# L'informatique des entrepôts de données

**Daniel Lemire**

# SEMAINE 11
# Notions intermédiaires de MDX

## 11.1. Présentation de la semaine

Cette semaine, nous poursuivons notre introduction au langage MDX. En particulier, nous allons couvrir les fonctions sur les sets, la fonction CrossJoin, les membres calculés, et la manipulation de la dimension temps.

## 11.2. Redémarrons

Rappel : La personne qui vous encadre n'a pas accès à votre machine. Elle n'a pas accès à ce que vous avez fait. En cas de problème, faites des saisies d'écran complètes montrant la totalité de vos manipulations techniques depuis le début de la semaine 10 et transmettez-les à la

personne chargée de votre encadrement. Elle pourra vérifier avec vous que vous avez suivi toutes les étapes, point par point.

La semaine passée nous avons installé Tomcat et Mondrian. Il faut maintenant relancer Mondrian :

- Retrouvez le dossier dans lequel vous avez placé Tomcat (par exemple, sur votre bureau).
- Si vous utilisez Windows, exécutez le script `bin\startup.bat`. Si vous utilisez Linux ou Mac OS, exécutez le script `bin/startup.sh`.
- Vous devez vérifier que Tomcat fonctionne. Visitez le site `http://localhost:8080/`. Vous devez voir la page d'accueil de Tomcat. Rappelez-vous qu'en cas de problème, vous pouvez consulter le fichier logs/catalina.out pour consulter les messages d'erreur de Tomcat.
- Rendez-vous à l'adresse `http://localhost:8080/mondrian-embedded/`. Vous devez voir la page d'accueil de Tomcat de Mondrian.
- Suivez le lien Basic interface for ad hoc queries. Vous êtes maintenant prêt!

## 11.3. Approche pédagogique

Nous suivons la même approche qu'à la semaine précédente. Vous devez recopier les requêtes MDX et les exécuter. N'oubliez pas que vous pouvez aussi créer vos propres requêtes! Rappelez-vous qu'il n'est pas nécessaire de mémoriser toutes les fonctions MDX (ce serait difficile).

## 11.4. Une référence incontournable

Il y a beaucoup de fonctions MDX. La semaine passée vous avez pu consulter certaines références sur le web. Cette semaine, je vous invite à consulter le guide de référence de Mondrian : `http://mondrian.pentaho.com/documentation/mdx.php`. Vous y trouverez une liste exhaustive des fonctions supportées par Mondrian.

## 11.5. Fonctions sur les membres et les dimensions

Jusqu'à présent, nous avons supposé que nous étions disposé à saisir les différentes valeurs des dimensions. Par exemple, la dimension Product comporte des membres Food et Drink. Mais se rappeler de toutes ces valeurs peut devenir fastidieux.

La fonction Members permet d'obtenir toutes les valeurs (de tous les niveaux hiérarchiques) :

```
select
{[Measures].[Unit Sales]} on columns,
{([Product].Members)} on rows
 from [Sales]
```

La fonction s'applique à toutes les dimensions :

```
select
{[Measures].[Unit Sales]} on columns,
{([Time].Members)} on rows
 from [Sales]
```

Comme on peut le constater, la liste des valeurs peut être longue ! On peut aussi cibler des éléments qui sont à un niveau précis de la hiérarchie avec la fonction Levels :

```
select
{[Measures].[Unit Sales]} on columns,
{([Product].Levels(2).Members)} on rows
 from [Sales]
```

Voici un autre exemple :

```
  select {
  ([Time].Children)
} on columns,
{ ([Store].Levels(1).Members) }
on rows
from [Sales]
```

On peut aussi se contenter de faire les listes des enfants, c'est-à-dire des valeurs qui sont immédiatement en dessous dans la hiérarchie avec la fonction Children :

```
select{
  ([Time].Children)
} on columns,
{([Gender].Members) }
on rows
from [Sales]
```

Notez ici que [Time] a comme valeur par défaut l'année 1997 (cela a été configuré dans la base de données) et que la requête précédente est en fait équivalente à celle-ci :

```
select{
  ([Time].[1997].Children)
} on columns,
{([Gender].Members) }
on rows
from [Sales]
```

Évidemment, la fonction Children est applicable à toutes les dimensions :

```
select
{[Measures].[Unit Sales]} on columns,
{([Product].[Non-Consumable].Children)} on rows
 from [Sales]
```

Comme on peut le voir, la fonction Children nous permet de faire un drill-down.

