Mondrian学习总结

1.概述

1.1. Mondrian简介

Mondrian 是一个用 Java 写成的 OLAP 引擎，是开源项目 Pentaho 的一部分。它实现了 MDX 语言、XML 解析、JOLAP 规范。它从 SQL 和其它数据源读取数据并把数据聚集在内存缓存中，然后经过 Java API用多维的方式对结果进行展示，同时可以不写 SQL就能分析存储于 SQL 数据库的庞大数据集，可以封装 JDBC数据源并把数据以多维的方式展现出来。

JPivot 是 Mondrian 默认的表现层工具，它是一个 JSP 自定制的标签库，可以绘制 OLAP 分析图表。用户可以执行典型的 OLAP 导航，如下钻、切片。JPivot 使用 Mondrian 作为它的 OLAP 服务器但也支持 XML/A 数据源访问。它使用 WCF (Web Component Framework)框架 ，基于XML /XSLT来渲染 Web UI组件。

Mondrian 支持的数据库或数据仓库主要有：LucidDb、Oracle、 Access、MySQL、Sybase、Ingres、Postgres、Hypersonic、Teredata等。Mondrian 主要特点是对立方体进行了缓存，众所周知，缓存庞大的立方体对性能有很大的影响，但是 Mondrian利用 java语言的特点对这一点进行了很好的控制。其次由于 Mondrian基于 java语言，所以它能运行在不同的平台之上。

1.2. Mondrian架构

Mondrian从架构上可以分为四个层次：表现层、计算层、聚合层、存储层。

- 计算层：分析、验证、执行 MDX语句，先计算坐标轴，然后再计算每个单元格的值，从效率上的考虑，计算层批量从聚合层获取单元格数据集合。
- 聚合层：聚合层中缓存了多维查询结果，即单元格的数据集合，如果计算层所需要的数据不在缓存中，从存储层中进行查询获取数据并缓存。
- 存储层：采用关系数据库实现，一般采用星型模型构建，提供维表、事实表和聚合表。

系统部署结构上，可以分三层结构分开部署，将表现层部署在一台机器上，计算层和聚合层部署在第二台，存储层部署在第三台。

1.3. Mondrian特性

Mondrian用一个 XML文件来存储 Cube元数据信息，描述 Cube的逻辑视图。有以下特性：

- 支持共享维度。
- 支持虚拟 Cube。
- 支持一个维度有多个 Hierarchy。
- 支持标准维、雪化维和父子维。
- 支持自定义成员/命名集。

2.配置Mondrian

Mondrian下载地址：http://sourceforge.net/projects/mondrian

Mondrian配置过程大致如下：

文档目录：

相关文档：

- Mondrian Schema Workbench操作教程
- Mondrian Schema Workbench创建模型的操作录像

1、下载文件：从SourceForge网站上下载最新版本的mondrian-version.zip并解压。

2、准备关系数据库：默认可使用 Mondrian自带的 Access（MondrianFoodMart.mdb）数据库，需在机器上配置一个 ODBC数据源连接至此数据库；也可以配置其他数据库。

3、发布 Mondrian应用：

(1)准备一个 JDK 1.4.2或以上版本的 Web环境，如 Tomcat 5.5；

(2)从第(1)步中解压后的目录中找到 mondrian.war文件，解压至/Tomcat_Home/webapps/mondrian目录下；

(3)打开 Tomcat_Home/webapps/mondrian/WEB-INF/mondrian.properties文件，修改mondrian.jdbcDrivers为实际使用的数据库信息。

(4)打开 Tomcat_Home /webapps/mondrian/WEB-INF/datasources.xml文件，修改DataSourceInfo为实际使用的数据库信息。

(5)将 xalan.jar文件和实际使用的数据库的 JDBC驱动程序 jar文件拷贝至Tomcat_Home /webapps/mondrian/WEB-INF/lib/目录下。

4、启动Web application即可通过 http://localhost:8080/mondrian 访问了。

初次使用 Mondrian时，可采用其内置的 Access 数据库及默认的配置文件，无须修改太多内容即可完成配置。如果您要深入使用 Mondrian 时，有如下几个配置文件需要了解清楚：

1、mondrian.properties：属性文件，主要设置Mondrian的运行属性。

2、datasources.xml：数据源定义文件，Mondrian支持多个数据源、多个目录（Catalog）。

3、web.xml：Web应用配置文件，如配置MondrianXmlaServlet。默认可不作修改。

2.1. Mondrian Configuration 文件配置

Mondrian的一些属性值决定 Mondrian OLAP的运行属性，有两种办法指定属性：

- 作为 JVM属性值，运行 java.exe时使用“-D….”指定。
- 在其属性文件“mondrian.properties”设定。

> 注意：要使“mondrian.properties”生效，必需将其放在“当前工作目录”或者“/WEB-INF/classes”下。假如：使用Tomcat服务器，在/bin/start.bat启动Tomcat，就要将该文件放在/bin目录下，或者“/WEB-INF/classes”目录下。
>
> Mondrian.war默认将此文件放在/WEB-INF目录下，实际该文件是不生效的，需要修改目录。

mondrian.properties属性文件中属性说明：

属性	类型	默认值	描述
mondrian.query.limit	Int	40	同时并发查询最大数目。这个数目要小于关系数据库所配置的最大并发查询数。例如 Oracle的init.ora的“processes”参数。
mondrian.result.limit	Int	0	MDX查询结果集的最大行数，‘0’表示不限定。
mondrian.rolap.CachePool.costLimit	Int	10,000	Obsolete
mondrian.olap.case.sensitive	boolean	false	MDX分析器是否大小写敏感。
Aggregate tables			
mondrian.rolap.aggregates.Use	boolean	false	是否使用聚合表。
mondrian.rolap.aggregates.Read	boolean	false	是否读取聚合表。
mondrian.rolap.aggregates.generateSql	boolean	false	是否打印生成 aggregate tables.的 SQL语句。

