Ecole Polytechnique Universitaire de Marseille
Département Informatique
5A – *Option INSI (Ingénierie des Systèmes d'Information)*

Travail Pratique

associé au cours

« Entrepôts de données »

Bernard ESPINASSE
2017-2018

2 - Mise en œuvre de MONDRIAN et JRUBIK

Sommaire

1. Objectif du travail pratique

Ce TP est associé au cours « Entrepôt de données » de 5 ième année de l'option INSI (INgénierie des Systèmes d'information). Il a pour objectif, à partir d'un entrepôt de données existant, de développer des cubes adaptés à des contextes décisionnels spécifiques, et d'en faire des analyses OLAP en utilisant le langage MDX introduit en cours.

2. Architecture logicielle utilisée

D'une façon générale, un entrepôt de données relationnel se résume en une architecture trois tiers. Celle-ci est composée d'une base relationnelle qui stocke et structure les données de l'entrepôt, d'une couche d'analyse OLAP ou serveur OLAP, et d'interfaces client permettant à l'utilisateur d'écrire, d'exécuter des requêtes OLAP, et d'en visualiser les résultats.

L'architecture logicielle utilisée pour ce TP est composée essentiellement de logiciels « open-source ». On utilise le SGBD PostgreSQL pour stocker la base de données relationnelle de l'entrepôt et le système décisionnel libre JRUBIK en Java, client utilisant le serveur OLAP MONDRIAN aussi écrit en Java.

Dans ce qui suit, nous présentons brièvement le serveur MONDRIAN et le client JRUBIK que nous utilisons pour le TP.

1) Le serveur MONDRIAN et le client JRUBIK

MONDRIAN est un serveur OLAP, développé en Open-Source depuis 2001 sous licence CPL. Ce serveur se présente sous la forme d'un ensemble de librairies Java. Ces librairies permettent tout d'abord la construction de structures multidimensionnelles (cubes notamment) à partir d'un entrepôt stocké dans une base de données relationnelle. Ces structures multidimensionnelles, ou cubes MONDRIAN, sont définies selon un schéma spécifié dans un fichier XML préalablement élaboré. Ce cube MONDRIAN peut être interrogé grâce à un client spécifique, ici JRUBIK.

JRUBIK est une application Open-Source Java autonome permettant la connexion et l'analyse OLAP d'un cube MONDRIAN en langage MDX. L'utilisateur dispose avec JRUBIK d'une interface graphique à base d'onglets lui permettant d'effectuer de façon interactive des analyses, des requêtes OLAP en MDX sur un cube MONDRIAN et d'en visualiser les résultats. Pour cela JRUBIK doit tout d'abord charger, à partir de la base de données relationnelle de l'entrepôt de l'entrepôt, les données dans le cube MONDRIAN spécifié dans le schéma XML, et ensuite agréger ces données en mémoire cache.

2) Fonctionnement général

Une fois le cube MONDRIAN spécifié dans un fichier en XML, et rempli par les données de l'entrepôt, le client JRUBIK et le serveur MONDRIAN fonctionnent ainsi (cf Fig. 1) :

1. le client JRUBIK transmet une requête au format MDX écrite par l'utilisateur au serveur MONDRIAN,

2. après validation du format de la requête MDX, MONDRIAN utilise le schéma XML pour transformer la requête MDX en requêtes SQL et passe la main à la couche de requêtage SQL,
3. la couche de requêtage SQL de MONDRIAN accède à la base de données relationnelle physique de l'entrepôt et récupère les données résultantes des requêtes SQL exécutées,
4. les résultats sont consolidés et transmis au client JRUBIK pour être présentés à l'utilisateur.

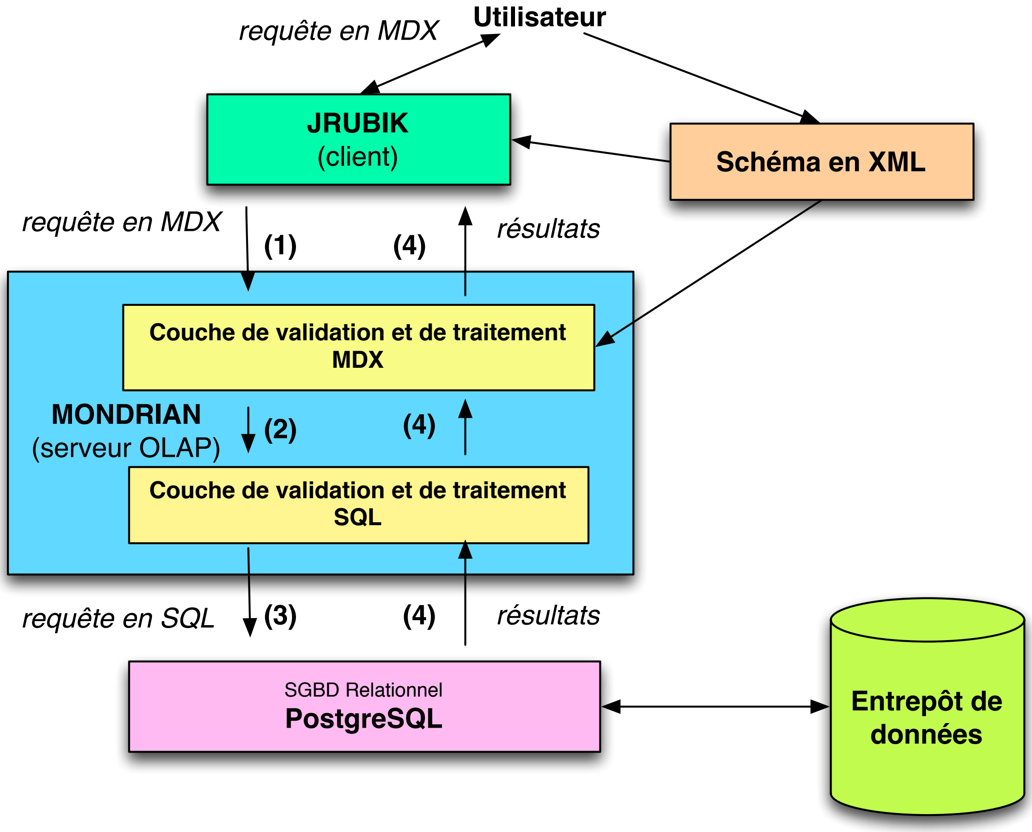


Fig. 1 : Architecture utilisée

3. Installation et configuration de JRUBIK/MONDRIAN

1) Installation de JRUBIK/MONDRIAN

Le télécharger ou sur la page Web du cours (ou à l'adresse : http://rubik.sourceforge.net/jrubik/intro.html)

JRUBIK contient la bibliothèque MONDRIAN, ainsi son installation installe en même temps le serveur OLAP MONDRIAN (bibliothèque Java).