Il existe une multitude de fonctions similaires en MDX dont Parent, NextMember, PrevMember, FirstSibling, LastSibling, FirstChild, LastChild, etc. Je vous invite à tester ces fonctions avec les requêtes suivantes :

(a)

```
select{
  ([Time].[1997].[Q1].Parent)
} on columns,
{([Gender].Members) }
on rows
from [Sales]
```

```
select{
  ([Time].[1997].[Q1].NextMember)
} on columns,
{([Gender].Members) }
on rows
from [Sales]
```

(b)

```
select{
  ([Time].[1997].[Q2].PrevMember)
} on columns,
{([Gender].Members) }
on rows
from [Sales]
```

```
select{
  ([Time].[1997].[Q3].FirstSibling)
} on columns,
{([Gender].Members) }
on rows
from [Sales]
```

(d)

```
select{
  ([Time].[1997].[Q3].LastSibling)
} on columns,
{([Gender].Members) }
on rows
from [Sales]
```

```
select
{[Measures].[Unit Sales]} on columns,
{([Product].FirstChild)} on rows
 from [Sales]
```

(g)

```
select
{[Measures].[Unit Sales]} on columns,
{([Product].LastChild)} on rows
 from [Sales]
```

```
select
{[Measures].[Unit Sales]} on columns,
Descendants([Product].[Non-Consumable])
on rows
 from [Sales]
```

## 11.6. Fonctions sur les sets

La Table 1 donne une liste de fonctions utiles pour manipuler les sets. Encore une fois, pour mieux comprendre le sens de ces fonctions, nous vous proposons quelques exemples.

(h) On ne veut que les deux premiers trimestres :

```
select
{
  ([Measures].[Unit Sales])
}
on columns,
{
```

**Table 1.** Quelques fonctions MDX sur les sets

| Fonction | Syntaxe | Description |
|---|---|---|
| Head | Head( Set [, expression numérique ]) | Elements de tête d'un set |
| Tail | Tail( Set [, expression numérique ]) | Derniers éléments d'un set |
| Subset | Subset( Set , Start [, Count ]) | Sous-ensemble d'éléments d'un set |
| TopCount | TopCount( Set , Count [, expression numérique ]) | Les premiers éléments ayant la plus grande valeur de la mesure |
| Order | Order ( Set , String Expression \| expression numérique [, ASC \| DESC \| BASC \| BDESC]) | Tri des éléments d'un set |
| Filter | Filter( Set , conditions ) | Les éléments d'un set qui satisfont le filtre |
| Distinct | Distinct( Set ) | Élimine les duplicats |

```
    (Head([Time].Children, 2))
}
on rows
from [Sales]
```

- On ne veut que les trois derniers trimestres :

```
select
{
  ([Measures].[Unit Sales])
}
on columns,
{
    (Tail([Time].Children, 3))
}
on rows
from [Sales]
```

- On ne veut que le trimestre avec les meilleures ventes :

```
select
{
  ([Measures].[Unit Sales])
}
on columns,
{
    (Topcount([Time].Children,1,[Measures].[Unit
       Sales]))
}
on rows
```

```
from [Sales]
```

