1. O que é um esquema?

Um esquema define uma base de dados multidimensional. Ele contém um modelo lógico, que consiste em cubos, hierarquias e membros, e um mapeamento deste modelo para um modelo físico.

O modelo lógico consiste nas construções usadas para escrever consultas em linguagem MDX: cubos, dimensões, hierarquias, níveis e membros.

O modelo físico é uma fonte dos dados representada através do modelo lógico. É tipicamente um esquema em estrela, que é um conjunto de tabelas em um banco de dados relacional; depois, veremos exemplos de outros tipos de mapeamentos.

2. Arquivos de Esquema

Esquemas de Mondrian são representados em um arquivo XML. Um esquema de exemplo, que contém quase todas as construções que discutimos aqui, é fornecido como demonstração chamado FoodMart.xml na distribuição Mondrian. O conjunto de dados para preencher esse esquema também pode ser localizado neste link.

Atualmente, a única maneira de criar um esquema é editar um arquivo XML em um editor de texto. A sintaxe XML não é muito complicada, e isso não é tão difícil quanto parece, especialmente se você usar o esquema FoodMart como exemplo orientador.

A estrutura do documento XML é como se segue:

```
<Schema>
	<Cube>
		<Table>
			<AggName>
				aggElements
			<AggPattern>
				aggElements
		<Dimension>
			<Hierarchy>
				relation
				<Closure/>
				<Level>
				<KeyExpression>
					<SQL/>
				<NameExpression>
					<SQL/>
				<CaptionExpression>
					<SQL/>
				<OrdinalExpression>
					<SQL/>
				<ParentExpression>
					<SQL/>
				<Property>
					<PropertyExpression>
						<SQL/>
		<DimensionUsage>
		<Measure>
			<MeasureExpression>
				<SQL/>
			<CalculatedMemberProperty/>
		<CalculatedMember>
			<Formula/>
			<CalculatedMemberProperty/>
		<NamedSet>
			<Formula/>
	<VirtualCube>
		<CubeUsages>
			<CubeUsage>
		<VirtualCubeDimension>
		<VirtualCubeMeasure>
	<Role>
		<SchemaGrant>
			<CubeGrant>
				<DimensionGrant>
				<HierarchyGrant>
					<MemberGrant/>
		<Union>
```

```
                    <RoleUsage/>
        <UserDefinedFunction/>
        <Parameter/>

relation ::=
        <Table>
                <SQL/>
        <View>
                <SQL/>
        <InlineTable>
                <ColumnDefs>
                        <ColumnDef>
                <Rows>
                        <Row>
                                <Value>
        <Join>
                relation

aggElement ::=
        <AggExclude>
        <AggFactCount>
        <AggIgnoreColumn>
        <AggForeignKey>
        <AggMeasure>
        <AggLevel>
```

Nota: A ordem dos elementos XML é importante. Por exemplo, o elemento <UserDefinedFunction> tem que ocorrer dentro do elemento <Schema> após todas as coleções de elementos <Cube>, <VirtualCube>, <NamedSet> e <role>. Se você incluí-lo antes do primeiro <Cube> elemento, o resto do esquema será ignorada.

O conteúdo de cada elemento XML é descrito no Apêndice A e no esquema XML.

2.1 Anotação (Annotation)

A maioria dos elementos (esquema(schema), cubo(cube), cubo virtual(virtual cube), dimensão compartilhada(shared dimension), dimensão(dimension), hierarquia(hierarchy), nível(level), métrica(measure), membro calculado(calculated member)) suportam anotações. Uma anotação é uma maneira de associar uma propriedade definida pelo usuário com um elemento de metadados, e em particular, permite adicionar metadados sem estender o esquema Mondrian oficial.

Cria-se um elemento <Annotations> como um filho do elemento você deseja anotar (geralmente é o primeiro elemento filho, mas verifique a definição de esquema para mais detalhes), então inclua a quantidade de elementos <Annotation> necessários. Elementos <Annotation> deve ser exclusivo no seu elemento. Se você estiver adicionando anotações para apoiar uma determinada ferramenta que você mantenha, escolha nomes de anotação com cuidado, para garantir que eles não conflitem com as anotações utilizadas por outras ferramentas.

O exemplo a seguir mostra "Autor" e as anotações "Data" ligados a um objeto <Schema>.

```
<Schema name="Rock Sales">
	<Annotations>
		<Annotation name="Author">Fred Flintstone</Annotation>
		<Annotation name="Date">10,000 BC</Annotation>
	</Annotations>
	<Cube name="Sales">
	…
```

3. Modelo Lógico

Os componentes mais importantes de um esquema são cubos, métricas e dimensões:

- Um cubo é um conjunto de dimensões e métricas em áreas específicas.
- Uma métrica é uma quantidade que você está interessado em medir, por exemplo, as vendas unitárias de um produto, ou preço de custo dos itens de estoque.
- Uma dimensão é um atributo ou conjunto de atributos, através do qual você pode dividir medidas em subcategorias. Por exemplo, você pode querer quebrar as vendas de produtos pela sua cor, o sexo do cliente e a loja em que o produto foi vendido; cor, sexo, e loja são todas as dimensões.

Vejamos a definição XML de um esquema simples.

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

Este esquema contém um único cubo, chamado de "Vendas". O cubo de vendas tem duas dimensões, "Time" e "Gender", e quatro medidas, "Unity Sales", "Store Sales", "Store Cost", e "Profit".

Podemos escrever uma consulta MDX sobre este esquema:

```
SELECT {[Measures].[Unit Sales], [Measures].[Store Sales]} ON COLUMNS,
{descendants([Time].[1997].[Q1])} ON ROWS
```

```
FROM [Sales]
WHERE [Gender].[F]
```

Esta consulta refere-se ao cubo Sales ([Sales]), cada uma das dimensões [Métricas], [Time], [Sexo], e vários membros dessas dimensões. Os resultados são os seguintes:

[Time]	[Métricas].[Unit Sales]	[Métricas].[Store Sales]
[1997].[Q1]	0	0
[1997].[Q1].[Jan]	0	0
[1997].[Q1].[Feb]	0	0
[1997].[Q1].[Mar]	0	0

Agora vamos olhar para a definição de esquema com mais detalhes.

3.1 Cubo

O cubo (veja <Cube>) é o nome da coleção de métricas e dimensões. A única coisa que as métricas e as dimensões têm em comum é a tabela verdade, aqui "sales_fact_1997". Como veremos, a tabela de fatos contém as colunas a partir da qual são calculadas as métricas, e contém referências para as tabelas que possuem as dimensões.

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

A tabela de fatos é definida usando o elemento <table>. Se a tabela de fato não é no esquema padrão, você pode fornecer um esquema explícito usando o atributo "esquema", por exemplo.

```
<Table schema=" dmart" name="sales_fact_1997"/>
```

Você também pode usar o <View> construção para construir instruções SQL mais complicadas. O <Join> construção não é suportada para tabelas de fatos.

3.2 Métricas

O cubo Sales define várias métricas, incluíndo "Unit Sales" e "Store 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"/>
```

Cada medida (veja <Métrica>) tem um nome, uma coluna na tabela de fatos, e um agregador. O agregador é geralmente "sum", mas "count", "min", "max", "avg" e "distinct-count" também estão autorizados; "distinct-count" tem algumas limitações se o cubo contém uma hierarquia pai-filho.

A opção do atributo do tipo dados defini os valores das celulas no cache do Mondrian e como eles são devolvidos via XML para análise. O Tipo de dados podem ser "String", "Integer", "Numeric", "Boolean", "Date", "Time", and "Timestamp". O padrão das métricas é “Numeric”, com exceção de “count” e "distinct-count" que são “Integer”.

A opção formatString especifica como o valor deve ser impresso. Aqui, optamos por “Sales Unity” sem casas decimais (uma vez que é um número inteiro), e as “Store Sales” com duas casas decimais (uma vez que é um valor de moeda). O simbolos ‘.’,’,’ são dependem do idioma, por isso, se você estava executando em português, as vendas da loja pode aparecer como "48.123,45". Você pode conseguir mais efeitos usando usingadvanced para o formato “String”.

A métrica pode ter um atributo de legenda a ser devolvido pelo método Member.getCaption() em vez do nome. A definição de uma “Label” específica faz sentido se pretender usar as letras especiais (por exemplo Σ ou Π) devem ser apresentadas:

```
<Measure name="Sum X" column="sum_x" aggregator="sum" caption="&#931; X"/>
```

Em vez de vir de uma coluna, uma medida pode vir de um leitor de celula, ou uma medida pode usar uma expressão SQL para calcular o seu valor. A medida "Promotion Sales" é um exemplo disso.

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

Neste caso, as vendas apenas são incluídos no somatório se eles correspondem a uma promoção de venda. Qualquer expressões SQL podem ser usadas, incluindo sub consultas. No entanto, o banco de dados subjacente deve ser capaz de suportar essa expressão SQL no contexto de um agregado. Variações na sintaxe entre diferentes bancos de dados é tratada pelo especificadamente pelo dialeto definido na tag SQL.

A fim de proporcionar uma formatação específica dos valores de células, uma medida pode usar um formatador de célula.

3.3 Dimensões(Dimensions), Hierarquias(Hierarchies) e Niveis(Levels)

Algumas definiões:

- Um membro é um ponto dentro de uma determinada dimensão demarcado por um conjunto de valores de atributos. A hierarquia de gênero tem os dois membros 'M' e 'F'. 'San Francisco', 'California' e 'EUA' são todos membros da hierarquia loja.
- Uma hierarquia é um conjunto de membros organizados dentro de uma estrutura para a análise mais conveniente. Por exemplo, a hierarquia de armazenamento consiste no nome da loja, cidade, estado e nação. A hierarquia permite-lhe formar subtotais intermediários: o subtotal para um estado é a soma dos subtotais de todas as cidades de um determinado estado, cada um dos quais é a soma dos subtotais das lojas naquela cidade.
- Um nível(level) é uma coleção de membros que têm a mesma distância a partir da raiz da hierarquia.
- Uma dimensão é uma coleção de hierarquias discriminadas na mesmo atributo da tabela fato (por exemplo, o dia em que a venda ocorreu).

Por razões de uniformidade, métricas são tratadas como membros de uma dimensão especial chamadas chamada “'Measures'”.

Um exemplo

Vamos olhas para uma dimensão simples.

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

Esta dimensão é gerada por uma única hierarquia, que consiste de um único nível de chamada de Gender. (Como veremos mais adiante, há também um nível especial chamado [(All)] contendo um total geral.)

Os valores para a dimensão vir a partir da coluna de Gender na tabela de customer. A coluna "Gender" contém dois valores, 'F' e 'M', de modo que a dimensão do Gender contém os membros [Gender].[F] e [Gender].[M].

Para qualquer venda, a dimensão do Gender é o sexo do cliente que fez essa compra. Isto é expresso juntando-se a partir da tabela de fatos "sales_fact_1997.customer_id" para a tabela de dimensão "customer.customer_id".

3.3.1 Mapeamento dimensões e hierarquias em tabelas

Uma dimensão é unida a um cubo por meio de um par de colunas, um na tabela fato, o outro na tabela de dimensão. O elemento <Dimension> tem um atributo foreignKey, que é o nome de uma coluna na tabela de verdade; o <hierarquia> elemento tem um atributo primaryKey.

Se a hierarquia tem mais de uma tabela, você pode distinguir usando o atributo primaryKeyTable.

O atributo coluna define a chave do nível. Deve ser o nome de uma coluna na tabela de níveis. Se a chave é uma expressão, você pode passar a usar o elemento <KeyExpression> dentro do nível. O que segue é equivalente ao exemplo acima:

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

Outros atributos <Level>, <Measure> e <Property> podem ter elementos aninhados correspondente:

Other attributes of <Level>, <Measure> and <Property> have corresponding nested elements:

Parent element	**Atributo**	**Elemento Aninhado Equivalente**	**Descrição**
<Level>	column	<KeyExpression>	Chave e Nível
<Level>	nameColumn	<NameExpression>	Expressão que define o nome de membros deste nível. Se não for especificado, a chave de nível é utilizado.
<Level>	ordinalColumn	<OrdinalExpression>	Expressão que define a ordem dos membros. Se não for especificado, a chave de nível é utilizado.
<Level>	captionColumn	<CaptionExpression>	Expressão que faz a legenda de membros. Se não for especificado, o nome do nível é usado.
<Level>	parentColumn	<ParentExpression>	Expression pelo qual os membros filhos referênciam o seu membro pai em uma hierarquia pai-filho. Não especificado em uma hierarquia regular.
<Measure>	column	<MeasureExpression>	Expressão SQL para calcular o valor da métrica (o argumento para a função de agregação SQL).
<Property>	column	<PropertyExpression>	Expressão SQL para calcular o valor da propriedade.

Os atributo uniqueMembers é usado para otimizar a geração de SQL. Se você sabe que os valores de uma determinada coluna nível na tabela de dimensão são únicos entre todos os outros valores dessa coluna através de níveis superiores,

em seguida, definir uniqueMembers = "true", caso contrário, definida como "false". Por exemplo, uma dimensão de tempo como [year].[Month] terá uniqueMembers = "false" no nível Mês, como o mesmo mês aparece em diferentes anos. Por outro lado, se você tivesse uma hierarquia [Product Class].[Product Name], e você estava certo de que [Product Name] é único, então você pode definir uniqueMembers = "true". Se você não tem certeza, então sempre definido uniqueMembers = "false". No nível superior, este será sempre uniqueMembers = "true", já que não há nível pai.

O atributo highCardinality é usado para notificar o Mondrian que há um número indefinido e muito elevado de elementos relativos a esta dimensão. Os valores aceitáveis são verdadeiros ou falsos (último é o valor padrão). Ações realizadas ao longo de todo o conjunto de elementos de dimensão não pode ser realizada quando se utiliza highCardinality = "true".

3.3.2 O membro ‘todos’('all')

Por padrão, cada hierarquia contém um nível superior chamada '(All)', que contém um único membro do chamado "(All {hierarchyName}) '. Esse membro é pai de todos os outros membros da hierarquia, e, portanto, representa um total geral. Ele também é o membro padrão da hierarquia; isto é, o elemento que é utilizado para calcular valores de célula quando a hierarquia não é incluído em um eixo ou no cortador. Os atributos TheallMemberName e allLevelName substituem os nomes padrões de todos os níveis e de todos os membros.

Se o elemento <Hierarchy> tem hasAll = "false", o nível de "todos" é suprimida. O membro padrão dessa dimensão agora será o primeiro membro do primeiro nível; Por exemplo, em uma hierarquia de tempo, que será o primeiro ano na hierarquia. Alterar o membro padrão pode ser confuso, então você deve geralmente usar hasAll = "true".

O elemento <Hierarchy> também tem um atributo DefaultMember, para substituir o membro padrão da hierarquia:

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

3.3.3 Dimensões de tempo

Dimensões de tempo com base no ano/mês/semana/dia são codificados de forma diferente no esquema Mondrian devido ao tempo MDX terem funções relacionadas, tais como:

- ParallelPeriod([level[, index[, member]]])
- PeriodsToDate([level[, member]])
- WTD([member])

- MTD([member])
- QTD([member])
- YTD([member])
- LastPeriod(index[, member])

Dimensões de tempo têm type = "TimeDimension". O papel de um nível em uma dimensão de tempo é indicada pelo atributo de níveis levelType, cujos valores admissíveis são como se segue:

levelType valores	**Significado**
TimeYears	Nível em anos
TimeQuarters	Nível em quartos
TimeMonths	Nível em meses
TimeWeeks	Nível em semanas
TimeDays	Nível em dias

Aqui é um exemplo de uma dimensão de tempo:

```
<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>
```

3.3.4 Ordem e visualização dos níveis

Observe no exemplo hierarquia de tempo acima do ordinalColumn e NameColumn atributos no elemento <Level>. Estes afetam o modo como os níveis são exibidos em um resultado. O atributo ordinalColumn especifica uma coluna na tabela de hierarquia que fornece a ordem dos membros de um determinado nível, enquanto o NameColumn especifica uma coluna que será exibida.

Por exemplo, no nível Mês acima, a tabela datehierarchy tem meses (1 .. 12) e MONTH_NAME (Janeiro, Fevereiro, ...) colunas. O valor da coluna que será usada internamente dentro MDX é a coluna do mês, as especificações do membro

serão válidos no formato: [Time] [2005] [Q1] [1].... Os membros do nível de [mês] será exibido na ordem janeiro, fevereiro, etc.

Em uma hierarquia pai-filho, os membros são sempre classificados em ordem hierárquica. O atributo ordinalColumn controla a ordem que os irmãos aparecem dentro de seu pai.

Colunas ordinais podem ser de qualquer tipo de dados que podem ser usadas em uma cláusula ORDER BY. O Âmbito da ordenação é per-parent, assim, no exemplo acima, a coluna day_in_month tem ciclo para cada mês. Valores devolvidos pelo driver JDBC deve haver casos não nulos de java.lang.Comparable que produzem a ordenação desejado quando seu método Comparable.compareTo é chamado.

Níveis contêm um atributo tipo, que podem ter valores "String", "Integer", "Numeric", "Boolean", "Date", "Time", and "Timestamp". O valor padrão é "Numeric", porque as colunas de chave geralmente têm um tipo numérico. Se é um tipo diferente, Mondrian precisa saber disso para que ele possa gerar instruções SQL corretamente; por exemplo, valores de cadeia será gerada entre aspas simples:

```
WHERE productSku = '123-455-AA'
```

3.3.5 Multiplas hierarquias

Uma dimensão pode conter mais do que uma hierarquia de:

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

Observe que a primeira hierarquia não tem um nome. Por padrão, a hierarquia tem o mesmo nome que a sua dimensão, por isso a primeira hierarquia é chamado de "Time".

Essas hierarquias não têm muito em comum - eles nem sequer têm a mesma tabela! - Exceto que eles são unidas pela mesma coluna na tabela fatos ", TIME_ID". A principal razão para colocar duas hierarquias da mesma dimensão é porque faz mais sentido para o usuário final: os usuários finais sabem que não faz sentido ter a hierarquia "Time" em um eixo e a hierarquia "Tempo Semanal" na outro eixo. Se duas hierarquias são a mesma dimensão, a linguagem MDX reforça o senso comum, e não permite que você use os dois na mesma consulta.

3.3.6 Dimensões degeneradas

Uma dimensão degenerada, uma dimensão que é tão simples que não vale a pena criar a sua própria tabela de dimensão. Por exemplo, considere a seguir a tabela de fatos:

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

e suponha que criamos uma tabela de dimensão para os valores na coluna payment_method:

payment_method
Credit
Cash
ATM

Esta tabela de dimensão é bastante inútil. Ele só tem 3 valores, e não acrescenta informação adicional, e incorre o custo extra para aderir.

Em vez disso, você pode criar uma dimensão degenerada. Para fazer isso, declare uma dimensão sem uma tabela, e o Mondrian irá assumir que as colunas vêm da tabela fatos.