Vérifier que jdbc a bien été installé sur la machine …

Un manuel d'installation et d'utilisation détaillé de JRUBIK est téléchargeable sur la page du cours ou au lien : http://rubik.sourceforge.net/jrubik/intro.html

2) Configuration de la connexion au serveur PostgreSQL

Afin de pouvoir exploiter les données stockées dans la base de données entrepôt sous PostgreSQL, il est nécessaire d'indiquer à JRUBIK, à partir du poste de l'administrateur Postgresql (poste de l'utilisateur administrateur Postgresql), où se trouve la BD de l'entrepôt et comment récupérer les données. Pour cela, il faut configurer la connexion en spécifiant l'adresse, le port et le type de base de données (cf. Fig. 2). Attention : pensez à modifier l'URL en spécifiant l'adresse du serveur PostgreSQL.

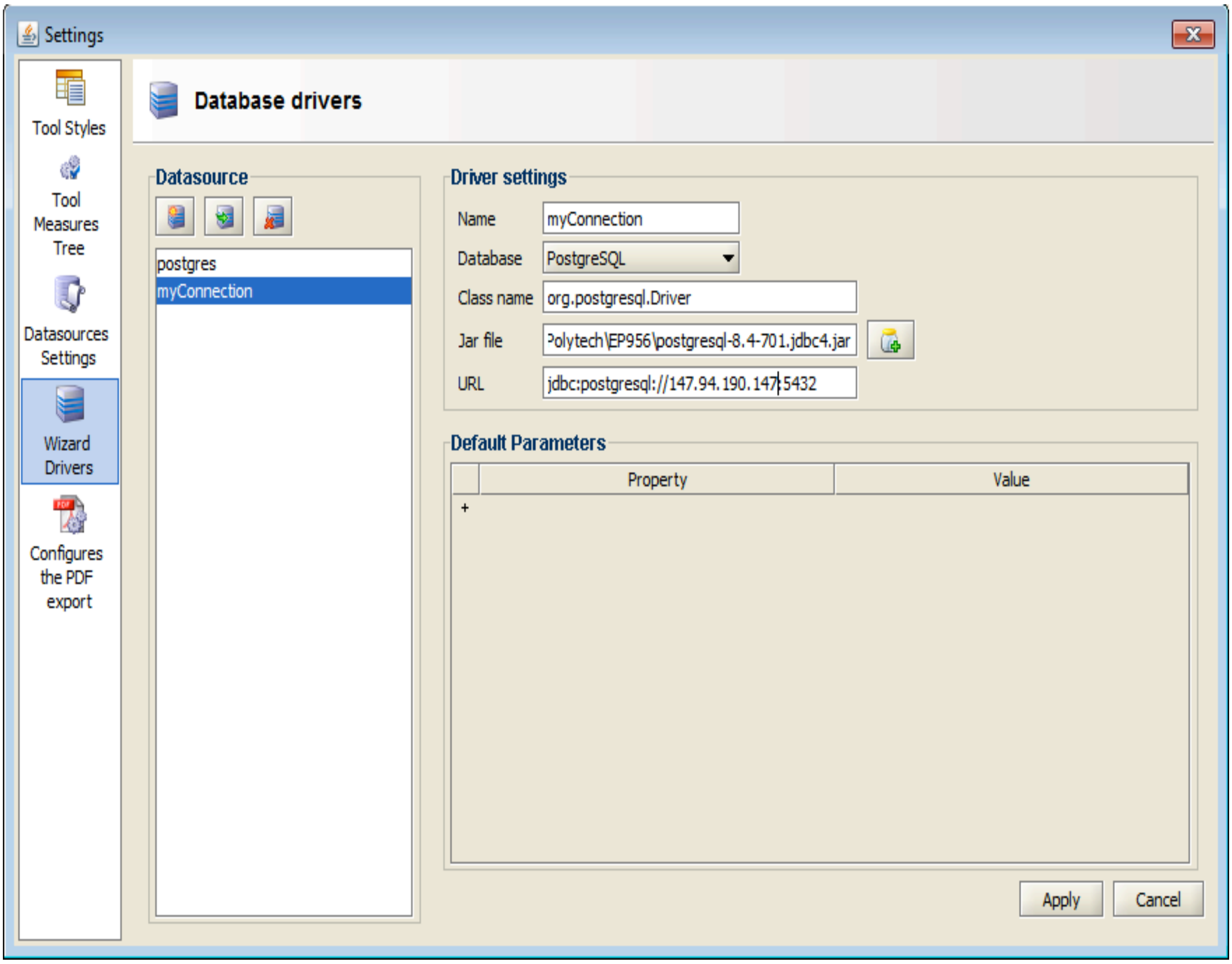


Fig. 2 : Configuration de la connexion en JRUBIK

3) Configuration de source de données en JRUBIK

La configuration de sources de données ou «DataSource», est la principale configuration du JRUBIK. C'est dans DataSource que JRUBIK saura où se trouve le fichier XML qui spécifie le cube, et où se trouve la base de données (en utilisant la connexion précédemment créée).

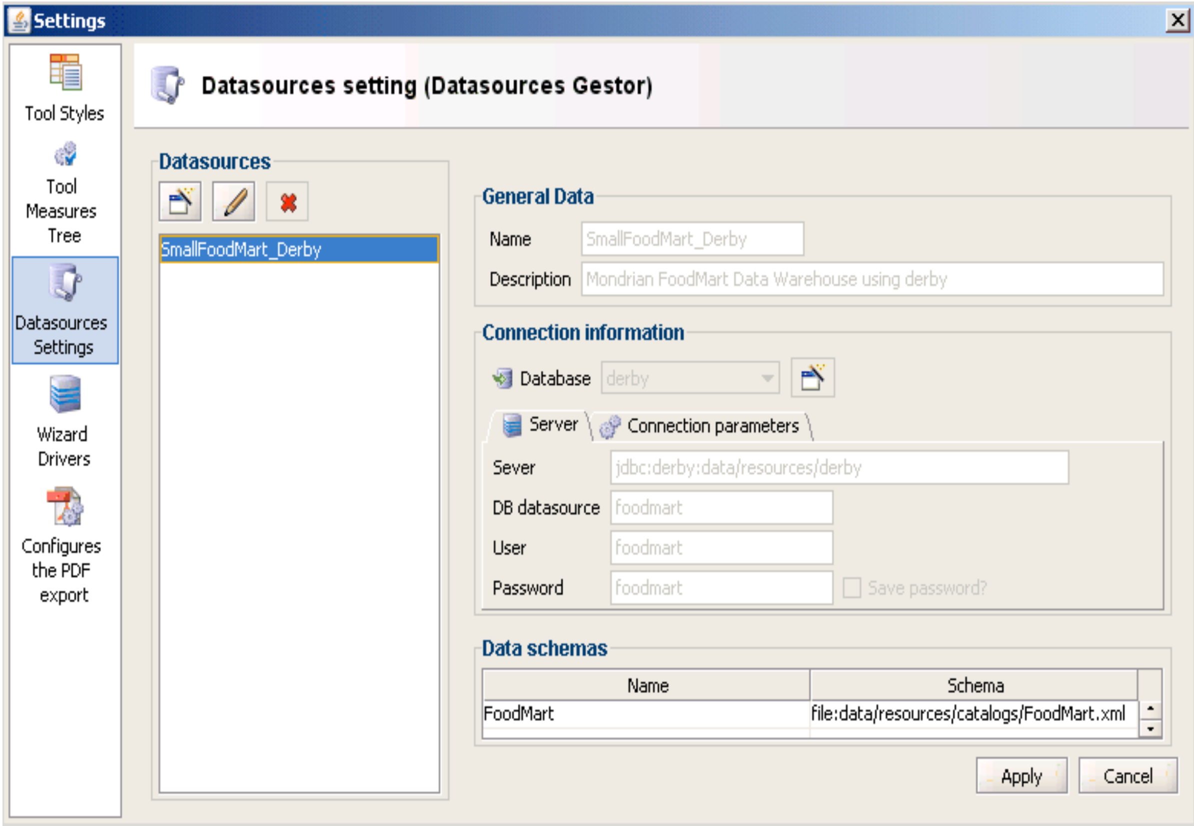
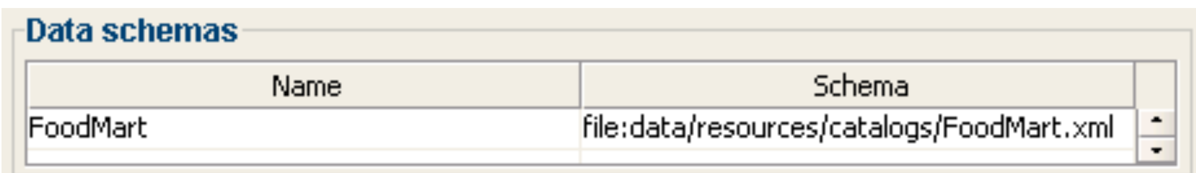