2.2. Datasource.xml文件配置

为使Mondrian作为一个XMLA提供者，需要配置datasource.xml文件。Mondrian支持多个数据源，多个目录（Catalog）。例如：

```
<?xml version="1.0" ?>
- <DataSources>
  - <DataSource>
      <DataSourceName>MondrianFoodMart</DataSourceName>
      <DataSourceDescription>FoodMart 2000 Data Warehouse From MS Analysis
        Services</DataSourceDescription>
      <URL>http://localhost:8080/mondrian/xmla</URL>
      <DataSourceInfo>Provider=mondrian; Jdbc=jdbc:odbc:MondrianFoodMart;
        JdbcDrivers=sun.jdbc.odbc.JdbcOdbcDriver</DataSourceInfo>
    - <Catalogs>
        <Catalog name="FoodMart">/WEB-INF/schema/FoodMart.xml</Catalog>
        <Catalog name="Markating">/WEB-INF/schema/Marketing.xml</Catalog>
      </Catalogs>
      <ProviderType>MDP</ProviderType>
      <AuthenticationMode>Unauthenticated</AuthenticationMode>
    </DataSource>
  - <DataSource>
      <DataSourceName>PostgreSQLTest</DataSourceName>
      <DataSourceDescription>Test Data Warehouse On PostgreSQL</DataSourceDescription>
      <URL>http://localhost:8080/mondrian/xmla</URL>
      <DataSourceInfo>Provider=mondrian; Jdbc=jdbc:postgresql://localhost/olap;
        JdbcDrivers=org.postgresql.Driver; JdbcUser=pgsql; JdbcPassword=pgsql</DataSourceInfo>
    - <Catalogs>
        <Catalog name="Test">/WEB-INF/schema/TestPgsql.xml</Catalog>
      </Catalogs>
      <ProviderName>Mondrian</ProviderName>
      <ProviderType>MDP</ProviderType>
      <AuthenticationMode>Unauthenticated</AuthenticationMode>
    </DataSource>
  </DataSources>
```

Mondrian需要在　datasource.xml指定：

属性	描述
DataSourceName	数据源的名称
DataSourceDescription	数据源的描述
URL	http请求　URL地址
DataSourceInfo	数据源的提供者（必须是 Provider=mondrian），JDBC连接字符串
Catalogs	多个目录，每个目录须指定名称以及对应的Schema文件的位置
ProviderName	值必须为　Mondrian
ProviderType	值必须为　MDP
Authentication	认证方式，可以是：Unauthenticated、Authenticated、Integrated

Mondrin要在　datasource.xml或者使用其标记库要配置连接字符串。连接字符串可有以下属性：

属性	必选	描述
Provider	是	值必须为　mondrian
Jdbc	二选一	连接 JDBC数据库的　URL，不同数据库写法不同。
DataSource		数据源类名，必须实现javax.sql.DataSource接口。
JdbcUser	否	登录数据库的用户名。假如“Jdbc”参数的 URL中已指定　user、password则不需要这个参数。 但是有时使用已指定 user、password的　JDBC的 URL不能连接数据库。这时必需使用 JdbcUser、JdbcPassword这两个参数。
JdbcPassword	否	登录数据库的密码。假如“Jdbc”参数的 URL中已指定　user、password，则不需要这个参数。
JdbcDrivers	是	JDBC驱动程序名，不同数据库写法不同。
Catalog	是	Schema文件的位置。

PoolNeeded	否	是否需要接接池。 如果使用“DataSource”参数，那么连接池已经存在。
Role	否	用户的角色。

2.3. Web.xml 文件配置

web.xml文件一般默认不用修改。有如下两种情况时，需要修改：

1、当访问含中文的 Cube出现乱码时，需指定字符集参数：CharacterEncoding=GBK。

2、要指定 datasources.xml文件的位置：此时需写绝对路径。

不指定 DataSourcesConfig参数时，系统就会默认查找 WEB-INF/datasources.xml文件；如指定此参数，则 param-value需写为绝对路径，如：

```
D:\ Tomcat 5.5.20\webapps\mondrian\WEB-INF\datasources.xml
```