```
<Cube name="Checkout">
	<!-- The fact table is always necessary. →
	<Table name="checkout">
	<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>
	<!-- other dimensions and measures -->
</Cube>
```

Note-se que não há adesão, o atributo foreignKey da dimensão não é necessária, e o elemento de hierarquia <Tabel> não tem nenhuma elemento filho ou um atributo chave primária.

3.3.7 Tabelas em linha

A construção <InlineTable> permite definir um conjunto de dados no arquivo de esquema. Você deve declarar os nomes das colunas, os tipos de coluna ("string" ou "numeric"), e um conjunto de linhas. Quanto <table> e <View>, você deve fornecer um alias exclusivo com a qual para se referir ao conjunto de dados.

Aqui está um exemplo:

```
<Dimension name="Severity">
	<Hierarchy hasAll="true" primaryKey="severity_id">
		<InlineTable alias="severity">
			<ColumnDefs>
				<ColumnDef name="id" type="Numeric"/>
				<ColumnDef name="desc" type="String"/>
			</ColumnDefs>
			<Rows>
				<Row>
					<Value column="id">1</Value>
					<Value column="desc">High</Value>
				</Row>
				<Row>
					<Value column="id">2</Value>
					<Value column="desc">Medium</Value>
				</Row>
				<Row>
					<Value column="id">3</Value>
					<Value column="desc">Low</Value>
				</Row>
			</Rows>
		</InlineTable>
		<Level	name="Severity"	column="id"	nameColumn="desc"
		uniqueMembers="true"/>
	</Hierarchy>
</Dimension>
```

Isto tem o mesmo efeito como se você tivesse uma tabela chamada "severity" no seu banco de dados:

id	desc
1	High
2	Medium
3	Low

e a declaração

```
<Dimension name="Severity">
	<Hierarchy hasAll="true" primaryKey="severity_id">
		<Table name="severity"/>
		<Level name="Severity" column="id" nameColumn="desc" uniqueMembers="true"/>
	</Hierarchy>
</Dimension>
```

Para especificar um valor NULL para uma coluna, omitir o <valor> para essa coluna, e o valor da coluna será o padrão NULL.

3.3.8 As propriedades de membros e formatadores

Como veremos mais tarde, uma definição de nível também pode definir propriedades de membro e um formatador membro.

3.3.9 Cardinalidade Nível Aproximado

O elemento <Level> permite especificar o atributo opcional "approxRowCount". Especificando approxRowCount pode melhorar o desempenho, reduzindo a necessidade de determinar o nível, hierarquia e dimensão cardinalidade. Isso pode ter um impacto significativo ao se conectar a Mondrian via XMLA.

3.3.10 Métricas Atributos Padrões

Os elementos <Cube> e <> VirtualCube permitem especificar o atributo opcional "defaultMeasure".

Especificando defaultMeasure no elemento <Cube> permite aos usuários especificar explicitamente qualquer medida base como uma medida padrão.

Especificando defaultMeasure no elemento <VirtualCube> permite aos usuários especificar explicitamente qualquer medida VirtualCube como uma medida padrão.

Observe que, se uma medida padrão não é especificado que leva a primeira medida definida no cubo como a medida padrão. No caso do cubo virtual, ele iria pegar a primeira medida base de dentro do primeiro cubo definido como padrão.

Especificando o defaultMeasure explicitamente é útil nos casos em que você gostaria de ter um membro calculado para ser levantado como a medida padrão. Para facilitar isto, o membro calculado pode ser definida em um dos cubos de base e especificada como o defaultMeasurein o cubo virtual.

```
<Cube name="Sales" defaultMeasure="Unit Sales">
	…
	<CalculatedMember name="Profit" dimension="Measures">
		<Formula>[Measures].[Store Sales] - [Measures].[Store Cost]</Formula>
		…
	</CalculatedMember>
</Cube>
<VirtualCube name="Warehouse and Sales" defaultMeasure="Profit">
	…
	<VirtualCubeMeasure cubeName="Sales" name="[Measures].[Profit]"/>
</VirtualCube>
```

3.3.11 Dependências funcionais e otimizações

Em algumas circunstâncias, pode ser possível optimizar o desempenho, tirando partido das dependências funcionais conhecidas nos dados a serem processados. Tais dependências são tipicamente resultado de regras de negócio associadas aos sistemas de produção dos dados, e muitas vezes não pode ser inferida apenas olhando para os dados em si.

Dependências funcionais são declarados no Mondrian usando o atributo dependsOnLevelValue do elemento <property> e theuniqueKeyLevelName atributo do <Hierarchy> elemento.

O atributo dependsOnLevelValue de uma propriedade membro é usado para indicar que o valor da propriedade membro é funcionalmente dependente do valor do <Level> em que a propriedade do membro é definido. Em outras palavras, para um dado valor do nível, o valor da propriedade não varia.

O atributo uniqueKeyLevelName de um <Hierarchy> é utilizado para indicar que o nível dado (se houver) tomado em conjunto com todos os níveis mais elevados na hierarquia atua como uma chave alternativa original, assegurando que, para qualquer combinação única dos valores de nível, não é exatamente uma combinação de valores para todos os níveis abaixo dele.

Para ilustrar, considere uma hierarquia modelar carros construídos e licenciadas nos Estados Unidos:

```
<Dimension name="Automotive" foreignKey="auto_dim_id">
       <Hierarchy hasAll="true" primaryKey="auto_dim_id" uniqueKeyLevelName="Vehicle
          Identification Number">
              <Table name="automotive_dim"/>
              <Level name="Make" column="make_id" type="Numeric"/>
              <Level name="Model" column="model_id" type="Numeric"/>
              <Level name="ManufacturingPlant" column="plant_id" type="Numeric"/>
                     <Property name="State" column="plant_state_id" type="Numeric"
                        dependsOnLevelValue="true"/>
                     <Property name="City" column="plant_city_id" type="Numeric"
                        dependsOnLevelValue="true"/>
              <Level name="Vehicle Identification Number" column="vehicle_id"
                 type="Numeric"/>
                     <Property name="Color" column="color_id" type="Numeric"
                        dependsOnLevelValue="true"/>
                     <Property name="Trim" column="trim_id" type="Numeric"
                        dependsOnLevelValue="true"/>
              <Level name="LicensePlateNum" column="license_id" type="String"/>
                     <Property name="State" column="license_state_id" type="Numeric"
                        dependsOnLevelValue="true"/>
       </Hierarchy>
</Dimension>
```

No exemplo acima, sabemos que uma determinada fábrica só existe em uma única cidade e estado, que um determinado carro só tem um esquema de cores e um nível de acabamento, e que o número de licença está associada a um único estado. Portanto, podemos afirmar que todas estas propriedades membros são funcionalmente dependente dos valores de nível associados.

Além disso, sabemos que o número de identificação do veículo identifica unicamente cada carro, e que cada carro tem apenas uma licença. Assim, sabemos que a combinação de marca, modelo, Fábrica, e Número de Identificação do Veículo identifica unicamente cada veículo; o número de licença é redundante.

Estes atributos permitem a optimização da cláusula GROUP BY nas instruções geradas pelo SQL Mondrian. Na ausência de qualquer informação de dependência funcional, uma consulta típica na dimensão Automotive seria algo parecido com:

```
SELECT
`automotive_dim`.`make_id` AS c0,
`automotive_dim`.`model_id` AS c1,
`automotive_dim`.`plant_id` AS c2,
`automotive_dim`.`plant_state_id` AS c3,
`automotive_dim`.`plant_city_id` AS c4,
`automotive_dim`.`vehicle_id` AS c5,
`automotive_dim`.`color_id` AS c6,
`automotive_dim`.`trim_id` AS c7,
`automotive_dim`.`license_id` AS c8,
`automotive_dim`.`license_state_id` AS c9
FROM
`automotive_dim` AS `automotive_dim`,
GROUP BY
`automotive_dim`.`make_id`,
`automotive_dim`.`model_id`,
`automotive_dim`.`plant_id`,
`automotive_dim`.`plant_state_id`,
`automotive_dim`.`plant_city_id`,
`automotive_dim`.`vehicle_id`,
`automotive_dim`.`color_id`,
`automotive_dim`.`trim_id`,
`automotive_dim`.`license_id`,
`automotive_dim`.`license_state_id`
ORDER BY
`...
```

Dada a dependência funcional do exemplo no esquema acima, sabemos que a consulta é selecionar uma profundidade que inclui o nível "único key", e que todas as propriedades na consulta também são funcionalmente dependentes de seus níveis. Nesses casos, a cláusula GROUP BY é redundante e pode ser completamente eliminada, aumentando o desempenho da consulta SQL significativamente em algumas bases de dados:

```
SELECT
`automotive_dim`.`make_id` AS c0,
`automotive_dim`.`model_id` AS c1,
`automotive_dim`.`plant_id` AS c2,
`automotive_dim`.`plant_state_id` AS c3,
`automotive_dim`.`plant_city_id` AS c4,
`automotive_dim`.`vehicle_id` AS c5,
`automotive_dim`.`color_id` AS c6,
`automotive_dim`.`trim_id` AS c7,
`automotive_dim`.`license_id` AS c8,
`automotive_dim`.`license_state_id` AS c9
FROM
`automotive_dim` AS `automotive_dim`,
ORDER BY
`...
```

A consulta não foi profunda o suficiente para incluir o nível "unique key", ou teve alguma das propriedades de membro não foram funcionalmente dependente de seu nível, essa otimização não seria possível.

Em alguns casos, uma optimização diferente pode ser feita quando não existe um nível "chave única", mas algumas ou todas as propriedades do membro são funcionalmente dependente do seu nível. Alguns bancos de dados (nomeadamente MySQL) permitir colunas a serem listados na cláusula SELECT que também não aparecer na cláusula GROUP BY. Em tais bancos de dados, Mondrian pode simplesmente deixar as propriedades de membro funcionalmente dependentes

fora do GROUP BY, o que pode reduzir o tempo de processamento SQL consulta substancialmente:

```
SELECT
`automotive_dim`.`make_id` AS c0,
`automotive_dim`.`model_id` AS c1,
`automotive_dim`.`plant_id` AS c2,
`automotive_dim`.`plant_state_id` AS c3,
`automotive_dim`.`plant_city_id` AS c4,
`automotive_dim`.`vehicle_id` AS c5,
`automotive_dim`.`color_id` AS c6,
`automotive_dim`.`trim_id` AS c7,
`automotive_dim`.`license_id` AS c8,
`automotive_dim`.`license_state_id` AS c9
FROM
`automotive_dim` AS `automotive_dim`,
GROUP BY
`automotive_dim`.`make_id`,
`automotive_dim`.`model_id`,
`automotive_dim`.`plant_id`,
`automotive_dim`.`vehicle_id`,
`automotive_dim`.`license_id`,
ORDER BY
`...
```

Por favor, note que mudanças significativas estão previstos para a sintaxe do esquema no Mondrian 4.0, incluindo uma nova abordagem para declarar dependências funcionais. Enquanto a expectativa é que o processador do esquema 4.0 irá manter a compatibilidade com versões anteriores com esquemas desenvolvidos para Mondrian 3.1, estes são atributos transitórias introduzidas para permitir um apoio nesse ínterim, e o 4.0 não será compatível com eles. Portanto, qualquer esquema utilizando esses atributos terá de ser migrados para a nova sintaxe como parte da atualização a Mondrian 4.0.

3.3.12 Table Hints

Mondrian suporta um conjunto limitado de sugestões específicas do banco de dados para o elemento <table>, que serão então repassados para consultas SQL que envolvem a mesa. Estas pistas são como se segue:

Database	Hint Type	Permitted Values	Description
MySQL	force_index	O nome de um índice sobre esta tabela	Força o índice chamado para ser usado ao selecionar valores de nível a partir desta tabela.

Como exemplo

```
<Table name="automotive_dim">
       <Hint type="force_index">my_index</Hint>
</Table>
```

Tal como acontece com as otimizações de dependência funcional, suporte para dicas de tabela está em uma fase de transição, e são susceptíveis de mudar em Mondrian 4.0. Qualquer esquema de usá-los pode precisar de ser migrados para a nova sintaxe do esquema como parte da atualização a Mondrian 4.0.

4. Esquema Estrela e Floco de Neve

Vimos anteriormente como construir um cubo com base em uma tabela de fatos e dimensões na tabela de fatos ("Payment method") e de uma tabela unida à tabela de fatos ("Gender"). Este é o tipo mais comum de mapeamento, e é conhecido como um esquema em estrela.

Mas uma dimensão pode basear-se em mais do que uma tabela, desde que haja um caminho bem definido para se juntar a estas tabelas para a tabela de verdade. Este tipo de dimensão é conhecido como um floco de neve, e é definida usando o <Join> operador. Por exemplo:

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

Isto define uma dimensão "Product" com três tabelas. A tabela de fatos se junta ao "product" (via a chave estrangeira "product_id"), que se junta a "product_class" (através da chave estrangeira "product_class_id"), que se junta a "product_type" (através do chave estrangeira "product_type_id"). Nós exigimos um elemento <join> aninhado dentro de um elemento <join>, porque <Join> têm duas duas operações; As operações podem ser tabelas, join's, ou mesmo queries.

A disposição das tabelas parece complexa; a simples regra de ouro é para ordenar as tabelas pelo número de linhas que contêm. A tabela "product" tem a maioria de linhas, por isso junta-se à tabela de fatos e aparece em primeiro lugar; "Product_class" tem menos linhas, e "product_type", na ponta do floco, tem menos de todos.

Note-se que o elemento <Join> externo tem um atributo rightAlias. Isto é necessário porque o componente direito da junção (o elemento <join> interior) consiste em mais do que uma tabela. Não é necessário o atributo leftAlias, neste caso, porque a coluna leftKey inequivocamente vem da tabela do "product".

4.1 Dimensões Compartilhadas

Ao gerar o SQL com um Join, o Mondrian precisa saber qual coluna será usada no Join. Se você estão se juntando a um Join, então você precisa dizer a ele qual das tabelas na junção pertence à coluna (geralmente será a primeira tabela na junção).

Porque dimensões compartilhadas não pertencem a um cubo, você tem que dar-lhes uma tabela explícita (ou outra fonte de dados). Quando você for usá-los em um cubo particular, você deve especificar a chave estrangeira. Este exemplo mostra a dimensão "Store Type" sendo unidos ao cubo Sales usando thesales_fact_1997.store_id como chave estrangeira, e ao cubo Warehouse usando a chave estrangeira warehouse.warehouse_store_id:

```
<Dimension name="Store Type">
       <Hierarchy hasAll="true" primaryKey="store_id">
              <Table name="store"/>
              <Level name="Store Type" column="store_type" uniqueMembers="true"/>
       </Hierarchy>
</Dimension>

<Cube name="Sales">
       <Table name="sales_fact_1997"/>
       …
       <DimensionUsage name="Store Type" source="Store Type" foreignKey="store_id"/>
</Cube>

<Cube name="Warehouse">
       <Table name="warehouse"/>
       …
       <DimensionUsage        name="Store        Type"        source="Store        Type"
          foreignKey="warehouse_store_id"/>
</Cube>
```

4.2 Otimização do Join

O mapeamento da tabela no esquema diz ao Mondrian como obter os dados, mas Mondrian é inteligente o suficiente para não ler o esquema literalmente. Aplica-se uma série de otimizações ao gerar consultas:

- TODO: descreve grande apoio dimensão
- Se a dimensão (ou, mais precisamente, o nível da dimensão a ser acedido) está na tabela de verdade, Mondrian não executar o Join.
- Se duas dimensões acessar a mesma tabela através do mesmo caminho do do Join, Mondrian executa apenas uma vez. Por exemplo, [Gender] e [Age] pode ser ambas colunas na tabela de clientes, juntou-se via sales_1997.cust_id = customers.cust_id.

5. Construtores lógicos avançados

5.1 Cubos Virtuais

Um cubo virtual combina dois ou mais cubos regulares. É definido pelo elemento <VirtualCube>:

```
<VirtualCube name="Warehouse and Sales">
    <CubeUsages>
          <CubeUsage cubeName="Sales" ignoreUnrelatedDimensions="true"/>
          <CubeUsage cubeName="Warehouse"/>
    </CubeUsages>
    <VirtualCubeDimension cubeName="Sales" name="Customers"/>
    <VirtualCubeDimension cubeName="Sales" name="Education Level"/>
    <VirtualCubeDimension cubeName="Sales" name="Gender"/>
    <VirtualCubeDimension cubeName="Sales" name="Marital Status"/>
    <VirtualCubeDimension name="Product"/>
    <VirtualCubeDimension cubeName="Sales" name="Promotion Media"/>
    <VirtualCubeDimension cubeName="Sales" name="Promotions"/>
    <VirtualCubeDimension name="Store"/>
    <VirtualCubeDimension name="Time"/>
    <VirtualCubeDimension cubeName="Sales" name="Yearly Income"/>
    <VirtualCubeDimension cubeName="Warehouse" name="Warehouse"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Sales Count]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Store Cost]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Store Sales]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Unit Sales]"/>
    <VirtualCubeMeasure cubeName="Sales" name="[Measures].[Profit Growth]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Store Invoice]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Supply Time]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Units Ordered]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Units Shipped]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Warehouse Cost]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Warehouse Profit]"/>
    <VirtualCubeMeasure cubeName="Warehouse" name="[Measures].[Warehouse Sales]"/>
    <VirtualCubeMeasure  cubeName="Warehouse"  name="[Measures].[Average  Warehouse
      Sale]"/>
    <CalculatedMember name="Profit Per Unit Shipped" dimension="Measures">
          <Formula>[Measures].[Profit] / [Measures].[Units Shipped]</Formula>
    </CalculatedMember>
</VirtualCube>
```

O elemento <CubeUsages> é opcional. Ele especifica os cubos que são importados para o cubo virtual. Detém elementos CubeUsage.

O elemento <CubeUsage> é opcional. Ele especifica o cubo da base que é importado para o cubo virtual. Atualmente, é possível definir um VirtualCubeMeasure e as importações semelhantes a partir de um cubo de base sem definir CubeUsage para o cubo. O atributo CubeName especifica o cubo da base que está sendo importado. O atributo ignoreUnrelatedDimensions especifica quais as metricas do cubo base será importado dos membros do nível superior. Este comportamento é suportado para a agregação. Este atributo é por falso padrão. ignoreUnrelatedDimensions é um recurso experimental semelhante ao recurso de nome semelhante na documentação SSAS 2005. A documentação MSDN "Quando IgnoreUnrelatedDimensions é true, dimensões independentes são obrigados a seu nível superior;. quando o valor é falso, as dimensões não são forçados a sua nível superior, esta propriedade é semelhante ao Multidimensional Expressions (MDX) função ValidMeasure ". A iimplementação atual do Mondrian para

ignoreUnrelatedDimensions depende do uso de ValidMeasure. Por exemplo. Se queremos aplicar esse comportamento para medida "Unit Sales" no "Warehouse and Sales" cubo virtual, em seguida, precisamos definir uma entrada CubeUsage para "Sales" cubo como mostrado no exemplo acima e também envolvê desta medida com ValidMeasure.

O elemento <VirtualCubeDimension> importa uma dimensão de um dos cubos constituintes. Se você não especificar o atributo CubeName, isso significa que você está importando uma dimensão compartilhada. (Se uma dimensão compartilhada é usada mais do que uma vez no cubo, não há nenhuma maneira, no momento, para desfazer a ambiguidade que o uso da dimensão partilhada a ser importada).

O elemento <VirtualCubeMeasure> importa uma medida de um dos cubos constituintes. É importado com o mesmo nome. Se você quiser criar uma fórmula, ou apenas para mudar o nome de uma medida antes de importá-lo, use o elemento <CalculatedMember>.

Cubos virtuais podem ocorrer com surpreendente frequência em aplicações do mundo real. Eles ocorrem quando você tem tabelas de fatos de diferentes granularidades (dizer uma medida no nível do dia, outro no nível mês), ou tabelas de fatos de diferentes dimensionalidades (digamos um em Produto, Tempo e Cliente, outro sobre Produto, Tempo e Armazém) , e pretende apresentar os resultados para um usuário final que não sabe ou não tem o cuidado como os dados são estruturados.

As dimensões comuns - dimensões compartilhadas que são usadas por ambos os cubos constituintes - são automaticamente sincronizados. Neste exemplo, [Time] e [Product] são dimensões comuns. Então, se o contexto é ([Time].[1997].[Q2], [Product].[CervBeereja].[Miller Lite]), metricas a partir de qualquer cubo irá relacionar a este contexto.

Dimensões que só pertencem a um cubo são chamados de dimensões não conformes. O [Gender] dimensão é um exemplo disso: ele existe no cubo de vendas, mas não Warehouse. Se o contexto for ([Gender].[F], [Time].[1997].[Q1]), faz sentido perguntar o valor da métrica [Unit Sales] (que vem do [Sales] cubo) mas não as métricas [Units Ordered] (de [Warehouse]). No contexto da [Gender].[F], [Units Ordered] tem um valor NULL.

5.2 Hierarquias pai-filho

Uma hierarquia convencional tem um conjunto rígido de níveis e membros que aderem a esses níveis. Por exemplo, na hierarquia de produto, qualquer membro do nível Product Name tem um pai no nível Marca, que tem um dos progenitores no nível Sub-produto, e assim por diante. Esta estrutura é por vezes demasiado rígida para modelar dados do mundo real.

A hierarquia pai-filho tem apenas um nível (não contando o 'all' nível especial), mas qualquer membro pode ter pais no mesmo nível. Um exemplo clássico é a estrutura de relatórios na hierarquia de funcionários:

```
<Dimension name="Employees" foreignKey="employee_id">
    <Hierarchy          hasAll="true"          allMemberName="All          Employees"
      primaryKey="employee_id">
          <Table name="employee"/>
          <Level    name="Employee    Id"    uniqueMembers="true"    type="Numeric"
            column="employee_id"                              nameColumn="full_name"
            parentColumn="supervisor_id" nullParentValue="0">
                <Property name="Marital Status" column="marital_status"/>
                <Property name="Position Title" column="position_title"/>
                <Property name="Gender" column="gender"/>
                <Property name="Salary" column="salary"/>
                <Property name="Education Level" column="education_level"/>
                <Property name="Management Role" column="management_role"/>
          </Level>
    </Hierarchy>
</Dimension>
```

Os atributos importantes aqui são parentColumn e nullParentValue:

- O atributo parentColumn é o nome da coluna que liga um membro para o membro pai; neste caso, é a coluna de chave estrangeira que aponta para supervisor de um empregado. O <ParentExpression> elemento filho do <nível> é equivalente ao atributo theparentColumn, mas permite que você defina uma expressão SQL arbitrária, assim como o <Expression> elemento. O (elemento ou <ParentExpression>) parentColumnattribute é a única indicação a Mondrian que a hierarquia tem uma estrutura de pai-filho.

- O atributo nullParentValue é o valor que indica que um membro não tem pai. O padrão é nullParentValue = "null", mas uma vez que muitos de banco de dados não indexe valores nulos, os designers de esquema, por vezes, usar valores como a cadeia vazia, 0 e -1 em vez disso.

5.2.1 Ajustando hierarquias pai-filho

Há um problema sério com a hierarquia pai-filho acima definido, e que é a quantidade de trabalho Mondrian tem que fazer, a fim de calcular células-totais. Vamos supor que a tabela de funcionários contém os seguintes dados:

employee

supervisor_id	employee_id	full_name
null	1	Frank
1	2	Bill
2	3	Eric
1	4	Jane
3	5	Mark
2	6	Carla

Se quisermos calcular o orçamento total salário para Bill, precisamos adicionar nos salários de Eric e Carla (que se reportam a Bill) e Mark (que se reporta ao Eric). Normalmente Mondrian gera um GROUP BY SQL declaração para calcular estes totais, mas não há (geralmente disponíveis) construção de SQL que pode atravessar hierarquias. Então, por padrão, Mondrian gera uma instrução SQL por supervisor, para recuperar e total de todos os relatórios diretos do supervisor.

Esta abordagem tem algumas desvantagens. Primeiro, o desempenho não é muito boa, se uma hierarquia contém mais do que uma centena de membros. Segundo, porque Mondrian implementa o agregador distinta de contagem através da geração de SQL, você não pode definir uma medida distinta de contagem em qualquer cubo que contém uma hierarquia pai-filho.

Como podemos resolver esses problemas? A resposta é para melhorar os dados para que Mondrian é capaz de recuperar as informações necessárias usando SQL padrão. Mondrian suporta um mecanismo chamado de tabela de fecho para esse fim.

5.2.2 Tabela de fechamento

A tabela de encerramento é uma tabela SQL que contém um registro para cada relação empregado / supervisor, independentemente da profundidade. (Em termos matemáticos, isso é chamado de “fechamento transitivo reflexiva” da relação empregado / supervisor. A coluna de distância não é estritamente necessário, mas isso torna mais fácil para preencher a tabela.)

employee_closure

supervisor_id	employee_id	distance
1	1	0
1	2	1
1	3	2
1	4	1
1	5	3
1	6	2
2	2	0
2	3	1
2	5	2
2	6	1
3	3	0
3	5	1
4	4	0
5	5	0
6	6	0

No catálogo XML, o <Closure> elemento mapeia o nível em um <Table>:

```
<Dimension name="Employees" foreignKey="employee_id">
      <Hierarchy        hasAll="true"        allMemberName="All        Employees"
primaryKey="employee_id">
            <Table name="employee"/>
            <Level name="Employee Id" uniqueMembers="true" type="Numeric"
column="employee_id"      nameColumn="full_name"      parentColumn="supervisor_id"
nullParentValue="0">
                  <Closure                          parentColumn="supervisor_id"
            childColumn="employee_id">
                        <Table name="employee_closure"/>
                  </Closure>
                  <Property name="Marital Status" column="marital_status"/>
                  <Property name="Position Title" column="position_title"/>
                  <Property name="Gender" column="gender"/>
                  <Property name="Salary" column="salary"/>
                  <Property name="Education Level" column="education_level"/>
                  <Property name="Management Role" column="management_role"/>
            </Level>
      </Hierarchy>
</Dimension>
```

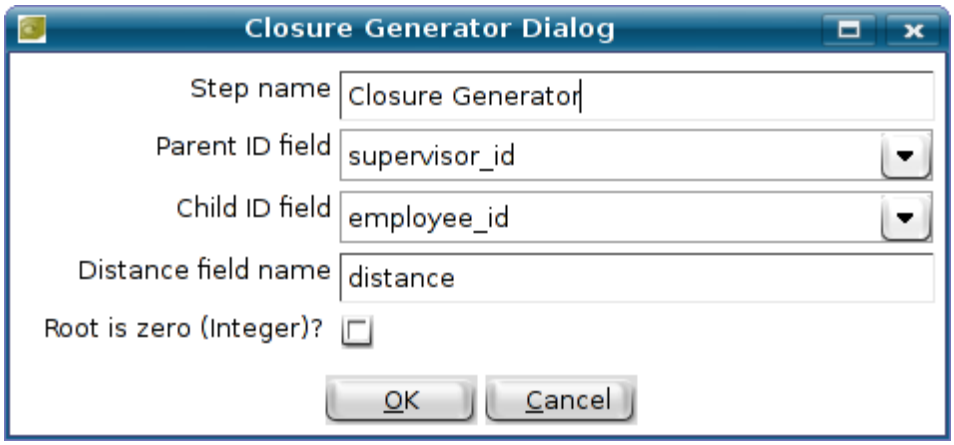
Esta tabela permite totais a serem avaliados no SQL puro. Mesmo que este introduz uma tabela extra para a consulta, otimizadores de bancos de dados são muito bons em lidar com joins. Eu recomendo a você declarar tanto supervisor_id e não employee_id NULL, e indexá-los da seguinte forma:

```
CREATE UNIQUE INDEX employee_closure_pk ON employee_closure (
   supervisor_id,
   employee_id);
CREATE INDEX employee_closure_emp ON employee_closure (
   employee_id);
```

5.2.3 Preencher tabelas de fechamento

A tabela precisa ser repreenchidas sempre que a hierarquia muda, e é responsabilidade do aplicativo para fazê-lo - O Mondrian não faz isso!

Se você estiver usando Pentaho Data Integration (Kettle), há uma etapa especial para preencher tabelas de fechamento, como parte do processo de ETL. Mais detalhes no wiki Pentaho Data Integration.



Closure Generator step in Pentaho Data Integration

Se você não estiver usando Pentaho Data Integration, você pode preencher a tabela usando SQL. Aqui está um exemplo de um procedimento armazenado MySQL que preenche uma tabela de encerramento.