Ces requêtes illustrent l'utilisation des fonctions de navigation hiérarchique en MDX pour explorer les relations entre membres d'une dimension, comme `[Time]` ou `[Product]`, en affichant des valeurs associées aux mesures `[Unit Sales]` ou aux membres de `[Gender]`. La première récupère le parent de `[1997].[Q1]` (l'année 1997) sur les colonnes pour toutes les entrées de genre ; la deuxième avance au membre suivant (`[Q2]`) ; la troisième recule au précédent (`[Q1]`) depuis `[Q2]` ; les quatrième et cinquième identifient respectivement le premier et dernier frère de `[Q3]` (soit `[Q1]` et `[Q4]`) dans l'année ; la sixième et la septième sélectionnent le premier et dernier enfant de la dimension `[Product]` (les catégories de base comme `[Drink]` ou `[Food]`) ; enfin, la huitième utilise `Descendants` pour lister tous les descendants du membre `[Non-Consumable]` dans `[Product]`, produisant une vue hiérarchique exhaustive des produits non consommables avec leurs ventes unitaires, facilitant ainsi l'exploration interactive des structures multidimensionnelles sans requêtes manuelles complexes.

On peut combiner différentes listes :

```
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]
```

Cette requête démontre la combinaison d'ensembles distincts sur les lignes via une union implicite dans MDX, en appliquant la fonction `Head` pour sélectionner les deux premiers enfants de la dimension `[Time]` (typiquement les premiers trimestres), `Tail` pour les trois derniers (les trimestres finaux), et `TopCount` pour identifier le trimestre unique avec les ventes unitaires les plus élevées parmi ces enfants, le tout opposé à la mesure `[Unit Sales]` sur les colonnes et extrait de la base `[Sales]`. Cette approche permet de construire un axe rows hétérogène et ciblé, idéal pour juxtaposer des vues partielles ou classées sans énumération

exhaustive, favorisant une analyse sélective des performances temporelles dominantes ou chronologiques.

On voit dans cet exemple que certaines valeurs sont répétées. Comme en SQL, on peut utiliser l'instruction distinct pour éviter que cela soit le cas :

```
select
{
  ([Measures].[Unit Sales])
}
on columns,
distinct( {
     (Head([Time].Children, 2)),
    (Tail([Time].Children, 3)),
    (Topcount([Time].Children,1,[Measures].[Unit
       Sales]))
})
```

On peut ensuite trier les valeurs de telle manière que les mesures soient en ordre décroissant :

```
select
{
  ([Measures].[Unit Sales])
}
on columns,
order(distinct( {
     (Head([Time].Children, 2)),
    (Tail([Time].Children, 3)),
    (Topcount([Time].Children,1,[Measures].[Unit
       Sales]))
}),[Measures].[Unit Sales],DESC)
on rows
from [Sales]
```

On peut aussi filter les valeurs de telle manière que seules les mesures les plus significatives soient affichées, selon un seuil fixe (il s'agit d'une forme de requête iceberg) :

```
select
{
  ([Measures].[Unit Sales])
}
on columns,
filter([Time].Children,[Measures].[Unit
   Sales]>65000)
```

```
on rows
from [Sales]
```

Cette requête utilise la fonction `FILTER` appliquée aux enfants de la dimension `[Time]` pour ne retenir sur les lignes que les périodes où les ventes unitaires `[Unit Sales]` excèdent le seuil fixe de 65 000, implémentant ainsi une technique de requête iceberg qui priorise les données les plus pertinentes et évite l'encombrement visuel. Les colonnes se concentrent exclusivement sur cette mesure, tandis que la source de données reste la base `[Sales]`, favorisant une analyse concise des performances temporelles dominantes sans nécessiter de post-traitement externe.