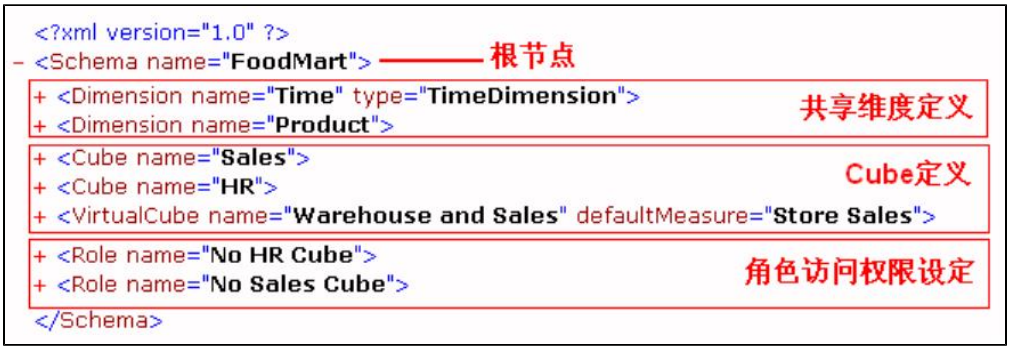
3.定义多维模型（Schema）

配置好 Mondrian后，重点就是要学习如何定义多维模型，其实也就是编写 Schema文件。

Schema文件是 Mondrian应用中最核心、最关键的文件，它基于 XML格式，定义了一个多维数据库，在这个文件中形成逻辑模型和数据库物理模型的对应。逻辑模型定义好后，即可以书写 MDX语言实现多维数据查询。这个逻辑模型实际上提供了这几个概念：立方体（Cubes）、维度（Dimensions）、层次（Hierarchies）、级别（Levels）、和成员（Members）。

3.1. Schema文件

Schema文件结构如下图所示：



根节点为 Schema，其 name需与 datasource.xml文件中配置的 Catalog name保持一致。在Schema下，可以定义多个共享维 Dimension节点，可以定义多个 Cube节点和虚拟 Cube节点，并可以定义多个角色设置对上述 Cube的访问权限。

3.2. Cube定义

3.2.1.普通 Cube

Cube节点如下图所示，首先须指定事实表，然后定义维度 Dimension和度量值 Measure以及计算成员。

```
- <Cube name="Sales">                                                    Cube定义
    <Table name="sales_fact_1997" /> 指定事实表
    <DimensionUsage name="Time" source="Time" foreignKey="time_id" /> 引用共享维
    <DimensionUsage name="Product" source="Product" foreignKey="product_id" />
  + <Dimension name="Gender" foreignKey="customer_id">
  + <Measure name="Store Sales" column="store_sales" aggregator="sum" formatString="#,###.00">
  + <Measure name="Store Cost" column="store_cost" aggregator="sum" formatString="#,###.00">
  + <CalculatedMember name="Profit" dimension="Measures">                  定义度量值
    </Cube>        定义计算成员
```

3.2.2.虚拟 Cube

Mondrian可以通过已有的 Cube的连接，定义一个虚拟 cube。例如：

```
- <VirtualCube name="Warehouse and Sales" defaultMeasure="Store Sales">
    <VirtualCubeDimension cubeName="Sales" name="Gender" />
    <VirtualCubeDimension name="Store" />                                  虚拟Cube定义
    <VirtualCubeDimension name="Time" />
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Sales Count]" />
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Store Cost]" />
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Store Sales]" />
  </VirtualCube>
```

3.3. Dimension定义

3.3.1.普通维

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

属性	含义
Dimension name	定义一个维度，给出维度名
foreingKey	事实表的外键
Hierarchy	定义一个层次（可以定义多个 Hierarchy）
hasAll	表示是否有“ALL”成员
primaryKey	维度表的主键
Table name	维度表名
Level name	定义一个级别，给出名称
column	级别对应维度表的列
uniqueMembers	维度表的成员是否唯一

3.3.2.时间维

```
- <Dimension name="Time" type="TimeDimension" primaryKey="time_id">
  - <Hierarchy hasAll="false" primaryKey="the_date">
      <Table name="time_by_day" />
      <Level name="Year" column="the_year" type="String" uniqueMembers="true" levelType="TimeYears" />
      <Level name="Quarter" column="quarter" uniqueMembers="false" levelType="TimeQuarters" />
      <Level name="Month" column="month_of_year" uniqueMembers="false" type="String" levelType="TimeMonths" />
      <Level name="Day" column="d_name" uniqueMembers="false" type="String" levelType="TimeDays" />
    </Hierarchy>
  - <Hierarchy hasAll="true" name="Weekly" primaryKey="time_id" defaultMember="[Time.Weekly].[All Time.Weeklys].[1997]">
      <Table name="time_by_day" />
      <Level name="Year" column="the_year" type="Numeric" uniqueMembers="true" levelType="TimeYears" />
      <Level name="Week" column="week_of_year" type="Numeric" uniqueMembers="false" levelType="TimeWeeks" />
      <Level name="Day" column="day_of_month" uniqueMembers="false" type="Numeric" levelType="TimeDays" />
    </Hierarchy>
  </Dimension>
```

时间维同样拥有普通维的各种属性，除此之外，还有如下属性：

属性	含义
type="TimeDimension"	声明这是一个时间维
levelType	定义时间维级别的类型
Hierarchy	多个层次并列申明

3.3.3.雪花维（snowflake schemas）

以 Sales的 producrt 维度为例，如下图所示：

```
- <Dimension name="Product">
  - <Hierarchy hasAll="true" primaryKey="product_id" primaryKeyTable="product">
    - <Join leftKey="product_class_id" rightKey="product_class_id">
        <Table name="product" />
        <Table name="product_class" />
      </Join>
      <Level name="Product Family" table="product_class" column="product_family" uniqueMembers="true" />
      <Level name="Product Department" table="product_class" column="product_department" uniqueMembers="false" />
      <Level name="Product Category" table="product_class" column="product_category" uniqueMembers="false" />
      <Level name="Product Subcategory" table="product_class" column="product_subcategory" uniqueMembers="false" />
      <Level name="Brand Name" table="product" column="brand_name" uniqueMembers="false" />
      <Level name="Product Name" table="product" column="product_name" uniqueMembers="true" />
    </Hierarchy>
  </Dimension>
```

