dimension** nommée « slicer-dimension »

.

REMARQUE : En MDX les **mesures** sont des éléments d'une **dimension spéciale nommée « *Measures* »** (ces mesures peuvent être utilisées aussi dans les clauses WHERE et FROM).

Membres en MDX

- Un **membre** = une **instance** d'un **niveau** d'une **dimension**,
- Est généralement spécifié entre **crochets** [...]

  Ex : [Food], [Drink] = membres de la dimension "Products" de niveau 1
- Les **membres** = items accessibles dans les hiérarchies pouvant être référencés de différentes façons :

  [2012]
  [Time].[2012]
  [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].[2012] ON COLUMNS FROM [Sales]
SELECT [Product].[Food] ON COLUMNS FROM [Sales]
SELECT [Product].[Food].[Baked Goods] ON COLUMNS FROM [Sales]
```


Tuples, cellule et mesure

- Un **tuple** = suite de **membres** entre **parenthèses** séparés par une **virgule**:

  Ex : ([Time].[2012] , [Product].[Food])

  on peut omettre les parenthèses si on a un tuple avec un seul membre.
- Un **tuple** permet d'identifier une ou plusieurs **cellules** dans un cube situées à l'intersection de ses membres :

```
SELECT ([Time].[2012], [Product].[Food]) ON COLUMNS
FROM [Sales]

SELECT ([Product].[All Products].[Food].[Baked Goods], [2012]) 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 définit sur une même dimension**
- Un set **commence** par une accolade "**{**", dans laquelle sont énumérés les **tuples** séparés par des **virgules**, et se termine par une accolade appariée "**}**"

```
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 membres contigus différents** ([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** ([2012]) :

```
SELECT
{ ([2012]) } ON COLUMNS    ou  { [2012] } 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 en 2012 et 2013 pour les produits "Drink" et "Food" :*

```
SELECT
{  ([Measures].[Unit Sales], [Product].[Food]),
     ([Measures].[Unit Sales], [Product].[Drink]) } ON AXIS(0),
{ ([Time].[2012]), ([Time].[2013]) } ON AXIS(1)
FROM [Sales]
```

ou

```
SELECT
   { ([Measures].[Unit Sales], [Product].[Food]),
      ([Measures].[Unit Sales], [Product].[Drink]) } ON COLUMNS,
   { ([Time].[2012]), ([Time].[2013]) } 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** 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
```


Spécification d'un axe en MDX (3)

Cette requête MDX :

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

Son résultat est une table avec 2 axes :

	2010	*2011*	*2013*	*2014*
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

- l'axe **horizontal** est (i.e. COLUMNS) contient tous les membres de la **dimension** '**Years**' *(ici 2010, 2011, 2013, 2014)*
- l'axe **vertical** est (i.e. ROWS) contient **tous les membres** du **niveau** '**Continent**' de la **dimension** '**Regions**' *(ici N & S America, Europe, Asia)*

REMARQUE : ici 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 dimension ou du niveau**,** aussi dans MDX on peut **spécifier une liste de membres à considérer** : { dim.member1, dim.member2, … , dim.membern}

Ex *: On considére seulement les ventes sur les 2 années 2012 et 2013:*

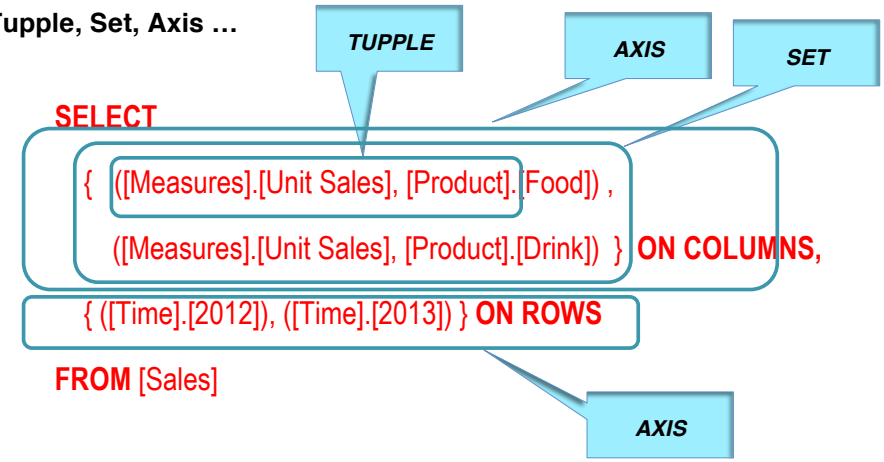
```
SELECT
  { Years.[2012], Years.[2013] }  ON COLUMNS,
    Regions.Continent.MEMBERS    ON ROWS
FROM Sales
```