Fig. 3 : Configuration de sources de données en JRUBIK

Pour ajouter une nouvelle *Datasource,* il faut appuyer sur le bouton  (create new datasource). Et ensuite remplir les champs suivants (cf. Fig. 3) :

- *Name:* N'importe quelle source de données doit avoir un nom avec lequel l'outil l'identifie. Exemple : SmallFoodMart_Derby.
- *Description.* Une brève description de la source de données est indiquée *:*
  - *Database: type de la base de données (ex: MySql, PostgreSQL, Oracle, etc.)*
  - *Server:* jdbc:postgresql://<server>:5432, le texte <server> va être remplacé par le serveur qui contient l'information. Le mot JDBC (Java Database Conectivity) indique que c'est le «protocole» avec lequel nous pouvons accéder à partir de MONDRIAN à la base de données puisqu'il s'agit d'une librairie Java.
  - *Source of the data (Schema of BD):* Il est nécessaire d'indiquer dans quel schéma l'information se trouve dans le serveur PostgreSQL.
  - *User and user's password:* sont le nom et le mot de passe de l'utilisateur nécessaires pour établir la connexion avec la base de donnée pour faire les requêtes.
  - *Information schema:* Un tableau apparaît avec 2 colonnes Nom et Schema, où sont saisi les fichiers spécifiant le ou les schéma(s) en XML. (par exemple celui d'un cube « vente ») :



4. Exemple d'entrepôt sous MONDRIAN

1) Description de l'entrepôt

Soit un entrepôt de données organisé en étoile autour d'une **table de fait « SALES »** avec les **mesures** (*measures*) suivantes : *Unit Sales, Store Cost, Store Sales, Sales Count* et *Customer Count,* et les **dimensions** suivantes : *Time, Product, Yearly Income, et Customers*.

Les **dimensions** sont généralement organisées en hiérarchies, ainsi dans notre exemple, la dimension ***Time*** et la dimension ***Product*** sont organisées ainsi (cf Fig. 4) :

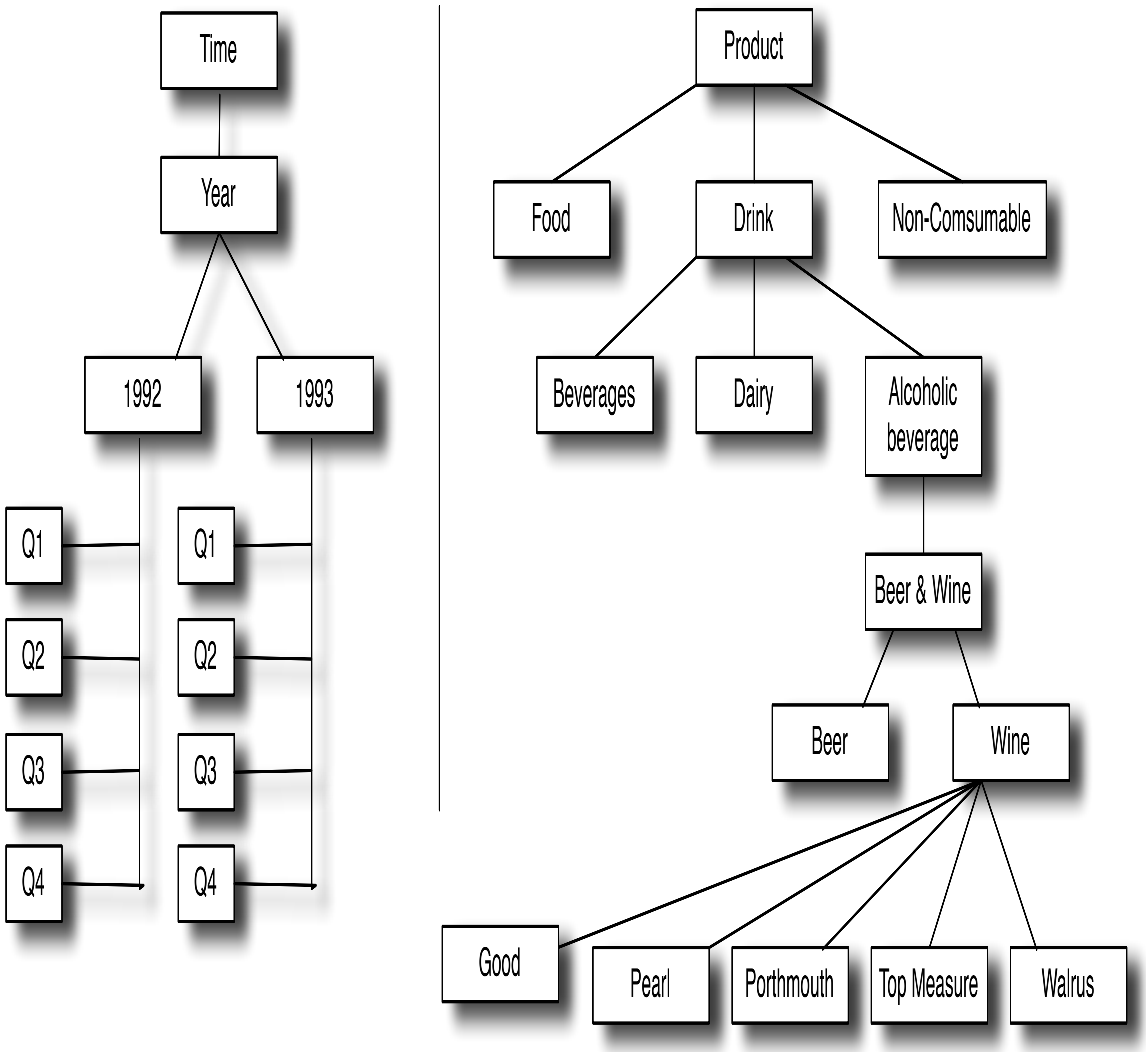


Fig. 4: Les dimensions Time et Product de l'entrepôt exemple.