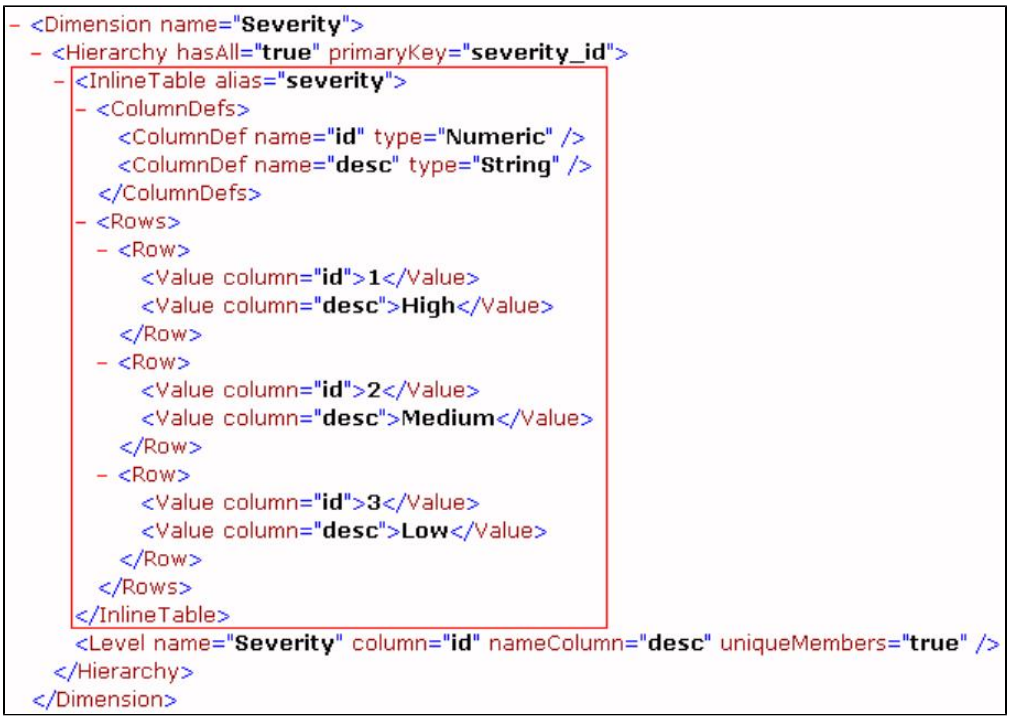
雪花维与普通维度区别有两点：

- 通过 primaryKeyTable定义维度表
- 在<Join>中定维度表间的连接。

3.3.4.父子维度

为提高处理性能，Mondrian建议要建立“Closure tables”表，记录父子关系和深度。其创建方法，用父子维度的定义请看 Mondrian官方文档。

3.3.5.利用内联表（Inline tables），建立虚似维度



在上例中，我们在<InlineTable>中建立虚拟表格severity，然后，我们就可以定义一个维度<Dimension name="Severity">。

id	desc
1	High
2	Medium
3	Low

3.3.6. Degenerate dimensions（退化维度）

例如：有一事实表：

product_id	time_id	payment_method	customer_id	store_id	item_count	dollars
55	20040106	Credit	123	22	3	$3.54
78	20040106	Cash	89	22	1	$20.00
199	20040107	ATM	3	22	2	$2.99
55	20040106	Cash	122	22	1	$1.18

要建立维度：payment_method；由于 payment_method十分简单，不需要建立一个维度表：

payment_method
Credit
Cash
ATM
Cash

我们可建立一个这样的 Degenerate dimension：

```
- <Dimension name="Payment method">
  - <Hierarchy hasAll="true">
      <!--  No table element here. Fact table is assumed.   -->
      <Level name="Payment method" column="payment_method" uniqueMembers="true" />
    </Hierarchy>
  </Dimension>
```

3.3.7. 共享维度（Shared dimensions）

```
<Schema name="FoodMart">
- <Dimension name="Time" type="TimeDimension">
  - <Hierarchy hasAll="false" primaryKey="time_id">                    定义共享维
      <Table name="time_by_day" />
      <Level name="Year" column="the_year" type="String" uniqueMembers="true" levelType="TimeYears" />
      <Level name="Quarter" column="quarter" uniqueMembers="false" levelType="TimeQuarters" />
      <Level name="Month" column="month_of_year" uniqueMembers="false" type="String" levelType="TimeMonths" />
      <Level name="Day" column="d_name" uniqueMembers="false" type="String" levelType="TimeDays" />
    </Hierarchy>
  </Dimension>
+ <Dimension name="Product">
- <Cube name="Sales">
    <Table name="sales_fact_1997" />
    <DimensionUsage name="Time" source="Time" foreignKey="time_id" />
    <DimensionUsage name="Product" source="Product" foreignKey="product_id" />   使用共享维
  + <Measure name="Store Sales" column="store_sales" aggregator="sum" formatString="#,###.00">
  </Cube>
```

共享维度在 Cube定义之前定义，维度属性同上面的普通维/时间维。说明两点：

- 在共享维度的定义<Dimension>中，不需要定义外键“foreignKey”属性。
- 而在<Cube>中使用共享维<DimensionUsage>时指定外键“foreignKey”属性。

3.4. Measures定义

3.4.1. 普通度量值

```
<Measure name="Unit Sales" column="unit_sales" aggregator="sum" datatype="Integer" formatString="#,###" />
```