## 11.7. Fonction CrossJoin

Pour réaliser des représentations du type tableau croisé, MDX fournit l'opérateur CrossJoin. Cet opérateur permet de croiser des sets. Plus formellement, étant 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 :

```
select
{
  CrossJoin
  (
    {([Time].[1997].[Q1]), ([Time].[1997].[Q2])},
    {([Measures].[Unit Sales]), ([Measures].[Store
       Sales])}
  )
}
on columns,
{
  ([Product].[Drink].Children)
}
on rows
from [Sales]
```

Il arrive qu'une opération CrossJoin génère des cellules vides :

```
select
  Crossjoin
  (
    {[Measures].[Unit Sales], [Measures].[Store
       Sales]},
    {[Time].[1997].[Q1]}
```

```
  )
on columns,
  Crossjoin
  (
    {[Promotion Media].[Cash Register Handout],
    [Promotion Media].[Sunday Paper, Radio, TV],
    [Promotion Media].[TV]},
    {[Gender].Children}
  )
on rows
from [Sales]
```

On peut améliorer le résultat avec l'instruction Non Empty :

```
select
  Non Empty Crossjoin
  (
    {[Measures].[Unit Sales], [Measures].[Store
       Sales]},
    {[Time].[1997].[Q1]}
  )
on columns,
  Non Empty Crossjoin
  (
    {[Promotion Media].[Cash Register Handout],
    [Promotion Media].[Sunday Paper, Radio, TV],
    [Promotion Media].[TV]},
    {[Gender].Children}
  )
on rows
from [Sales]
```

Voici un autre exemple :

```
select
  Crossjoin
  (
    {[Measures].[Unit Sales], [Measures].[Store
       Sales]},
    {[Gender]}
  )
on columns,
  {
    [Promotion Media]
  }
on rows
```

```
from [Sales]
```

Vous ne pouvez pas mettre plus d'une fois la même hiérarchie dans plusieurs axes indépendants d'une requête. En exécutant la requête suivante, vous obtiendrez une erreur :

```
select
  Crossjoin
  (
    {[Measures].[Unit Sales], [Measures].[Store
       Sales]},
    {[Gender]}
  )
on columns,
  {
    [Gender].[F]
  }
on rows
from [Sales]
```

## 11.8. Membres calculés (with)

MDX permet à l'aide du mot-clé `with` de calculer de nouveaux membres.

```
with Member [nom du membre 1] as [<specification>]
Member [nom du membre 2] as [<specification>]
...
select [<axis_specification>
[, <spécification_des_axes>...]]
from [<spécification_d_un_cube>]
[where [<spécification_de_filtres>]]
```

Pour illustrer l'utilisation de `with`, on se propose d'ajouter un nouveau membre calculé [Store Sales by Unit Sales].

```
with
  Member [Measures].[Store Sales by Unit Sales]
as
  (
    [Measures].[Store Sales] / [Measures].[Unit
       Sales]
  )
select
  {
    CrossJoin
    (
```

```
      {([Time].[1997].[Q1])},
      {([Measures].[Unit Sales]),
         ([Measures].[Store Sales]),
      [Measures].[Store Sales by Unit Sales]}
    )
  }
on columns,
  {
    ([Product].[Drink].Children)
  }
on rows
from [Sales]
```

On peut aussi en profiter pour formater les valeurs avec l'instruction FORMAT_STRING :

```
with
  Member [Measures].[Store Sales by Unit Sales]
as
  (
    [Measures].[Store Sales] / [Measures].[Unit
       Sales]
  ),
FORMAT_STRING = "$#,###.00"
select
  {
    CrossJoin
    (
      {([Time].[1997].[Q1])},
      {([Measures].[Unit Sales]),
         ([Measures].[Store Sales]),
      [Measures].[Store Sales by Unit Sales]}
    )
  }
on columns,
  {
    ([Product].[Drink].Children)
  }
on rows
from [Sales]
```

Il est plus naturel de créer de nouveaux membres à partir des mesures. Cependant, vous pouvez également en créer sur n'importe quel autre axe. Prenons, l'exemple de l'axe temps. On voudrait, par exemple, calculer l'évolution des mesures [Measures].[Unit Sales] et