Remarques :

- il est recommandé de choisir des **noms de membres** entre crochet **[ ]** avec aucun symbole blanc, point, …
- ici [2012] et [2013] sont des **membres** et pas des valeurs !!!
- ici la **mesure** (measure) pas explicitement définie => mesure par défaut
- quand une dimension a le niveau '**All**', le **premier membre** du **premier niveau** sera choisi par **défaut**

Récapitulatif :

Tupple, Set, Axis …



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.[2012], Years.[2013] } 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: Slice sur un produit (computer) et un client particulier (AT&T) : Ventes de computer à AT&T pour les années 2012 et 2013 pour tous les continents.

```
SELECT
  { Years.[2012], Years.[2013] }  ON COLUMNS,
  Regions.Continent.MEMBERS ON ROWS
FROM Sales
WHERE
(
   Products.[Product Group].[Computers],
   Customers.[AT&T]
)
```


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** :
- *Ex : Unités de computer expédiées à AT&T pour les années 2012 et 2013 pour tous les continents.*

```
SELECT
    { Years.[2012], Years.[2013] } ON COLUMNS,
   Regions.Continent.MEMBERS ON ROWS
 FROM Sales
WHERE
(
   Products.[Product Group].[Computers],
   Customers.[AT&T],
   Measures.[Units]
)
```

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 */
*/
```


3 – 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**
- *Ex : On veut voir les **ventes** de **computers** et **printers** en **Europe** et en **Asie** sur les années **2012** et **2013**.*
- On peut le faire avec une requête MDX avec **3 axes** :

```
SELECT
   { Continent.[Europe], Continent.[Asia] }       ON AXIS(0),
   { Product.[Computers], Product.[Printers] }    ON AXIS(1),
   { Years.[2012], Years.[2013] }                 ON AXIS(2)
FROM Sales
```

- 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 les 3 dimensions, **avec seulement 2 axes** avec une dimension emboitée dans une autre

Emboitement (Nest) de tuples dans MDX (2)

- Les lignes 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** de **dimensions différentes** est un **Tuple**

***Ex** : Ventes de computers et printers pour l'Europe et l'Asie pour 2012 et 2013:*

```
SELECT
  { Year.[2012], Year.[2013] } ON COLUMNS,
  { ( Continent.Europe, Product.Computers ),
    ( Continent.Europe, Product.Printers  ),
    ( Continent.Asia, Product.Computers )
    ( Continent.Asia, Product.Printers  ) } ON ROWS
FROM Sales
```

		2012	2013
Europe	Computers		
	Printers		
Asia	Computers		
	Printers		

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**
- La syntaxe pour les **membres calculés** est de mettre la construction suivante **WITH** en face de l'instruction SELECT :

```
WITH MEMBER parent.name AS 'expression'
```

***Ex** : Trouver les profits des produits au cours des années*

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

- Le parent du nouveau membre est de **dimension mesure** (measures), le nom est **Profit** et l'expression est : Profit = Sales – Cost :

Membres calculés dans MDX (2)

- Les membres calculés sont traités de la **même manière** que des membres ordinaires, ils peuvent donc être utilisés :
  - **dans la définition d'axes**
  - **dans une clause WHERE**
- On peut définir des membres calculés avec d'autres membres calculés

***Ex** : 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)

Ex : Supposons que l'on veuille comparer la compagnie entre 2012 et 2013, on peut :

- construire une requête qui a un axe {[2012], [2013]} et regarder les paires de nombres pour chaque mesure
- **définir un membre calculé** dans le niveau *Year*, parallèle à 2012 et 2013, qui contiendra la différence entre eux :

```
WITH MEMBER Time.[12 to 13] AS 'Time.[2012] – Time.[2013]'
SELECT
  { Time.[12 to 13] } ON COLUMNS,
  Measures.MEMBERS ON ROWS
FROM Sales
```


Membres calculés dans MDX (4)

Supposons qu'on veuille voir comment les profits ont évolués entre 2012 et 2013:

```
WITH
   MEMBER Measures.Profit AS 'Measures.Sales – Measures.Cost'
   MEMBER Time.[12 to 13] AS 'Time.[2012] – Time.[2013]'
SELECT
   { Measures.Sales, Measures.Cost, Measures.Profit } ON COLUMNS,
   { Time.[2012], Time.[2013], Time.[12 to 13] } ON ROWS
FROM Sales
```

On obtient la table :

	Sales	*Cost*	*Profit*
2012	300	220	80
2013	350	210	140
12 to 13	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. Quand la cellule contient un de ses coordonnés un membre NULL, la valeur numérique de la cellule sera zero ainsi : (Sales, [January].PrevMember) = 0
  2. Toute fonction de membre appliquée à un membre NUL retourne NULL
  3. Quand un membre NULL member est inclus dans un set, il est juste ignoré, 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.

4 – Fonctions sur les *Membres* et les *Sets* dans MDX

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

Fonctions de navigation sur les membres des 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 région 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 navigation sur les membres des dimensions (2)

Fonction	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] et 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 de navigation sur les membres des dimensions (3)

*Ex 1 : 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
```