属性	含义
Name	指定指标名称；
Column	指定数据来源的事实表的列。
Aggregator	聚合方式。通常是"sum"或"count"、"mix"、"max"、"avg"、"distinct count"
Datatype	指定数据类型，可以是"String", "Integer"、"Numeric"(默认值)
formatString	定义数据显示格式。
Caption	指定 caption

3.4.2. 通过SQL语句定义度量值

指标数据的来源可以从 SQL语名得到（针对不同的数据库类型写不同的 SQL语句），例如：

```
- <Measure name="Promotion Sales" aggregator="sum" formatString="#,###.00">
  - <MeasureExpression>
      <SQL dialect="access">Iif("sales_fact_1997"."promotion_id" = 0, 0, "sales_fact_1997"."store_sales")</SQL>
      <SQL dialect="oracle">(case when "sales_fact_1997"."promotion_id" = 0 then 0 else "sales_fact_1997"."store_sales" end)</SQL>
      <SQL dialect="db2">(case when "sales_fact_1997"."promotion_id" = 0 then 0 else "sales_fact_1997"."store_sales" end)</SQL>
    </MeasureExpression>
  </Measure>
```

3.5. 维度扩展

3.5.1. 定义成员属性

```
- <Dimension name="Store">
  - <Hierarchy hasAll="true" primaryKey="store_id">
      <Table name="store" />
      <Level name="Store Country" column="store_country" uniqueMembers="true" />
      <Level name="Store State" column="store_state" uniqueMembers="true" />
      <Level name="Store City" column="store_city" uniqueMembers="false" />
    - <Level name="Store Name" column="store_name" uniqueMembers="true">
        <Property name="Store Type" column="store_type" />
        <Property name="Store Manager" column="store_manager" />
      </Level>
    </Hierarchy>
  </Dimension>
```

成员属性

3.5.2. 定义计算成员

Mondrian可以在 Cube中定义计算成员，如下图所示。这样计算成员就可以像普通的[Measures]成员一样，在 MDX语句中使用。

```
- <CalculatedMember name="Profit Growth" dimension="Measures" formula="([Measures].
    [Profit] - [Measures].[Profit last Period]) / [Measures].[Profit last Period]"
    visible="true" caption="Gewinn-Wachstum">
    <CalculatedMemberProperty name="FORMAT_STRING" value="0.0%" />
  </CalculatedMember>
```

3.5.3. 定义命名集（Named sets）

Mondrian可以在 Cube中定义一个命名集。例如：

```
- <Cube name="Warehouse">
    <Table name="inventory_fact_1997" />
    <DimensionUsage name="Store" source="Store" foreignKey="store_id" />
    <DimensionUsage name="Time" source="Time" foreignKey="time_id" />
    <DimensionUsage name="Product" source="Product" foreignKey="product_id" />
    <Measure name="Store Invoice" column="store_invoice" aggregator="sum" />
    <Measure name="Supply Time" column="supply_time" aggregator="sum" />
  + <Measure name="Warehouse Profit" aggregator="sum">
  + <CalculatedMember name="Average Warehouse Sale" dimension="Measures">
  - <NamedSet name="Top Sellers">
      <Formula>TopCount([Warehouse].[Warehouse Name].MEMBERS, 5, [Measures].
        [Warehouse Sales])</Formula>
    </NamedSet>
  </Cube>
```

这样可以在 MDX中使用集合[Top Sellers]

```
SELECT
{[Measures].[Warehouse Sales]} ON COLUMNS,
{[Top Sellers]} ON ROWS
FROM [Warehouse]
WHERE [Time].[Year].[1997]
```

3.6. Role权限控制

在 schema文件中，可以定义角色（Role），并在连接数据库时使之生效，实现存取控制。以下一个角定义的例子。

```
- <Role name="California manager">
  - <SchemaGrant access="none">
    - <CubeGrant cube="Sales" access="all">
      - <HierarchyGrant hierarchy="[Store]" access="custom" topLevel="[Store].[Store Country]">
          <MemberGrant member="[Store].[USA].[CA]" access="all" />
          <MemberGrant member="[Store].[USA].[CA].[Los Angeles]" access="none" />
        </HierarchyGrant>
      + <HierarchyGrant hierarchy="[Customers]" access="custom" topLevel="[Customers].[State
          Province]" bottomLevel="[Customers].[City]">
        <HierarchyGrant hierarchy="[Gender]" access="none" />
      </CubeGrant>
    </SchemaGrant>
  </Role>
- <Role name="No HR Cube">
  - <SchemaGrant access="all">
      <CubeGrant cube="HR" access="none" />
    </SchemaGrant>
  </Role>
```

节点	含义
<SchemaGrant>	定义能否存取 Schema文件的对象。以上例子的值为‘none’，表示这个角色只能存取“Sales”cube，由于其在<CubeGrant>中定义为可存取。
<CubeGrant>	定义能否存取 Cube。
<HierarchyGrant>	定义能否存取hierarchy。access值为‘custom’、‘all’、‘none’。如果值为‘custom’可以定义topLevel、bottomLevel属性值，决定可见的最高级别、最低级别。
<MemberGrant>	定义能否访问Member。access值为‘none’，‘all’。子成员从父成员继承access，授权按成员的级别顺序，同时<HierarchyGrant>的topLevel、bottomLevel属性值也影响成员是否可见。

注：要使角色生效，还要在datasource.xml、web.xml的JDBC连接字符串，指定Role的角色。或者使用编程，Connection.setRole(Role)方法。