Différents cubes peuvent être construits à partir de ces dimensions (avec leur niveaux) et ces mesures. La figure suivante présente la visualisation en MONDRIAN/JRUBIK d'un cube « Sales » construit à partir de l'entrepôt (cf. Fig. 5) :

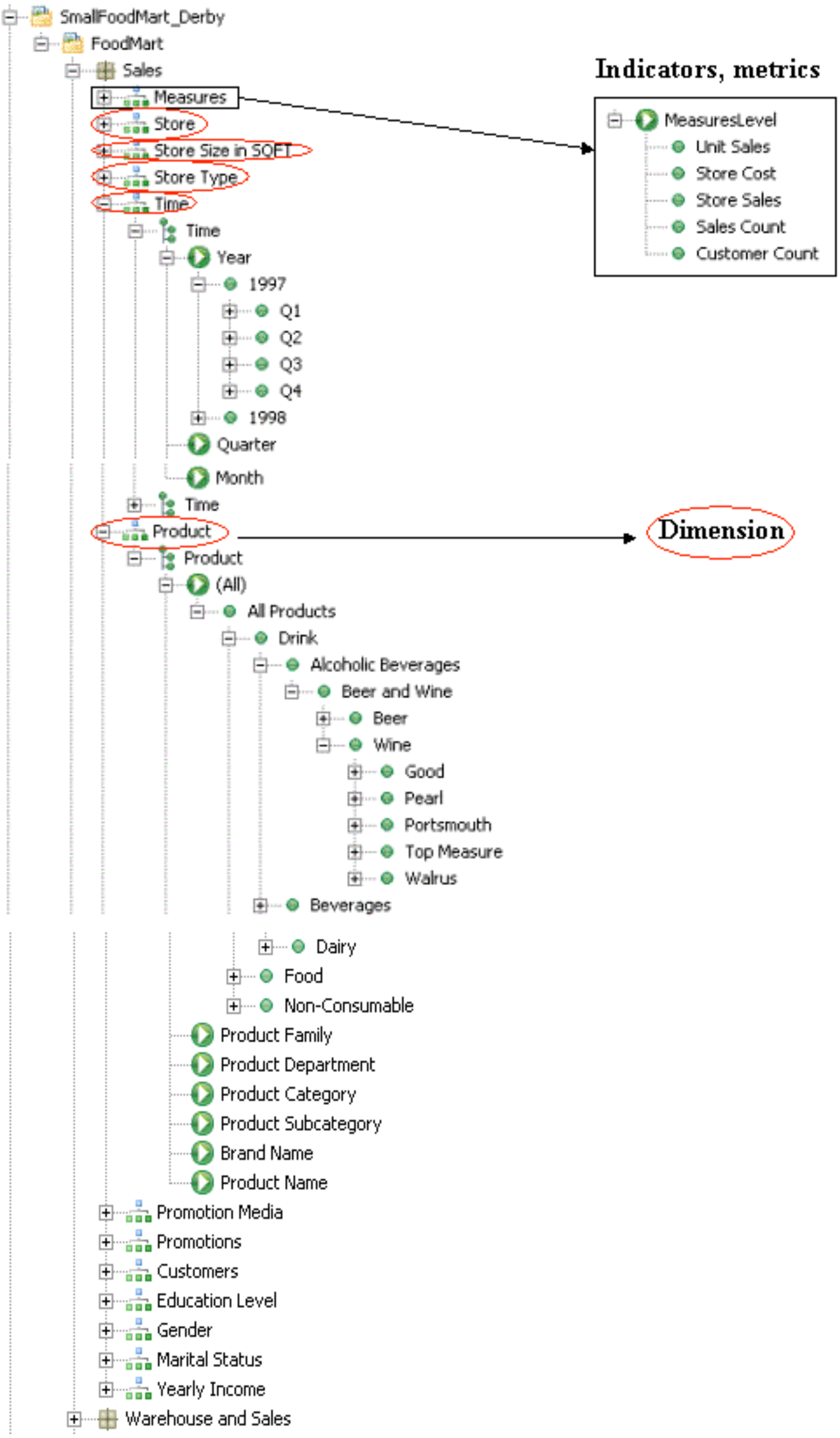


Fig. 4. Visualisation en MONDRIAN/JRUBIK d'un cube construit sur l'entrepôt.

2) Schéma MONDRIAN de l'entrepôt en XML

Un schéma MONDRIAN spécifie une **structure multidimensionnelle** en XML ainsi que la source des données représentées dans le modèle logique de données de la base de données relationnelle de l'entrepôt, généralement en étoile. Ci-dessous, un exemple d'un schéma MONDRIAN simple en XML spécifiant un cube « Sales » :

```
<Schema>
 <Cube name="Sales">
  <Table name="sales_fact_1997"/>
<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>
 <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>
  <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>
```

Dans ce cube « Sales », les ventes sont observées sur 2 dimensions "***Time***" et "***Gender***", et 4 mesures "***Unit Sales***", "***Store Sales***", "***Store cost***" et "***Profit***". Dans les sections suivantes on précise plus en détail la spécification des structures *cube*, *mesure* et *dimension*.

3) Cube

Un cube (<Cube>) est une collection de mesures et de dimensions, identifiée par un nom. Les dimensions et les mesures ont la table de faits en commun (dans l'exemple, la table de faits est "sales_fact_1997"). Le cube « Sales » est spécifié dans le schéma précédent ainsi :

```
<Cube name="Sales">
<Table name="sales_fact_1997"/>
...
</Cube>
```

La **table de faits** est définie en utilisant <**Table**>. Si la table de faits n'est pas dans le schéma par défaut, on peut fournir explicitement son schéma en utilisant l'attribut "schema". Par exemple, <Table schema="dmart" name="sales_fact_1997"/>. On peut aussi utiliser <View> et <Join> pour construire des commandes SQL encore plus complexes.