[Measures].[Store Sales] entre le premier et le deuxième trimestre de l'année 1997. La requête suivante satisfait ce besoin.

```
with
  Member [Time].[1997].[Evolution]
as
  (
    ([Time].[1997].[Q2] - [Time].[1997].[Q1]) /
       [Time].[1997].[Q1]
  ), format_string = "Percent"
select
  {
      ([Measures].[Unit Sales]), ([Measures].[Store
         Sales])
  }
on columns,
  {
    ([Time].[1997].[Q1]), ([Time].[1997].[Q2]),
    ([Time].[1997].[Evolution])
  }
on rows
from [Sales]
```

Cette requête définit un membre calculé `[Time].[1997].[Evolution]` sur l'axe temporel pour déterminer l'évolution en pourcentage des mesures `[Unit Sales]` et `[Store Sales]` entre le deuxième et le premier trimestre de 1997, en soustrayant les valeurs du premier trimestre de celles du deuxième puis en divisant par les valeurs initiales, avec un formatage en pourcentage pour une interprétation immédiate. Les colonnes exposent ces deux mesures de base, tandis que les lignes juxtaposent les trimestres concernés et leur évolution résultante, le tout extrait de la base `[Sales]`, offrant ainsi une vue comparative concise des performances trimestrielles qui met en lumière les variations relatives sans recourir à des calculs post-requête.

À consulter : http://msdn.microsoft.com/en-us/library/ms146017.aspx

## 11.9. Ordre de résolution des membres calculés

Il peut arriver que vous ayez des ambiguïtés si vous utilisez plusieurs membres calculés qui se suivent. Regardons de plus près l'exemple de la requête suivante :

```
with
  Member [Measures].[Store Sales by Unit Sales]
```

```
as
  (
    [Measures].[Store Sales] / [Measures].[Unit
       Sales]
  ), format_string = '##.##'

  Member [Time].[1997].[Evolution]
as
  (
    ([Time].[1997].[Q2] - [Time].[1997].[Q1]) /
       [Time].[1997].[Q1]
  ), format_string = "Percent"
select
  CrossJoin
  (
    {([Measures].[Unit Sales]), ([Measures].[Store
       Sales]),
    ([Measures].[Store Sales by Unit Sales])},
    {([Time].[1997].[Q1]), ([Time].[1997].[Q2]),
    ([Time].[1997].[Evolution])}
  )
on rows,
  {
      ([Product].[Drink])
  }
on columns
from [Sales]
```

On se rend compte que l'on calcule non pas le pourcentage d'évolution des [Store Sales by Unit Sales], mais le ratio des évolutions entre le pourcentage d'évolution de [Unit Sales] (-1.36%) et le pourcentage d'évolution de [Store Sales] (2.84%). Ce qui ne veut absolument rien dire (une perte de 209%). Le calcul effectué est (0.0284+0.0136)/(-0.0136)=-2.09.

Pour éviter ce type d'erreur, il est possible de spécifier un ordre de résolution des membres calculés à l'aide du mot-clé `solve_order` :

```
with
  Member [Measures].[Store Sales by Unit Sales]
as
  (
    [Measures].[Store Sales] / [Measures].[Unit
       Sales]
  ), solve_order = 0, format_string = '##.##'

  Member [Time].[1997].[Evolution]
```

```
as
  (
    ([Time].[1997].[Q2] - [Time].[1997].[Q1])
    / [Time].[1997].[Q1]
  ), solve_order = 1, format_string = "Percent"
select
CrossJoin
(
  {([Measures].[Unit Sales]), ([Measures].[Store
     Sales]),
  ([Measures].[Store Sales by Unit Sales])},
  {([Time].[1997].[Q1]), ([Time].[1997].[Q2]),
  ([Time].[1997].[Evolution])}
)
on rows,
{
    ([Product].[Drink])
}
on columns
from [Sales]
```