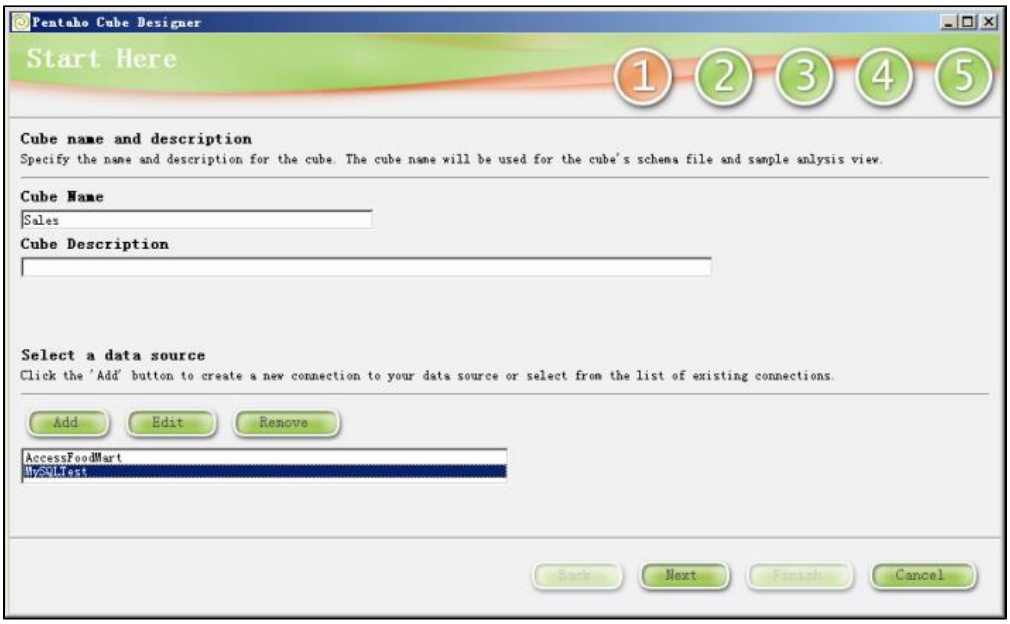
4.相关工具

4.1. CubeDesigner

Mondrian cubes创建有多种办法，最原始的办法就是用手写Schema文件，也有界面化的Cube创建平台。Cube Designer是Pentaho提供的界面化的Cube创建平台。Cube Designer使用过程大致如下：

1、 下载CubeDesigner-0.7.2.0_Win32.zip，解压后运行CubeDesigner.exe。

2、 创建new cube，定义数据源。定义Cube Name——选择数据源——编辑JNDI Name——Driver——Connect String——Username/Password，然后Test连接是否成功。



3、 定义数据模型和Table Mapping。选择数据库——选择Table/View——双击抽取目标数据——定义Mapping——选择要显示的字段。在这步数据模型建立的查询中，还可以定义where condition，group by，having，order by等。

4、 选择事实表和定义度量

5、 创建Dimensions及hierarchy，level

6、 生成Schema及Publish或者保存。

> 注意：在最后一步中，如果定义的Schema文件有错误，则看不到XML Source并且无法保存结果！要注意维度和度量的定义！

7、 至此，在保存的目录下就生成三个文件：***.mondrian.xml，***.properties，***.xaction。其中***.mondrian.xml就是我们所需要的Cubes。

4.2. Workbench

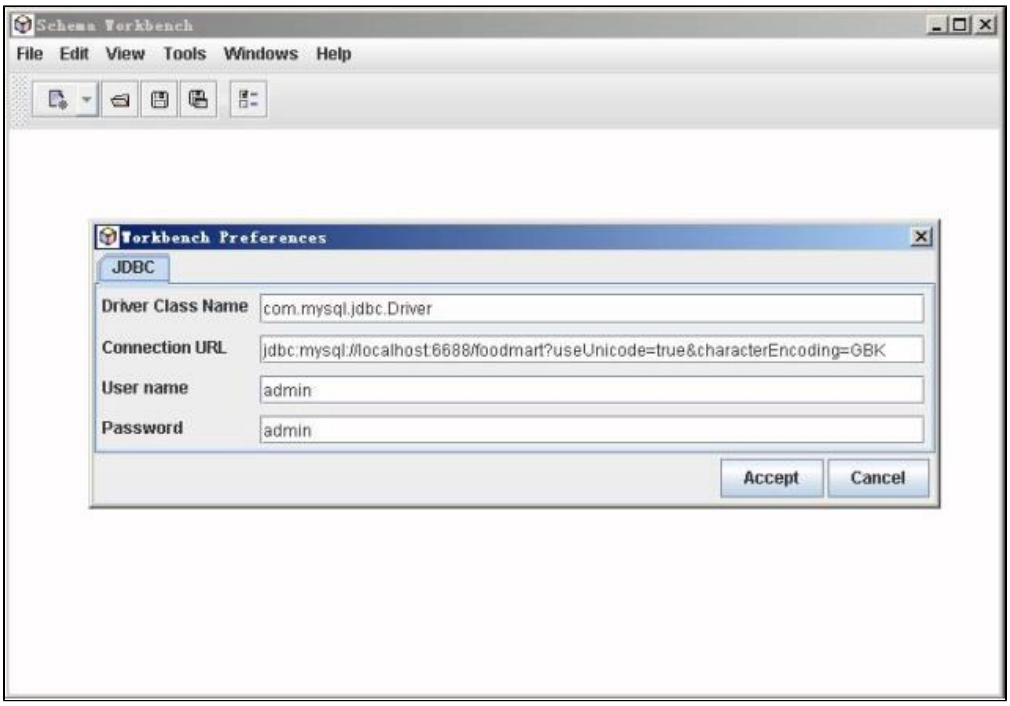
Workbench是另一个界面化的Cube创建平台。目前版本为workbench-2.3.2.9247。该平台有三个功能：生成Schema，执行MDX和JDBC Explore。