4) Mesures

Il existe 2 types de mesures : les mesures **calculées** et **non calculées**. Dans le schéma XML précédent, le cube « Sales » des ventes définit plusieurs mesures, dont "Unit Sales" et "Store Sales". La mesure "Profit" est une mesure calculée à partir des mesures "Store Sales" et "Store Cost". Voici ces spécifications liées au cube « Sales » :

```
<Measure name="Unit Sales" column="unit_sales" aggregator="sum" datatype="Integer"
    formatString="#,###"/>
<Measure name="Store Sales" column="store_sales" aggregator="sum" datatype="Numeric"
    formatString="#,###.00"/>
<CalculatedMember name="Profit" dimension="Measures" formula="[Measures].[Store Sales]-
    [Measures].[Store Cost]">
```

Chaque mesure (<**Measure**>) a un **nom**, une **colonne de correspondance dans la table de faits**, et un **opérateur d'agrégation**. L'opérateur d'agrégation est souvent "sum", mais d'autres opérateurs comme "count", "min", "max", "avg" et "distinct-count" peuvent être utilisés. L'opérateur "distinct-count" a des limitations si le cube contient une hiérarchie parent-fils.

Remarques :

1- **L'attribut optionnel "datatype"** spécifie les types selon lesquels les valeurs des cellules seront représentées dans le cache de MONDRIAN et comment elles seront retournées. L'attribut "datatype" peut avoir pour valeur "String", "Integer", "Numeric", "Boolean", "Date", "Time", et "Timestamp". La valeur par défaut est "**Numeric**" à l'exception des opérateurs "count" et "distinct-count" qui ont une valeur par défaut "**Integer**".

2- **L'attribut optionnel "formatString"** donne le format à utiliser lors de l'affichage des données. Par exemple, la mesure "Unit Sales" est affichée sans décimales, alors que la mesure "Store Sales" est affichée avec deux décimales. L'emplacement des caractères ',' et '.' dépend des notations adoptées pour afficher les décimales.

3- **L'attribut "caption".** Une mesure peut utiliser un attribut "caption" (étiquette en français) qui sera retourné à la place du nom lors d'un appel à l'aide de la méthode Member.getCaption() dans une requête MDX. La définition d'une étiquette a un sens lorsque les caractères comme Σ ou Π sont utilisés, par exemple :

```
<Measure name="Sum X" column="sum_x" aggregator="sum" caption="Σ X"/>
```

5) Dimensions, membres, hiérarchies et niveaux

Dans une *dimension*, un *membre* est un point déterminé par les valeurs d'attributs de cette dimension, par exemple la hiérarchie "Gender" a 2 membres 'M' et 'F'.

Une *hiérarchie* est un ensemble de membres organisés selon une structure appropriée pour l'analyse. Par exemple, les villes peuvent être regroupées par région et les régions par pays. Les *mesures* étant agrégées pour chaque niveau de la hiérarchie, les ventes d'un pays sont calculées à partir des ventes de ses régions.

Un *niveau* est une collection de membres qui ont la même distance de la racine de la hiérarchie. Une dimension est une collection de hiérarchies selon laquelle les faits sont observés. Ci-dessous un exemple d'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>
```

Pour chaque vente, la dimension "Gender" donne le sexe du client ayant réalisé un achat. La dimension « Gender » a une seule hiérarchie et un seul niveau, et 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 donc 2 membres "[Gender].[F]" et "[Gender].[M]". Cela s'exprime par la jointure entre le table de faits "sales_fact_1997" et la table de dimensions "customer" sur l'attribut "customer_id".

6) Correspondances des dimensions et des hiérarchies avec les tables

Une dimension est jointe avec un cube à l'aide de deux colonnes : une colonne dans la table de faits et l'autre 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, c'est 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" rightKey="product_class_id">
    <Table name="product_class"/>
     <Table name="product"/>
   </Join>
  <!-- Level declarations ... ->
  </Hierarchy>
 </Dimension>
</Cube>
```

Remarque : l'attribut **uniqueMembers** est utilisé pour optimiser les commandes SQL. Si on sait que les valeurs d'un niveau d'une table de dimensions sont uniques sur toutes les valeurs d'un niveau parent, il faut alors mettre l'attribut uniqueMembers="true" et "false" sinon. Par exemple, dans la dimension "Time", la hiérarchie "[Year].[Month]" peut avoir la valeur uniqueMembers="false" au niveau de "Month" car un même mois peut être présent dans différentes années. Dans la hiérarchie "[Product Class].[Product Name]", si l'on est sûr que "[Product Name]" est unique, alors on peut mettre uniqueMembers="true". Dans le cas contraire, mettez "false". Dans le niveau le plus haut de la hiérarchie, l'attribut uniqueMembers="true" car il n'y a aucun niveau parent.

7) Membre « All » et par défaut

Par défaut, toute hiérarchie contient un niveau haut appelé ***'All'***, qui contient un seul membre appelé '(All {hierarchyName})'. Ce membre est le parent de tous les autres membres de la hiérarchie et représente l'agrégation totale sur cette dimension. C'est le membre par défaut de la hiérarchie. Il est le membre utilisé pour calculer les valeurs d'une cellule lorsque cette hiérarchie n'est pas incluse dans un axe ou un "slicer".

Si l'élément <Hierarchy> a **hasAll="false",** le niveau 'All' est supprimé. Le membre *par défaut* d'une telle dimension est le premier membre du premier niveau. Par exemple, dans la hiérarchie "Time", le membre par défaut serait la première année de la hiérarchie. Le changement du membre par défaut peut prêter à confusion, alors on doit le laisser toujours vrai (hasAll="true"). L'élément <Hierarchy> a également l'attribut "defaultMember", utilisé pour changer le membre par défaut de la hiérarchie. Ci-dessous, un exemple :

```
<Dimension name="Time" type="TimeDimension" foreignKey="time_id">
<Hierarchy hasAll="false" primaryKey="time_id" defaultMember="[Time].[1997].[Q1].[1]"/>
   ...
```

8) La dimension « temps »

La dimension temps est basée sur l'année, le mois, la semaine et le jour. Elle est implantée d'une manière particulière sous MONDRIAN pour prendre en compte la spécificité des fonctions liées au temps comme :

- ParallelPeriod([level[, index[, member]]])
- PeriodsToDate([level[, member]])
- WTD([member])
- MTD([member])
- QTD([member])
- YTD([member])
- LastPeriod(index[, member])

La dimension temps a un attribut "Type" dont la valeur est "TimeDimension". Le rôle d'un niveau d'une dimension temps est indiqué par l'attribut "levelType", dont les valeurs permises sont les suivantes :

TimeYears	niveau représentant les années
TimeQuarters	niveau représentant les trimestres
TimeMonths	niveau représentant le mois

TimeWeeks niveau représentant la semaine
TimeDays niveau représentant les jours

```
<Dimension name="Time" type="TimeDimension">
  <Hierarchy hasAll="true" allMemberName="All Periods" primaryKey="dateid">
  <Table name="datehierarchy"/>
      <Level name="Year" column="year" uniqueMembers="true"
        levelType="TimeYears" type="Numeric"/>
      <Level name="Quarter" column="quarter" uniqueMembers="false"
        levelType="TimeQuarters" />
      <Level name="Month" column="month" uniqueMembers="false" ordinalColumn="month"
         nameColumn="month_name"
        levelType="TimeMonths" type="Numeric"/>
      <Level name="Week" column="week_in_month" uniqueMembers="false"
         levelType="TimeWeeks" />
      <Level name="Day" column="day_in_month" uniqueMembers="false"
        ordinalColumn="day_in_month" nameColumn="day_name"
         levelType="TimeDays" type="Numeric"/>
  </Hierarchy>
</Dimension>
```

5. Analyses OLAP en JRUBIK

Rappelons qu'un manuel d'utilisation détaillé de JRUBIK est téléchargeable sur la page du cours ou au lien : http://rubik.sourceforge.net/jrubik/intro.html

Pour réaliser des analyses multidimensionnelles OLAP en MDX, JRUBIK doit tout d'abord charger à partir de la base de données relationnelle de l'entrepôt, les données dans le cube MONDRIAN spécifié dans le schéma MONDRIAN XML, et ensuite agréger ces données dans une mémoire cache.