Dans cet exemple, le membre calculé `[Measures].[Store Sales by Unit Sales]` divise les ventes en magasin par les ventes unitaires pour obtenir un ratio, avec un `solve_order = 0` qui le place en premier dans l'ordre de résolution, évitant ainsi les conflits potentiels lors de l'évaluation des dépendances entre membres. Le membre `[Time].[1997].[Evolution]`, quant à lui, calcule l'évolution en pourcentage entre le deuxième et le premier trimestre de 1997 en soustrayant et divisant les valeurs correspondantes, son `solve_order = 1` assurant qu'il soit résolu après les membres de base pour une cohérence logique. La requête finale utilise `CrossJoin` pour combiner ces membres sur les lignes avec la catégorie de produits `Drink` sur les colonnes, produisant un tableau croisé dynamique qui met en évidence les métriques de performance temporelle et financière à partir de la base de données `Sales`.

## 11.10. Exemples de manipulation de la dimension temps

La navigation temporelle est un point important et communément utilisée dans l'OLAP afin de mener des analyses comparatives. Dans ce cas, MDX permet de résoudre un certain nombre de questions de manière élégante et évite de construire des hiérarchies temps trop rigides. Nous nous employons ici à donner quelques exemples significatifs de questions couvertes de manière simple par le MDX.

### 11.10.1. 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. Prenons l'exemple de la requête suivante; la mesure [Unit Sales] est affichée en parallèle sur les trimestres 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]
```

La fonction `ParallelPeriod` a la signature suivante :

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

où niveau représente la période, sur laquelle on veut remonter dans le temps, l'expression numérique donne le nombre de périodes, et membre permet de fixer un membre sur la dimension temps.

Dans notre exemple, nous n'avons pas spécifié de membre. La valeur par défaut étant Dimension.currentMember (chaque dimension a une valeur par défaut fixée par le schéma). Mondrian va donc chercher dynamiquement pour chaque ligne ou chaque croisement le même membre sur deux périodes (ici le trimestre précédent).

### 11.10.2. Calcul cumulatif

Il arrive souvent qu'on ait besoin de calculer des cumuls sur une période de temps. Prenons dans notre cas le calcul du cumul de la mesure [Unit Sales] durant l'année 1997. Dans un premier temps, calculons le total des ventes par trimestre.

```
select
{
([Measures].[Unit Sales])
}
on columns,
  {
    ([Time].[1997].Children)} on rows
from [Sales]
```

Comment faire la même chose, mais avec les mois? On peut envisager un accès par membre, mais l'expression correspondante est un peu longue :

```
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]
```

Une façon plus élégante et moins fastidieuse d'obtenir la même chose est d'utiliser la fonction Descendants. Elle retourne un ensemble de descendants d'un membre sur un niveau ou à une distance spécifiée, et en option permet d'inclure ou d'exclure des descendants d'autres niveaux. Dans l'exemple qui suit, nous calculons les descendants d'une distance égale à deux (mois) à partir de l'année 1997.

```
select
  {
    ([Measures].[Unit Sales])
  }
on columns,
  {
    Descendants([Time].[1997],2)
  }
on rows
from [Sales]
```

ou

```
select
  {
    ([Measures].[Unit Sales])
  }
on columns,
  {
    Descendants([Time],[Time].[Month])
  }
on rows
from [Sales]
```

Cette requête démontre l'emploi élégant de la fonction `Descendants` pour extraire sur les lignes l'ensemble des mois (niveau à distance 2) de l'année `[1997]` dans la hiérarchie `[Time]`, ou de manière équivalente en spécifiant directement le niveau cible `[Time].[Month]` sur l'ensemble de la dimension `[Time]`, évitant ainsi les énumérations manuelles fastidieuses des membres. Les colonnes se limitent à la mesure `[Unit Sales]`, issue de la base `[Sales]`, produisant un tableau des ventes unitaires ventilées par mois sur l'année entière, avec une flexibilité accrue pour ajuster la profondeur hiérarchique sans reformuler l'ensemble de la requête.

La requête suivante donne les descendants du niveau mois du membre dont la valeur est le deuxième semestre.