> 下载地址：http://sourceforge.net/project/showfiles.php?group_id=35302

Workbench使用过程大致如下：

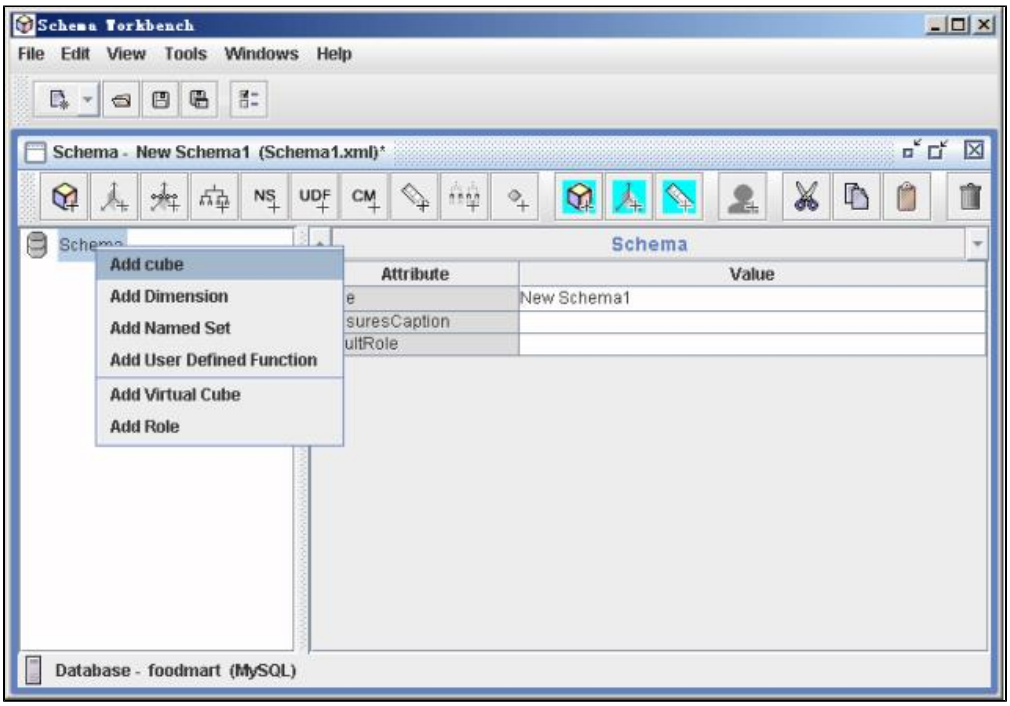
1、 下载workbench-2.3.2.9247.zip后解压缩，运行workbench.bat文件；

2、 配置数据库：tools-->Preferences，在弹出的对话框处填写JDBC连接信息。



注意：要把相应的JDBC驱动包拷贝到workbench-版本号\lib目录下。

3、 定制Cube：file-->new-->schema出来的界面如下图：



4、 根据界面菜单添加Cube、Dimenson等，逐步完成制作***.mondrian.xml(立方体)。

4.3. Cmdrunner

Mondrian提供一个命令行工具包　cmdrunner.jar。

1、 下载所需的 xalan.jar、math.jar包，设置环境变量，在执行 mondrian源文件的/src/build.bat －cmdrunner,编译成功；生成 cmdrunner.jar。

2、 配置 mondrian.properties文件，设置 mondrian.catalogURL（这是必需一个 URL如：file:///d:\\cmdrunner\\FoodMart.xml）、mondrian.test.jdbcURL、mondrian.jdbcDrivers、mondrian.rolap.aggregates.generateSql等属性。

3、 运行包 java -jar cmdrunner.jar.就可以用命令输入 MDX语句，“=”表示执行输入 MDX。

5.常见问题

5.1. 如何优化 Cube访问性能

Mondrian是一个 ROLAP，其性能主要由其数据库服务器决定。要进行性能优化：一个办法是调优其数据库服务器；第二个方法是使用聚合表（Aggregate Tables）。通过创建大量的临时聚合表，当用户进行比较高级的分析时，不用访问数据量庞大的基础事实表，只需要在已经形成的实体化视图或聚合表上作进一步的聚合就可以了，这样能够大大提高查询分析的效率，并且减少占用的系统资源。

1、 每个 Cube可以有 0个或多个 Aggregate Tables。是否需要 Aggregate Tables，要考虑两个方面：

- 该应用的 MDX的查询频度。例如要经常查询各类产品每个月的销售量、销售价，即可建立各类产品、各个月的聚合表。
- 级别和这个级别的上一个级别有一个很大的跳跃，成员有很多子成员。例如从年份级别直接到了时间级别。