1) Usage de l'interface JRUBIK

L'interface JRUBIK est à base d'onglets. Il est composé de (cf Fig. 6) :

- un menu principal
- une barre d'outils
- de plusieurs grandes fenêtres permettant chacune de :
    - choisir les objets à afficher et filtrer
    - choisir les lignes et colonnes d'affichage
    - d'afficher les résultats
    - d'afficher le code MDX généré et permettre sa modification (édition)

Pour le mettre en œuvre on procède globalement ainsi :

1) Menu File -> Settings -> **Wizard drivers** (*Configuration de la connexion*)
2) Menu File->Settings->**Datasources Settings** (*Configuration des sources de données*)
3) Menu Window->Focus View->**Discoverer**

Les sections suivantes expliquent plus en détail ces opérations.

2) Création de requêtes OLAP

Pour créer une requête OLAP dans JRUBIK, il est nécessaire d'utiliser **l'onglet « discoverer »**.

L'expression de la requête OLAP est faire à la souris, pour cela, faire glisser (Drag) les hiérarchies choisies (Seulement des hiérarchies) et les déposer (Drop) sur les boutons « ON Rows » ou «ON colonnes »,

Appuyer enfin sur le bouton « **Apply** » (cf. Fig. 6).

On notera que :

- La **fenêtre « Table_X »** de l'éditeur affiche les résultats de la requête.
- La **fenêtre de l'onglet « MDX query_X »** de l'éditeur, affiche le code MDX associé à la requête et généré par l'outil. Elle permet aussi de créer, modifier et lancer une requête OLAP en MDX.

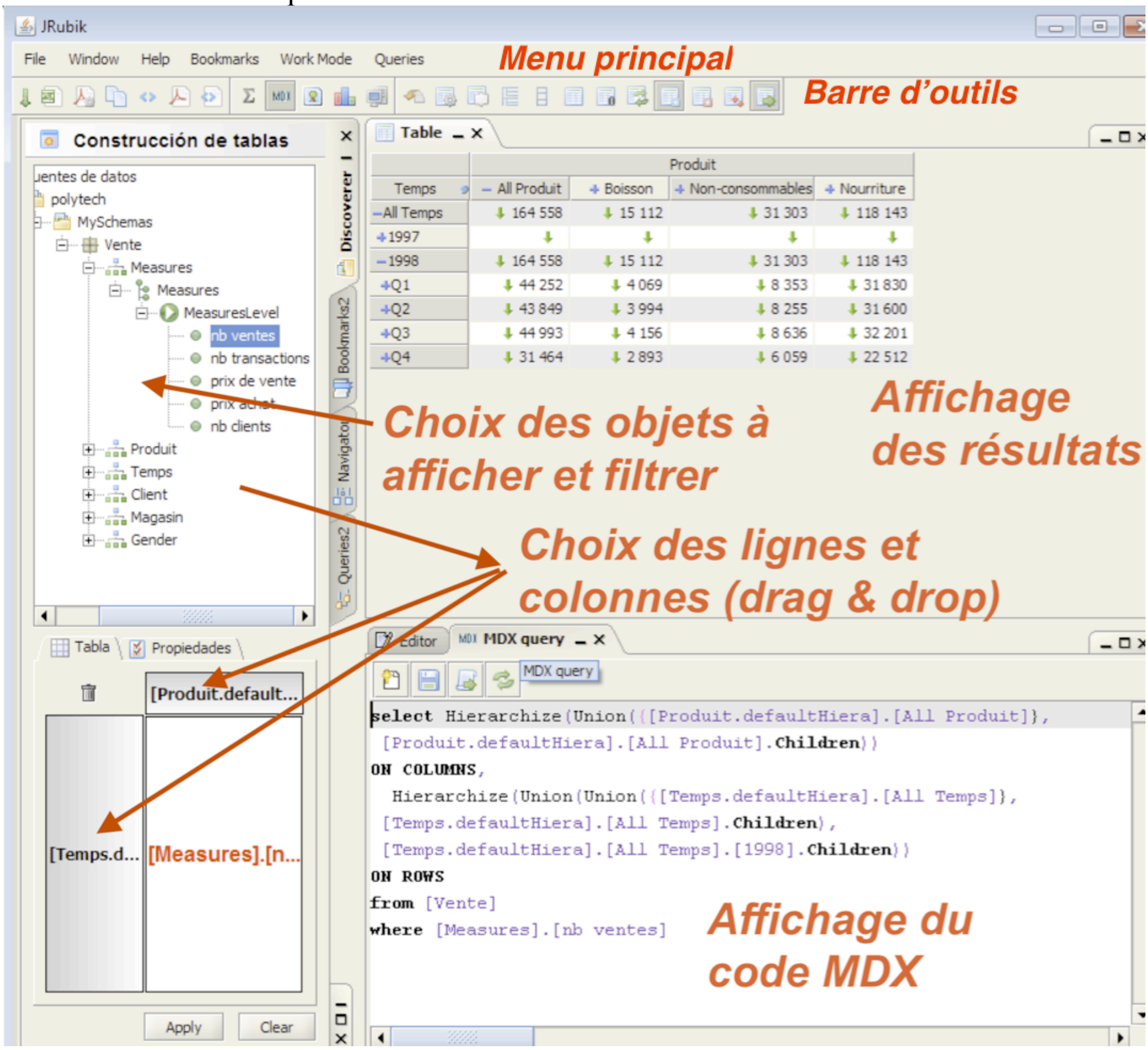


Fig. 6 : Création d'une première requête OLAP en MDX sous JRUBIK.

Une autre requête OLAP en JRUBIK est présentée à la Fig. 7.