```
select
  {
    ([Measures].[Unit Sales])
  }
on columns,
  {
    Descendants([Time].[1997].[Q2],[Time].[Month])
  }
on rows
from [Sales]
```

Cette requête emploie la fonction `Descendants` appliquée au membre `[Time].[1997].[Q2]` (représentant le deuxième trimestre de 1997) pour extraire sur les lignes l'ensemble des descendants au niveau `[Month]`, soit les mois constituant ce trimestre (juillet, août et septembre), offrant une ventilation fine des périodes mensuelles inférieures dans la hiérarchie temporelle. Les colonnes se focalisent sur la mesure `[Unit Sales]`, issue de la base de données `[Sales]`, afin de générer un tableau des ventes unitaires par mois au sein du trimestre sélectionné, facilitant

ainsi une analyse granulaire des tendances saisonnières sans inclure les niveaux supérieurs ou latéraux.

La fonction Descendants peut avoir un troisième paramètre optionnel, appelé drapeau pouvant prendre les valeurs suivantes :

- Self : Renvoie seulement 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 du 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 du 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 de 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 la 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.

Cette requête renvoie 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]
```

Cette requête applique la fonction `Descendants` à la hiérarchie `[Time]` pour sélectionner sur les lignes l'ensemble des membres situés aux niveaux descendants du niveau 1 (les trimestres), grâce au drapeau `AFTER` qui englobe ce niveau et tous les sous-niveaux inférieurs (tels que les mois ou jours), offrant une vue exhaustive des granularités temporelles fines. Les colonnes se limitent à la mesure `[Unit Sales]`, tirée de la base `[Sales]`, afin de produire un tableau détaillé des ventes unitaires ventilées par ces périodes descendantes, particulièrement utile pour des analyses approfondies où la précision chronologique prime sur la synthèse trimestrielle.

La requête qui suit donne 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]
```

Cette requête exploite la fonction `Descendants` sur la hiérarchie `[Time]` pour afficher sur les lignes l'ensemble des membres au niveau immédiatement supérieur au niveau 2 (les mois), c'est-à-dire les trimestres, en employant le drapeau `BEFORE` qui sélectionne les niveaux antérieurs au niveau spécifié sans inclure ce dernier ni les descendants inférieurs. Les colonnes se concentrent sur la mesure `[Unit Sales]`, extraite de la base de données `[Sales]`, générant ainsi un tableau synthétique des ventes unitaires agrégées par trimestre, idéal pour une vue consolidée à granularité intermédiaire qui évite la surcharge d'informations mensuelles tout en préservant la structure hiérarchique temporelle.

La requête suivante renvoie 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]
```

La fonction YTD (year-to-date) renvoie un set contenant les membres de même niveau en commençant par le membre début et en terminant par le membre fourni en paramètre. On peut aussi utiliser la fonction plus générale PeriodsToDate où la période peut être une année, un trimestre ou un mois. Il existe également d'autres fonctions comme MTD (month-to-date), QTD (quarter-to-date), WTD (week-to-date).

Dans notre cas, nous voulons cumuler (agrégation à l'aide de la fonction Sum) la valeur d'une mesure à partir d'une date début jusqu'à la date en cours (à l'aide de la fonction currentMeembre d'un membre donné). Dans l'exemple suivant, nous calculons le cumul de [Unit Sales] par intervalle d'un mois de l'année 1997. Si on voulait avoir le cumul par intervalle d'un trimestre, il suffit de reculer d'un niveau les descendants de la dimension [Time].

```
with
  Member [Measures].[Accumulated Unit Sales]
as
  Sum(YTD(), [Measures].[Unit Sales])
select
  {
    ([Measures].[Unit Sales]),
    ([Measures].[Accumulated Unit Sales])
  }
on columns,
  {
    Descendants([Time],[Time].[1997].[Q4].[12])
  }
on rows
from [Sales]
```