Ex 2 : On veut observer 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
```


Rappel : 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 { [2012], [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 { ([2012], [USA]), ([August], [WA]) } est **valide**,
    - le set { ([August], [WA]), ([2012], [USA]) } n'est ***pas valide***, car l'ordre des dimensions dans le tuple est inversé.

Opérations sur les Sets dans MDX : Intersect et Union

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

Intersection : **INTERSECT(set1, set2)** (Rarement utilisée)

Union : **UNION(set1, set2)**

Permet de combiner 2 ensembles ou plus.

Expressions MDX plus concises :

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

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

- On défini alors alors 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

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 : Topcount, Tail, Head, Filter

- Exemple d'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].[2012].[Q1]), ([Time].[2012].[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].[2012].[Q1]), ([Time].[ 2012].[Q2])},
      {([Measures].[Unit Sales]), ([Measures].[Store Sales])}  ) } ON COLUMNS,
    { ([Product].[Drink].Children),
    ([Product].[Food].[Baked Goods]) } ON ROWS
FROM [Sales]
```

- **Ex 3 :**

```
SELECT
     Crossjoin
     ( {[Measures].[Unit Sales], [Measures].[Store Sales]},
     {[Time].[ 2012].[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 (3)

- 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].[ 2012].[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.

5 – Expressions avancées de MDX

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

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 sur le trimestre antérieur :*

```
WITH MEMBER [Measures].[Unit Sales Q-1] AS (  ParallelPeriod( [Time].[ 2012].[Q1], 1)  )
SELECT
      { ([Measures].[Unit Sales]), ([Measures].[Unit Sales Q-1]) } ON COLUMNS,
      {  ([Time].Children)  } ON ROWS
FROM [Sales]
```


Calcul cumulatif

Pour calculer des cumuls sur une période de temps

Ex: calcul du cumul de la mesure "[Unit Sales]" durant l'année 2012.

- Dans un premier temps, on doit retrouver tous les mois de l'année 2012
- 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].[ 2012].[Q1].Children),
        ([Time].[ 2012].[Q2].Children),
        ([Time].[ 2012].[Q3].Children),
        ([Time].[ 2012].[Q4].Children) }  ON ROWS
FROM [Sales]
```

Calcul cumulatif : Descendants

- Plus élégant et moins fastidieux : utilisation de la fonction **Descendants** :
  - 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 2012:*

```
SELECT
  { ([Measures].[Unit Sales]) } ON COLUMNS,
  { Descendants([Time].[2012],2)
    -- L'appel ci-dessous donne le même résultat
    -- Descendants([Time],[Time].[Month])  } ON ROWS
FROM [Sales]
```

- **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].[2012].[Q2],[Time].[Month]) }  ON ROWS
FROM [Sales]
```


Calcul cumulatif : Descendant (1)

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 : Descendant (2)

- 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 : Descendant (3)

- 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]
```

- ***Ex*** *: Calcul du cumul de "[Unit Sales]" par intervalle d'un mois de l'année 2012:*

```
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].[2012].[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 : IIF (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 : IIF (2)

```
WITH
   MEMBER [Measures].[Q-1] AS
  ( ParallelPeriod([Time].[2012].[Q1],1, [Time].CurrentMember), [Measures].[Unit Sales] )
   MEMBER [Measures].[Evolution]  AS
   ( ([Time].CurrentMember, [Measures].[Unit Sales]) - (ParallelPeriod([Time].[2012].[Q1],1,
    [Time].CurrentMember), [Measures].[Unit Sales]) ), solve_order = 1
   MEMBER [Measures].[%] AS
   ( [Measures].[Evolution] /  (ParallelPeriod([Time].[2012].[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]
```


6 – Résumé des commandes MDX

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 un élément spécifique de donnée :

  [Time].[2012]
  [Customers].[All Customers].[Mexico].[Mexico]
  [Product].[All Products].[Drink]

- Un **tuple** = une **collection de membres de différentes dimensions** :

  ([Time].[2012], [Product].[All Products].[Drink])
  (2012, Drink)
  (2012, [Customers].[All Customers].[Mexico].[Mexico])

- Un **set** = **une collection de tuples** :

  {[Time].[2012], [Time].[2013], [Time].[2014]}
  {2012, 2013, 2014}
  {(2012, Drink), (2013, Drink)}

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 ([2012].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])