2、 Mondrian如何识别聚合表（Aggregate Tables）。

- 根据命名规则识别聚合表。判别一个数据表是否Aggregate Tables，Mondrian首先使用规则匹配Aggregate Tables和 fact tables的名称。默认的规则是：agg_.+_${fact_table_name}。（在mondrian.jar的 defaultrules.xml中定义）。也就是如果fact tables是sales_fact_1997，那么agg_l_05_sales_fact_1997、agg_c_14_sales_fact_1997会初步认为是聚合表。然后匹配事实表中的每一列。默认的规则：${hierarchy_name}_${level_name}, ${hierarchy_name}_${level_column_name},${usage_prefix}${level_column_name},${level_column_name}。匹配成功后就会把符合规则数据表的认为是这个事实表的的聚合表。
- 准确无误地定义聚合表。例如以下例子，使用 AggExclude准确无误地把"agg_c_14_sales_fact_1997"定义为不是聚合表，虽然它符合规则。在 AggName定义agg_c_special_sales_fact_1997为事实表，它名称不需要符合规则。AggIgnoreColumn忽略聚合表的列。

```
- <Cube name="Sales">
  - <Table name="sales_fact_1997">
      <AggExclude name="agg_c_14_sales_fact_1997" />
      <AggExclude name="agg_lc_10_sales_fact_1997" />
      <AggExclude name="agg_pc_10_sales_fact_1997" />
    - <AggName name="agg_c_special_sales_fact_1997">
        <AggFactCount column="FACT_COUNT" />
        <AggIgnoreColumn column="admin_one" />
        <AggIgnoreColumn column="admin_two" />
        <AggForeignKey factColumn="product_id" aggColumn="PRODUCT_ID" />
        <AggForeignKey factColumn="customer_id" aggColumn="CUSTOMER_ID" />
        <AggForeignKey factColumn="promotion_id" aggColumn="PROMOTION_ID" />
        <AggForeignKey factColumn="store_id" aggColumn="STORE_ID" />
        <AggMeasure name="[Measures].[Unit Sales]" column="UNIT_SALES_SUM" />
        <AggMeasure name="[Measures].[Store Cost]" column="STORE_COST_SUM" />
        <AggMeasure name="[Measures].[Store Sales]" column="STORE_SALES_SUM" />
        <AggLevel name="[Time].[Year]" column="TIME_YEAR" />
        <AggLevel name="[Time].[Quarter]" column="TIME_QUARTER" />
        <AggLevel name="[Time].[Month]" column="TIME_MONTH" />
      </AggName>
    - <AggPattern pattern="agg_sales_fact_1997_.*">
        ....
        <AggExclude name="agg_sales_fact_1997_olddata" />
        <AggExclude pattern="agg_sales_fact_1997_test.*" />
      </AggPattern>
    </Table>
```

> 注：在实施过程中，应使用第二种‘准确无误’地定义聚合表这种方法。

3、 如何建立聚合表。根据 Schema和聚合表的定义，就要在数据库中建立聚合表。聚合表的定义、如何生成建立聚合表的 SQL语句以及更详细的信息请查阅 Mondrian文档。Mondrian提供了根据 MDX查询，生成聚合表的 SQL语句。设置 Mondrian属性：

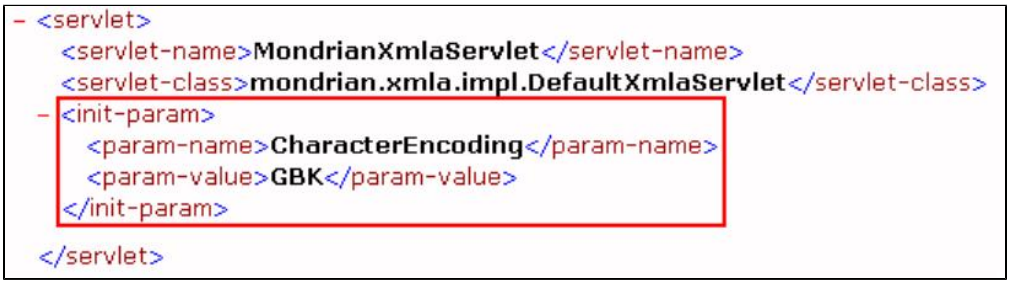
```
mondrian.rolap.aggregates.generateSql=true
mondrian.rolap.generate.formatted.sql=true
```

执行一个 MDX查询，就会在　console中显示生成聚合表的 SQL，提供开发人员参考。

5.2. Cube访问时出现乱码

如果通过前端工具查询 Mondrian做的　Cube出现，则需要检查 datasources.xml和　web.xml是否设置字符集。

- Web.xml：增加编码　CharacterEncoding=GBK



- datesource.xml：申明数据源时要增加编码申明，并注意Jdbc值要加上引号。如下：

```
<DataSourceInfo>Provider=mondrian;Jdbc="jdbc:informix-sqli://128.128.96.139:20003/atmvh:
INFORMIXSERVER=online9;NEWLOCALE=zh_cn,en_us;NEWCODESET=GBK,8859-1,819";
JdbcUser=atmvh;JdbcPassword=atmv123;JdbcDrivers=com.informix.jdbc.IfxDriver;Catalog=/WEB-INF/queries/ATMCube.
xml</DataSourceInfo>
```

5.3. 维度有‘跳级’的情况会出错

在 Store ragged维度的维度表里：

Strore name	city	state	country
Store 11	wanshington	USA	USA
Store 12	Ralm	CA	Canada

使用 discover方法查找　[wanshington]成员，返回的成员名：

```
[store ragged].[all storeraggeds].[USA].[USA].[wanshington]
```

但是使用以上的成员名做 MDX查询　select [store ragged].[all storeraggeds].[USA].[USA].[wanshington] …　会报错，说此成员不存在。若使有[store ragged].[all storeraggeds].[USA].[wanshington]，却可以。