Cette requête définit le membre calculé `[Accumulated Unit Sales]` en agrégeant via `Sum` les valeurs de `[Unit Sales]` depuis le début de l'année jusqu'au membre courant de la dimension `[Time]`, grâce à la fonction `YTD` qui, sans paramètre explicite, s'appuie sur `[Time].CurrentMember` pour délimiter l'intervalle temporel. Les colonnes opposent les ventes unitaires brutes à ce cumul progressif, tandis que les lignes énumèrent les descendants de `[Time].[1997].[Q4].[12]`, listant ainsi les intervalles mensuels de fin d'année 1997 (comme décembre et ses sous-périodes si applicables) issus de la base `[Sales]`, pour une vue synthétique

des performances accumulées. Pour passer à un cumul trimestriel sur l'ensemble de 1997, il suffit de remonter d'un niveau dans la hiérarchie des descendants de `[Time]`, adaptant aisément la granularité sans modifier la logique de calcul. Dans ce dernier exemple, comme on ne comme on ne spécifie pas de paramètre pour YTD, il utilisera `[Time].currentMember`.

### 11.10.3. Formules conditionnelles

Parmi les fonctionnalités les plus intéressantes de MDX, il y a celle de pouvoir réaliser des tests conditionnels à l'aide du mot-clé IIF. La syntaxe de IIF est simple: IIF (condition, resultat si vrai, resultat si faux).

(Attention à la syntaxe: il s'agit bien de IIF et non pas de IF ou de IFFF. Par ailleurs, la fonction IIF prend 3 paramètres, jamais 2 ou 4.)

Dans l'exemple qui suit, nous construisons 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

```
with
    Member [Measures].[Q-1]
as
    (
    ParallelPeriod([Time].[Quarter],1,
    [Time].CurrentMember),
    [Measures].[Unit Sales]
    )
    Member [Measures].[Evolution]
as
    (
    ([Time].CurrentMember, [Measures].[Unit Sales])
    - (ParallelPeriod([Time].[Quarter],1,
    [Time].CurrentMember),
    [Measures].[Unit Sales])
    ), solve_order = 1
    Member [Measures].[%]
as
    ([Measures].[Evolution] /
     (ParallelPeriod([Time].[Quarter],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]
```

Cet exemple illustre l'utilisation pratique de la fonction `ParallelPeriod` pour naviguer dans la hiérarchie temporelle d'un cube OLAP, combinée à des expressions conditionnelles imbriquées avec `IIF`, qui permet de créer une logique décisionnelle sophistiquée. La clause `with` définit plusieurs membres calculés pour enrichir l'analyse des ventes unitaires `[Unit Sales]` au fil des trimestres. Le membre `[Measures].[Q-1]` utilise `ParallelPeriod([Time].[Quarter], 1, [Time].CurrentMember)` pour identifier le membre parallèle au trimestre courant, décalé d'un niveau vers l'arrière dans la hiérarchie des trimestres, puis associe cette position à la mesure `[Unit Sales]` afin de récupérer les ventes du trimestre précédent. L'instruction `Select` projette ces mesures sur l'axe des colonnes sous forme d'ensemble simple, tandis que l'axe des lignes liste les descendants de niveau 1 de la dimension `[Time]`, et la clause `from [Sales]` cible le cube des ventes. Cette requête transforme ainsi des métriques brutes en un tableau décisionnel, facilitant l'identification de tendances et l'activation de règles métier, tout en démontrant la puissance des calculs dynamiques en MDX pour une analyse temporelle agile.

À consulter : http://msdn.microsoft.com/en-us/library/ms145994.aspx

**Votre avis compte !**

Chers étudiants,
Si vous repérez une erreur dans ce document ou si vous avez une suggestion pour l'améliorer, nous vous invitons à remplir notre **formulaire anonyme**.
Votre contribution est précieuse pour nous !

Cliquez ici pour accéder au formulaire