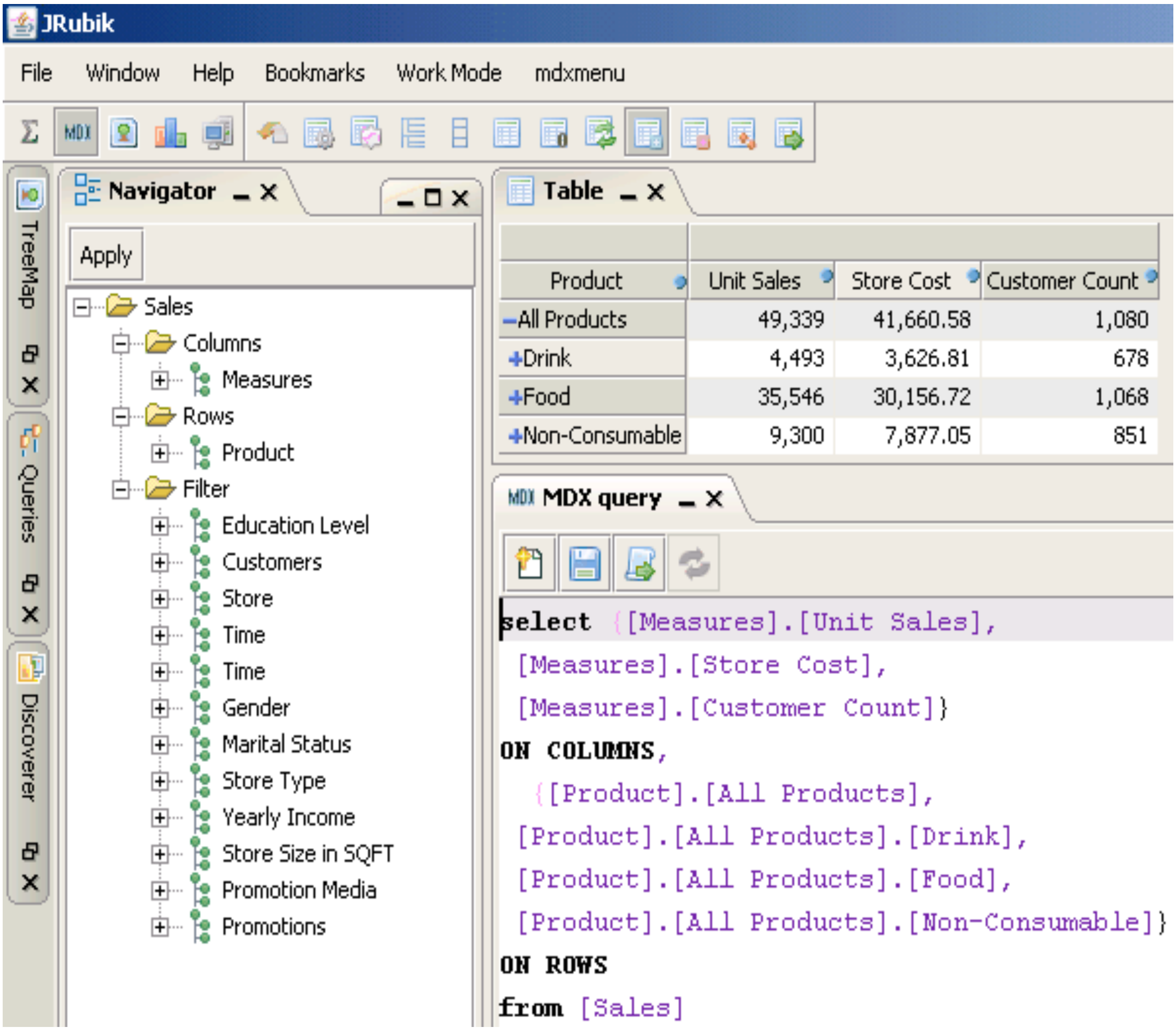


Fig. 7 : Une autre requête OLAP en MDX sous JRUBIK.

3) Fonctions MDX usuelles

- « **Descendant** » : Retourne le jeu de descendants d'un membre à un niveau spécifié ou à une distance spécifiée, en incluant ou en excluant éventuellement des descendants dans d'autres niveaux.
- « **AddCalculateMembers** » : Retourne un jeu généré par l'ajout de membres calculés à un jeu spécifié.
- « **TopCount** » : Trie un jeu en ordre décroissant et retourne le nombre spécifié d'éléments avec les valeurs les plus élevées.

  Syntaxe : TopCount(Set_Expression,Count [,Numeric_Expression])
- « **Filter** » : Retourne le jeu résultant du filtrage d'un jeu spécifié selon une condition de recherche.

Syntaxe: Filter(Set_Expression, Logical_Expression) ex : ([A].members, not IsEmpty(b.CurrentMember)).

- « **IsEmpty** »: Retourne une valeur Boolean indiquant si l'expression évaluée est la valeur de la cellule vide.
- « **Tail** »: Retourne un sous-ensemble de la fin d'un jeu.

  Syntaxe: Tail(Set_Expression [,Count]) ex : Tail([A].Members, 5) ;

- « **Union** » : Retourne un jeu produit par l'union de deux jeux, en conservant éventuellement les membres en double.
- « **Lag** » : Retourne le membre qui est un nombre spécifié de positions avant un membre spécifié au niveau du membre. Syntax : Member_Expression.Lag(Index).

  ex : L'exemple ci-après retourne la valeur du mois de décembre 2001 :

  ```
  SELECT [Date].[Fiscal].[Month].[February 2002].Lag(2) ON 0
  FROM [Adventure Works]
  ```

- « **Generate** » : Applique un jeu à chaque membre d'un autre jeu, puis effectue la jointure par union des jeux résultants. Cette fonction retourne également une chaîne concaténée créée par l'évaluation d'une expression de chaîne sur un jeu.

  Syntaxe :
  Generate(Set_Expression1 , (Set_Expression2 [, ALL]))
  Generate(Set_Expression1 , (String_Expression [, Delimiter]))
  Exemple: la requête suivante retourne un jeu qui contient le montant des ventes sur Internet de Mesure quatre fois, parce qu'il y a 4 membres dans le jeu [Date].[Calendar Year].[Calendar Year].MEMBERS :

  ```
  SELECT GENERATE( [Date].[Calendar Year].[Calendar Year].MEMBERS,
  {[Measures].[Internet Sales Amount]}, ALL) ON 0
  FROM [Adventure Works]
  ```

- « **Filter** » : Retourne le jeu résultant du filtrage d'un jeu spécifié selon une condition de recherche.

  Syntaxe: Filter(Set_Expression, Logical_Expression)
  L'exemple suivant illustre l'utilisation de la fonction Filter sur l'axe des lignes d'une requête afin de retourner uniquement les dates où le Montant des ventes sur Internet est supérieur à $10 000 :

  ```
  SELECT [Measures].[Internet Sales Amount] ON 0,
  FILTER( [Date].[Date].[Date].MEMBERS, [Measures].[Internet Sales
  Amount]>10000)
  ON 1
  FROM
  [Adventure Works]
  ```

- « **Ancestor** » : Retourne l'ancêtre d'un membre spécifié à un niveau ou une distance spécifique du membre.

  Exemple : L'exemple suivant utilise une expression de niveau et retourne la mesure Internet Sales Amount (volume de vente Internet) pour chaque State-Province (état-Province) en Australie ; il dévoile également le pourcentage de volume de vente Internet total pour l'Australie:

```
WITH MEMBER Measures.x AS [Measures].[Internet Sales Amount] /
  (
  [Measures].[Internet Sales Amount],
    Ancestor
      (
      [Customer].[Customer Geography].CurrentMember,
        [Customer].[Customer Geography].[Country]
      )
  ), FORMAT_STRING = '0%'
SELECT {[Measures].[Internet Sales Amount], Measures.x} ON 0,
{
  Descendants
    (
     [Customer].[Customer Geography].[Country].&[Australia],
       [Customer].[Customer Geography].[State-Province], SELF
    )
} ON 1
FROM [Adventure Works]
```

- « **TopPercent** » : Trie un jeu en ordre décroissant et retourne un jeu de tuples avec les valeurs les plus élevées dont le total cumulé est égal ou supérieur à un pourcentage spécifié.

  Syntaxe : TopPercent(Set_Expression, Percentage, Numeric_Expression)

- « **BottomPercent** » : Trie un jeu en ordre croissant, et retourne un jeu de tuples avec les valeurs les plus basses dont le total cumulé est supérieur ou égal à un pourcentage spécifié.

  Syntaxe : BottomPercent(Set_Expression, Percentage, Numeric_Expression)

- « **Crossjoin** » : Retourne le produit croisé d'un ou plusieurs jeux.

  Syntaxe : Crossjoin(Set_Expression1 ,Set_Expression2 [,...n])

  Exemple: La requête suivante affiche des exemples simples de l'utilisation de la fonction Crossjoin sur les axes de colonnes et de lignes d'une requête :

```
SELECT
[Customer].[Country].Members *
[Customer].[State-Province].Members
ON 0,
Crossjoin(
[Date].[Calendar Year].Members,
[Product].[Category].[Category].Members)
ON 1
FROM [Adventure Works]
WHERE Measures.[Internet Sales Amount]
```

- « **Sum (MDX)** » : Retourne la somme d'une expression numérique évaluée sur un jeu spécifié.

  Syntaxe : Sum(Set_Expression [, Numeric_Expression])

L'exemple ci-dessous retourne la somme de la mesure Reseller Sales Amount (volume de vente du revendeur) de tous les membres de la hiérarchie d'attribut Product.Category pour les années civiles 2001 et 2002.

```
WITH MEMBER Measures.x AS SUM
  ( { [Date].[Calendar Year].&[2001]
      , [Date].[Calendar Year].&[2002] }
    , [Measures].[Reseller Sales Amount]
  )
SELECT Measures.x ON 0
,[Product].[Category].Members ON 1
FROM [Adventure Works]
```

- « **IIf (MDX)** » : Retourne l'une des deux valeurs déterminées par un test logique.

Syntax :

```
IIf(Logical_Expression, Expression1 [HINT <hints>], Expression2 [HINT <hints>])
[HINT <hints>]
<hints> ::= <hint> [<hints>]
<hint> ::= EAGER | STRICT | LAZY
```

avec:

Logical_Expression : Expression logique MDX (Multidimensional Expressions) valide qui prend la valeur true ou false.

Expression1 [HINT <hints>] : Expression MDX (Multidimensional Expressions) valide. HINT <hints> est un modificateur facultatif qui détermine comment et quand l'expression est évaluée. Pour plus d'informations, consultez la section Remarques.

Expression2[HINT <hints>] *:* Expression MDX (Multidimensional Expressions) valide. HINT <hints> est un modificateur facultatif qui détermine comment et quand l'expression est évaluée. Pour plus d'informations, consultez la section Remarques.

La requête suivante illustre une utilisation simple de IIF à l'intérieur d'une mesure calculée pour retourner l'une des deux valeurs de chaîne distinctes lorsque la mesure Internet Sales Amount est supérieure ou inférieure à 10 000 € :

```
WITH MEMBER MEASURES.IIFDEMO AS
IIF([Measures].[Internet Sales Amount]>10000
, "Sales Are High", "Sales Are Low")
SELECT {[Measures].[Internet Sales Amount],MEASURES.IIFDEMO} ON 0,
[Date].[Date].[Date].MEMBERS ON 1
FROM [Adventure Works]
```

Une utilisation très courante **d'IIF** est de gérer les erreurs de 'division par zéro' dans les mesures calculées, comme dans l'exemple suivant :

```
WITH
//Returns 1.#INF when the previous period contains no value
//but the current period does
MEMBER MEASURES.[Previous Period Growth With Errors] AS
([Measures].[Internet Sales Amount]-([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER))
/
```

```
([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER)
,FORMAT_STRING='PERCENT'
//Traps division by zero and returns null when the previous period contains
//no value but the current period does
MEMBER MEASURES.[Previous Period Growth] AS
IIF(([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER)=0,
NULL,
([Measures].[Internet Sales Amount]-([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER))
/
([Measures].[Internet Sales Amount],
[Date].[Date].CURRENTMEMBER.PREVMEMBER)
),FORMAT_STRING='PERCENT'
SELECT {[Measures].[Internet Sales Amount],MEASURES.[Previous Period
Growth With Errors], MEASURES.[Previous Period Growth]} ON 0,
DESCENDANTS(
[Date].[Calendar].[Calendar Year].&[2004],
[Date].[Calendar].[Date])
ON 1
FROM [Adventure Works]
WHERE([Product].[Product Categories].[Subcategory].&[26])
```

Les éléments suivants sont un exemple de **IIF** qui retourne l'un des deux jeux à l'intérieur de la fonction Generate pour créer un jeu complexe de tuple sur les lignes :

```
SELECT {[Measures].[Internet Sales Amount]} ON 0,
//If Internet Sales Amount is zero or null
//returns the current year and the All Customers member
//else returns the current year broken down by Country
GENERATE(
[Date].[Calendar Year].[Calendar Year].MEMBERS
, IIF([Measures].[Internet Sales Amount]=0,
{([Date].[Calendar Year].CURRENTMEMBER, [Customer].[Country].[All
Customers])}
, {{[Date].[Calendar Year].CURRENTMEMBER} *
[Customer].[Country].[Country].MEMBERS}
))
ON 1
FROM [Adventure Works]
WHERE([Product].[Product Categories].[Subcategory].&[26])
```

- « **Union (MDX)** » : Retourne un jeu produit par l'union de deux jeux, en conservant éventuellement les membres en double.

  Syntaxe : Union(Set_Expression1, Set_Expression2 [,...n][, ALL])

L'exemple suivant décrit le comportement de la fonction **Union** avec chaque syntaxe (Syntaxe standard, doublons supprimés):

```
SELECT Union
  ([Date].[Calendar Year].children
  , {[Date].[Calendar Year].[CY 2002]}
  , {[Date].[Calendar Year].[CY 2003]}
  ) ON 0
FROM [Adventure Works]
```