```
DELIMITER //

CREATE PROCEDURE populate_employee_closure()
BEGIN
  DECLARE distance int;
  TRUNCATE TABLE employee_closure;
  SET distance = 0;
  -- seed closure with self-pairs (distance 0)
  INSERT INTO employee_closure (supervisor_id, employee_id, distance)
       SELECT employee_id, employee_id, distance
       FROM employee;

  -- for each pair (root, leaf) in the closure,
  -- add (root, leaf->child) from the base table
  REPEAT
```

```
      SET distance = distance + 1;
      INSERT INTO employee_closure (supervisor_id, employee_id, distance)
      SELECT employee_closure.supervisor_id, employee.employee_id, distance
      FROM employee_closure, employee
      WHERE employee_closure.employee_id = employee.supervisor_id
      AND employee_closure.distance = distance - 1;
  UNTIL (ROW_COUNT() == 0))
  END REPEAT;
END //

DELIMITER ;
```

5.3 As propriedades membros

As propriedades membros são definidos pelo elemento <Property> dentro de um <Level>, como este:

```
<Level name="MyLevel" column="LevelColumn" uniqueMembers="true">
      <Property                  name="MyProp"                  column="PropColumn"
      formatter="com.example.MyPropertyFormatter"/>
<Level/>
```

Atributo de formatador define um formato de propriedade, que é explicado mais tarde.

Uma vez que as propriedades foram definidas no esquema, você pode usá-los em instruções MDX através dos member.Properties ("PROPERTYNAME") função, por exemplo:

```
SELECT {[Store Sales]} ON COLUMNS,
  TopCount(Filter([Store].[Store Name].Members,
                  [Store].CurrentMember.Properties("Store         Type")         =
"Supermarket"),
           10,
             [Store Sales]) ON ROWS
FROM [Sales]
```

Mondrian deduz o tipo de expressão de propriedade, se puder. Se o nome da propriedade é uma string constante, o tipo baseia-se no typeattribute ("String", "Numeric" ou "Boolean") da definição de propriedade. Se o nome da propriedade é uma expressão (por exampleCurrentMember.Properties ("Store" + "Type")), Mondrian retornará um valor sem tipo.

5.4 Membros Calculados

Suponha que você queira criar uma métrica cujo valor não vem de uma coluna da tabela de verdade, mas a partir de uma fórmula de MDX. Uma maneira de fazer isso é usar uma cláusula WITH MEMBER, como este:

```
WITH  MEMBER  [Measures].[Profit]  AS  '[Measures].[Store  Sales]-[Measures].[Store
Cost]',
   FORMAT_STRING = '$#,###'
SELECT {[Measures].[Store Sales], [Measures].[Profit]} ON COLUMNS,
  {[Product].Children} ON ROWS
FROM [Sales]
WHERE [Time].[1997]
```

Mas em vez de incluir esta cláusula em todas as consultas MDX do seu aplicativo, você pode definir o membro em seu esquema, como parte de sua definição de cubo:

```
<CalculatedMember name="Profit" dimension="Measures">
      <Formula>[Measures].[Store Sales] - [Measures].[Store Cost]</Formula>
      <CalculatedMemberProperty name="FORMAT_STRING" value="$#,##0.00"/>
</CalculatedMember>
```

Você também pode declarar a fórmula como um atributo XML, se você preferir. O efeito é o mesmo.

```
<CalculatedMember name="Profit" dimension="Measures" formula="[Measures].[Store
Sales]-[Measures].[Store Cost]">
    <CalculatedMemberProperty name="FORMAT_STRING" value="$#,##0.00"/>
</CalculatedMember>
```

Observe que o elemento <CalculatedMemberProperty> (não <property>) corresponde à FORMAT_STRING = '$ #, ###' fragmento da instrução MDX. Você pode definir outras propriedades aqui também, mas FORMAT_STRING é de longe o mais útil na prática.

O valor da propriedade FORMAT_STRING também pode ser avaliada utilizando uma expressão. Ao formatar uma célula em particular, pela primeira vez a expressão é avaliada para produzir uma cadeia de formato, em seguida, a seqüência de formato é aplicada ao valor da célula. Aqui é a mesma propriedade com uma cadeia de formatação condicional:

```
<CalculatedMemberProperty name="FORMAT_STRING" expression="Iif(Value < 0,
'|($#,##0.00)|style=red', '|$#,##0.00|style=green')"/>
```

Para mais detalhes sobre cadeias de formato, consulte a especificação MDX.

Uma propriedade membro calculada adicional que vale a pena mencionar é DATATYPE. Tal como acontece com as métricas, definindo o tipo de dados especificado como o membro calculado é retornado via XML para análise. A propriedade DATATYPE de um membro calculado pode ter valores "String", "Integer", ou "Numeric":

```
<CalculatedMemberProperty name="DATATYPE" value="Numeric"/>
```

Você pode especificar SOLVE_ORDER para a propriedade membro calculado. Solve determina a prioridade de cálculo no caso de expressões concorrentes

```
<CalculatedMemberProperty name="SOLVE_ORDER" value="2000"/>
```

Você pode fazer um membro calculado ou uma métrica invisível. Se você especificar visible = "false" (o padrão é "true") no elemento <Measure> ou <CalculatedMember>, interfaces de usuário, tais como JPivot vai notar esta propriedade e ocultar o membro. Isso é útil se você quiser executar cálculos em uma série de passos, e esconde etapas intermediárias de usuários finais. Por

exemplo, aqui apenas "Margem per Sqft" é visível, e seus fatores de "custo Store", "Margem" e "Store pé" estão escondidos:

```
<Measure name="Store Cost" column="store_cost" aggregator="sum"
formatString="#,###.00" visible="false"/>
<CalculatedMember name="Margin" dimension="Measures" visible="false">
      <Formula>([Measures].[Store Sales] - [Measures].[Store Cost]) /
      [Measures].[Store Cost]</Formula>
</CalculatedMember>
<CalculatedMember name="Store Sqft" dimension="Measures" visible="false">
      <Formula>IIF([Store].CurrentMember.Level.Name = "Store Name",
      [Store].Properties("Store Sqft"), NULL)</Formula>
</CalculatedMember>
<CalculatedMember name="Margin per Sqft" dimension="Measures" visible="true">
      <Formula>[Measures].[Margin] / [Measures].[Store Cost]</Formula>
<CalculatedMemberProperty name="FORMAT_STRING" value="$#,##0.00"/>
</CalculatedMember>
```

5.5 Os conjuntos nomeados

A cláusula WITH SET de uma instrução MDX permite declarar uma expressão de conjunto que pode ser usado em toda essa consulta. Por exemplo,

```
WITH SET [Top Sellers] AS
      'TopCount([Warehouse].[Warehouse Name].MEMBERS, 5, [Measures].[Warehouse
Sales])'
SELECT
      {[Measures].[Warehouse Sales]} ON COLUMNS,
      {[Top Sellers]} ON ROWS
FROM [Warehouse]
WHERE [Time].[Year].[1997]
```

A cláusula WITH SET é muito semelhante à cláusula WITH MEMBER, e como seria de esperar, tem uma construção em esquema análogo ao <CalculatedMember>. O elemento <NamedSet> permite definir um conjunto nomeado no seu esquema como parte de uma definição de cubo. É implicitamente disponível para qualquer consulta em relação a esse cubo:

```
<Cube name="Warehouse">
      …
            <NamedSet name="Top Sellers">
                  <Formula>TopCount([Warehouse].[Warehouse Name].MEMBERS, 5,
                        [Measures].[Warehouse Sales])</Formula>
            </NamedSet>
</Cube>
```

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

Warehouse	Warehouse Sales
Treehouse Distribution	31,116.37
Jorge Garcia, Inc.	30,743.77
Artesia Warehousing, Inc.	29,207.96
Jorgensen Service Storage	22,869.79
Destination, Inc.	22,187.42

Um conjunto nomeado definido contra um cubo não é herdado por um cubo virtual definido contra esse cubo. (Mas você pode definir um conjunto nomeado contra um cubo virtual.)

Você também pode definir um conjunto nomeado como global para um esquema:

```
<Schema>
	<Cube name="Sales" ... />
	<Cube name="Warehouse" ... />
	<VirtualCube name="Warehouse and Sales" .../>
	<NamedSet name="CA Cities" formula="{[Store].[USA].[CA].Children}"/>
	<NamedSet name="Top CA Cities">
		<Formula>TopCount([CA Cities], 2, [Measures].[Unit Sales])</Formula>
	</NamedSet>
</Schema>
```

Um conjunto nomeado definido contra um esquema está disponível em todos os cubos e cubos virtuais em que esquema. No entanto, só é válido se o cubo contém dimensões com os nomes necessários para fazer a fórmula válida. Por exemplo, seria válido usar [CA Cities] em consultas em relação a [Sales] e [Warehouse and Sales] cubos, mas se você usou em uma consulta contra o cubo [Warehouse] você receber um erro, porque [Warehouse] não tem uma dimensão [Store].

6. Plug-ins

Às vezes linguagem de esquema de Mondrian não é suficientemente flexível, ou a linguagem MDX não é suficientemente poderoso, para resolver o problema facilmente. O que você quer fazer é adicionar um pouco de seu próprio código Java no aplicativo Mondrian e um plug-in é uma maneira de fazer isso.

Cada uma das extensões de Mondrian é tecnicamente um Service Provider Interface (SPI); em suma, uma interface Java que você escrever código para implementar e que Mondrian vai chamar em tempo de execução. Você também precisa se registrar uma extensão (geralmente em algum lugar no seu arquivo schema.xml) e para garantir que ele aparece no classpath.

Os plug-ins incluem user-defined functions; cell, member and property formatters; dynamic schema processors e data source change listeners. Há suporte incompleta member readers e cell readers, no futuro, podemos apoiar dialetos SQL conectáveis.

Alguns plug-ins (função definida pelo usuário, formatador membro, formatador de propriedade, formatador de célula) pode ser implementado em uma linguagem de script como JavaScript. Neste caso, você não precisa escrever uma classe Java; você apenas coloque o código de script dentro de um elemento de script no arquivo de esquema Mondrian. Extensões implementadas em linguagens de script não, em geral, executar, bem como extensões implementadas em Java, mas eles são muito mais conveniente porque você não precisa compilar qualquer código. Basta modificar o código de script no arquivo de esquema Mondrian e re-carregar o esquema. O ciclo de código-debug-fix mais curto permite que você desenvolver seu aplicativo muito mais rápido. Depois de ter implementado o plug-in no script, se o desempenho ainda é uma preocupação, você pode traduzir o seu plug-in em Java.

Outras extensões incluem Dynamic datasource xmla servlet..

6.1 Funções definidas por usuário

Uma função definida pelo usuário deve ter um construtor público e implementar a interface função mondrian.spi.UserDefinedFunction. Por exemplo,

```
package com.example;

import mondrian.olap.*;
import mondrian.olap.type.*;
import mondrian.spi.UserDefinedFunction;

/**
 * A simple user-defined function which adds one to its argument.
 */
public class PlusOneUdf implements UserDefinedFunction {
	// public constructor
	public PlusOneUdf() {
	}
```

```
        public String getName() {
        return "PlusOne";
        }

        public String getDescription() {
        return "Returns its argument plus one";
        }

        public Syntax getSyntax() {
        return Syntax.Function;
        }

        public Type getReturnType(Type[] parameterTypes) {
        return new NumericType();
        }

        public Type[] getParameterTypes() {
        return new Type[] {new NumericType()};
        }

        public Object execute(Evaluator evaluator, Exp[] arguments) {
        final Object argValue = arguments[0].evaluateScalar(evaluator);
        if (argValue instanceof Number) {
                return new Double(((Number) argValue).doubleValue() + 1);
        } else {
                // Argument might be a RuntimeException indicating that
                // the cache does not yet have the required cell value. The
                // function will be called again when the cache is loaded.
                return null;
        }
        }

        public String[] getReservedWords() {
        return null;
        }
}
```

Declare em seu esquema

```
<Schema>
       …
       <UserDefinedFunction name="PlusOne" className="com.example.PlusOneUdf"/>
</Schema>
```

E use em uma instrução MDX:

```
WITH MEMBER [Measures].[Unit Sales Plus One]
       AS 'PlusOne([Measures].[Unit Sales])'
SELECT
       {[Measures].[Unit Sales]} ON COLUMNS,
       {[Gender].MEMBERS} ON ROWS
FROM [Sales]
```

Se uma função definida pelo usuário tem um construtor público com um argumento de cadeia, Mondrian vai passar no nome da função. Por quê? Isso permite que você defina duas ou mais funções definidas pelo usuário usando a mesma classe:

```
package com.example;

import mondrian.olap.*;
import mondrian.olap.type.*;
import mondrian.spi.UserDefinedFunction;

/**
 * A user-defined function which either adds one to or
 * subtracts one from its argument.
 */
public class PlusOrMinusOneUdf implements UserDefinedFunction {
       private final name;
       private final isPlus;
       // public constructor with one argument
       public PlusOneUdf(String name) {
       this.name = name;
       if (name.equals("PlusOne")) {
              isPlus = true;
       } else if (name.equals("MinusOne")) {
              isPlus = false;
       } else {
              throw new IllegalArgumentException("Unexpected name " + name);
       }
       }
       public String getName() {
       return name;
       }
       public String getDescription() {
       return "Returns its argument plus or minus one";
       }
       public Syntax getSyntax() {
       return Syntax.Function;
       }
       public Type getReturnType(Type[] parameterTypes) {
       return new NumericType();
       }
       public Type[] getParameterTypes() {
       return new Type[] {new NumericType()};
       }
       public Object execute(Evaluator evaluator, Exp[] arguments) {
       final Object argValue = arguments[0].evaluateScalar(evaluator);
```

```
        if (argValue instanceof Number) {
                if (isPlus) {
                return new Double(((Number) argValue).doubleValue() + 1);
                } else {
                return new Double(((Number) argValue).doubleValue() - 1);
                }
        } else {
                // Argument might be a RuntimeException indicating that
                // the cache does not yet have the required cell value. The
                // function will be called again when the cache is loaded.
                return null;
        }
        }
        public String[] getReservedWords() {
        return null;
        }
}
```

e registrar duas funções em seu esquema:

```
<Schema>
        …
        <UserDefinedFunction name="PlusOne"
className="com.example.PlusOrMinusOneUdf">
        <UserDefinedFunction name="MinusOne"
className="com.example.PlusOrMinusOneUdf">
</Schema>
```

Se você está cansado de escrever várias vezes as declarações de funções definidas pelo usuário em arquivos de esquema, você pode enpacotar a implementação das funções (Classes) definidas pelos usuários dentro de um arquivo jar, embarcado com o recurso META-INF/services/mondrian.spi.UserDefinedFunction. Este arquivo de recurso contém nomes de classe e interfaces interface de implementações de mondrian.spi.UserDefinedFunction, um nome por linha. Para mais detalhes, você pode olhar intosrc/mai/META-INF/services/mondrian.spi.UserDefinedFunction na distribuição fonte e na seção Service Provider da especificação de arquivos JAR. Funções definidas pelo usuário declarados por esta via estão disponíveis para todos os esquemas de Mondrian na JVM.

Cuidado: você não pode definir mais de uma implementações de função definida pelo usuário em uma classe quando você declarar funções definidas pelo usuário desta maneira. Uma função será carregado para cada classe, e dado o nome do método de getName() no retorno.

Funções definidas pelo usuário também pode ser implementado em uma linguagem de script, como JavaScript. Essas funções podem não funcionar tão bem como Java UDFs ou funções internas, mas eles são muito mais conveniente de implementar.

Para definir um UDF no script, use o elemento Script e incluir nele as seguintes funções:

- getName()
- getDescription()
- getSyntax()
- getParameterTypes()
- getReturnType(parameterTypes)
- execute(evaluator, arguments)

O getName(), getDescription(), getReservedWords() e getSyntax() são métodos opicionais ; getName () padrão para o atributo de nome no elemento UserDefinedFunction, getDescription() padrão para o nome, getReservedWords() retorna uma lista vazia, e getSyntax() padrão para mondrian.olap.Syntax.Function. Os outros métodos têm significados semelhantes aos do UserDefinedFunction SPI.

Aqui está um exemplo da função factorial como um UDF JavaScript:

```
<UserDefinedFunction name="Factorial">
	<Script language="JavaScript">
		function getParameterTypes() {
			return new Array(new mondrian.olap.type.NumericType());
		}
		function getReturnType(parameterTypes) {
			return new mondrian.olap.type.NumericType();
		}
		function execute(evaluator, arguments) {
			var n = arguments[0].evaluateScalar(evaluator);
			return factorial(n);
		}
		function factorial(n) {
			return n <= 1 ? 1 : n * factorial(n - 1);
		}
	</Script>
</UserDefinedFunction>
```

6.2 Membro Leitor

Um membro leitor é um meio de acesso dos membros. Hierarquias são geralmente como base em uma tabela de dimensão (um "braço" de um esquema em estrela), e, portanto, são preenchidos usando SQL. Mas mesmo se os dados não residir em um RDBMS, você pode fazê-lo aparecer como uma hierarquia escrevendo uma classe Java chamado um membro leitor personalizado.

Aqui estão alguns exemplos:

1. DateSource (a ser escrito) gera uma hierarquia de tempo. Convencionalmente, os implementadores de data warehouse gerar uma

tabela que contém uma linha para cada data de seu sistema é recomendado lidar com eles. Mas o problema é que essa tabela precisa ser carregada, e como o passar do tempo, eles vão ter que lembrar de adicionar mais linhas. DateSource gera membros de data na memória e sob demanda.

2. FileSystemSource (a ser escrito) apresenta o sistema de arquivos com uma hierarquia de diretórios e arquivos. Uma vez que um diretório tenha um diretório pai , é uma hierarquia pai-filho. Como a hierarquia de tempo criado por DateSource, esta é uma hierarquia virtual: o membro de um determinado arquivo é criado somente quando, e se, directório principal desse arquivo é expandido.
3. ExpressionMemberReader (a ser escrito) cria uma hierarquia baseada em uma expressão.

Um membro leitor personalizado deve implementar a interface mondrian.rolap.MemberSource. Se você precisa implementar um conjunto maior de operações de membro para um controle mais refinado, implemente a interface derivada mondrian.rolap.MemberReader; caso contrário, o Mondrian envolverá o seu leitor em um objeto mondrian.rolap.CacheMemberReader. Seu leitor membro deve ter um construtor público que aceita parâmetros (RolapHierarchy, Properties) e não lança exceções verificadas.

Membros leitores são declarados usando o <Hierarchy> elementos membro do atributo Leitor Classe; qualquer <Parameter> de elementos filho são passadas através do parâmetro propriedades construtor. Aqui está um exemplo:

```
<Dimension name="Has bought dairy">
	<Hierarchy hasAll="true"
	memberReaderClass="mondrian.rolap.HasBoughtDairySource">
		<Level name="Has bought dairy" uniqueMembers="true"/>
		<Parameter name="expression" value="not used"/>
	</Hierarchy>
</Dimension>
```

6.3 Leitor de Células

Ainda não implementado. Sintaxe seria algo como

```
<Measure name="name" cellReaderClass="com.example.MyCellReader"/>
```

e a classe "com.example.MyCellReader" teria que implementar a interface mondrian.olap.CellReader.

6.4 Formatador de Células

O formatador de células modifica o comportamento de Cell.getFormattedValue(). A classe precisa implementar a interface mondrian.spi.CellFormatter, e é especificado como este:

```
<Measure name="name">
        <CellFormatter className="com.example.MyCellFormatter"/>
</Measure>
```

(A sintaxe anterior, usando o atributo "formatador" do elemento Medida, é obsoleto e será removido em Mondrian-4.0.)

Você pode especificar um formatador em uma linguagem de script como JavaScript, usando o elemento Script:

```
<Measure name="name">
        <CellFormatter>
                <Script language="JavaScript">
                </Script>
        </CellFormatter>
</Measure>
```

O script tem disponível um valor variável, correspondente ao parâmetro do método mondrian.spi.CellFormatter.formatCell(Object value). O fragmento de código pode ter várias instruções, mas deve terminar em uma instrução de retorno.

Para um membro calculado que pertence a um cubo ou cubo virtual, você também pode usar o elemento CellFormatter:

```
<CalculatedMember name="name" dimension="dimension">
        <Formula>
                [Measures].[Unit Sales] * 2
        </Formula>
        <CellFormatter>
                <Script language="JavaScript">
                        var s = value.toString();
                        while (s.length() < 20) {
                        s = "0" + s;
                        }
                        return s;
                </Script>
        </CellFormatter>
</Measure>
```

Você também pode encontrar um formatador, definindo a propriedade CELL_FORMATTER do membro para o nome da classe de formatador.

```
<CalculatedMember name="name" formatter="com.example.MyCellFormatter">
      <CalculatedMemberProperty name="CELL_FORMATTER"
        value="com.example.MyCellFormatter"/>
</CalculatedMember>
```

Para uma medida calculada definido na cláusula WITH MEMBER de uma consulta MDX, yvocê pode definir a mesma propriedade no MDX para alcançar o mesmo efeito:

```
WITH MEMBER [Measures].[Foo]
  AS '[Measures].[Unit Sales] * 2',
   CELL_FORMATTER='com.example.MyCellFormatter'
SELECT {[Measures].[Unit Sales], [Measures].[Foo]} ON COLUMNS,
       {[Store].Children} ON ROWS
FROM [Sales]
```

Para definir um script formatador, use as propriedades CELL_FORMATTER_SCRIPT e CELL_FORMATTER_SCRIPT_LANGUAGE:

```
WITH MEMBER [Measures].[Foo]
  AS '[Measures].[Unit Sales] * 2',
   CELL_FORMATTER_SCRIPT_LANGUAGE='JavaScript',
   CELL_FORMATTER_SCRIPT='var s = value.toString(); while (s.length() < 20)
s = "0" + s; return s;'
SELECT {[Measures].[Unit Sales], [Measures].[Foo]} ON COLUMNS,
       {[Store].Children} ON ROWS
FROM [Sales]
```

A propriedade formatador de célula é ignorado se um membro não pertencem à dimensão [Measures].

6.5 Member formatter

A member formatter modifies the behavior of Member.getCaption(). The class must implement the mondrian.spi.MemberFormatterinterface, and is specified like this:

```
<Level name="name" column="column">
       <MemberFormatter className="com.example.MyMemberFormatter"/>
</Level>
```

(A sintaxe anterior, usando o atributo "'formatter'" do elemento Nível, é obsoleto e será removido em Mondrian-4.0.)

Você pode especificar um formatador em uma linguagem de script como JavaScript, usando o elemento Script:

```
<Level name="name" column="column">
       <MemberFormatter>
              <Script language="JavaScript">
                     return member.getName().toUpperCase();
              </Script>
       </MemberFormatter>
</Level>
```

O Script tem disponível uma variável de membro, correspondente ao parâmetro do metodo mondrian.spi.MemberFormatter.formatMember(Member member). O fragmento de código pode ter várias instruções, mas deve terminar em uma instrução de retorno.

6.6 Formatador de propriedade

Um formatador de propriedade modifica o comportamento de Property.getPropertyFormattedValue(). A classe deve implementar a interface mondrian.spi.PropertyFormatter, e é especificada como isso:

```
<Level name="MyLevel" column="LevelColumn" uniqueMembers="true">
        <Property name="MyProp" column="PropColumn">
                <PropertyFormatter className="com.example.MyPropertyFormatter"/>
        </Property
<Level/>
```

(A sintaxe anterior, utilizando o atributo 'formatter' do elemento de propriedades, é obsoleto e será removido em Mondrian-4.0.)

Você pode especificar um formatador em uma linguagem de script como JavaScript, usando o elemento Script:

```
<Level name="name" column="column">
        <Property name="MyProp" column="PropColumn">
                <PropertyFormatter>
                        <Script language="JavaScript">
                                return member.getName().toUpperCase();
                        </Script>
                </PropertyFormatter>
        </Property>
</Level>
```

O Script tem variáveis de membro, e as variaveis propertyName propertyValue, correspondendo com os parametros do metodo mondrian.spi.PropertyFormatter.formatProperty(Member member, String propertyName, Object propertyValue). O fragmento de código pode ter várias instruções, mas deve terminar em uma instrução de retorno.

6.7 Processador do Esquema

O processodor de esquema implementa a interface mondrian.spi.DynamicSchemaProcessor. Ele é especificado como parte da cadeia de conexão, como este:

```
Jdbc=jdbc:odbc:MondrianFoodMart; JdbcUser=ziggy; JdbcPassword=stardust;
DynamicSchemaProcessor=com.example.MySchemaProcessor
```

O efeito é que, quando a leitura do conteúdo do esquema de um URL, Mondrian vira-se para o processador de esquema em vez de manipulador URL

padrão do Java. Isto dá ao leitor esquema a oportunidade de executar um esquema através de um filtro, ou mesmo gerar um esquema inteiro ao vivo.

Quando DynamicSchemaProcessor é especificado, o esquema será processado e recarregado em cada requisição de conexão ROLAP. PropertyUseContentChecksum deve ser usado juntamente com um processador de esquema para habilitar o cache do esquema:

```
DataSource=java:/jdbc/MyWarehouse;
DynamicSchemaProcessor=com.example.MySchemaProcessor; UseContentChecksum=true
```

Neste caso o esquema será carregado apenas uma vez em cache até a sua alteração. Se o conteúdo do esquema mudou, será recarregada (e processado).

Esquemas dinâmicos são uma construção muito poderoso. Como veremos, uma aplicação importante para eles é a internacionalização.

6.8 Ouvinte de alteração de Data source

Um ouvinte de alteração de fonte de dados implementa a interface mondrian.spi.DataSourceChangeListener. Ele é especificado como parte da cadeia de conexão, como este:

```
Jdbc=jdbc:odbc:MondrianFoodMart; JdbcUser=ziggy; JdbcPassword=stardust;
DataSourceChangeListener=com.example.MyChangeListener;
```

Toda vez que Mondrian tem de decidir se ele vai usar dados de cache, ele irá chamar o ouvinte mudança. Quando o ouvinte de alteração diz ao mondrianque a fonte de dados mudou para uma dimensão, cubo, ... então mondrian vai liberar o cache e ler a partir do banco de dados novamente.

Esta classe deve ser chamada no Mondrian antes que qualquer dado seja lido, por isso mesmo antes do cache ser construído. Desta forma, o plugin é capaz de registrar o primeiro mondrian timestamp tenta ler a fonte de dados.

Cada vez que uma consulta for iniciada, o cache agregado é verificado para ver se ele mudou. Se assim for, o cache será liberado e agregados será recarregado a partir da fonte de dados.

Aqui está um exemplo de uma classe mudança ouvinte plug-in fonte de dados:

```
package com.example;

//...
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Timestamp;
//...
import mondrian.olap.MondrianDef;
import mondrian.rolap.RolapHierarchy;
import mondrian.rolap.RolapUtil;
import mondrian.rolap.agg.Aggregation;
import mondrian.rolap.RolapStar;
import mondrian.spi.impl.DataSourceChangeListenerImpl;
//...

public class MyChangeListener extends DataSourceChangeListenerImpl {
    public MyChangeListener() {
    }
    public synchronized boolean isHierarchyChanged(RolapHierarchy hierarchy) {

          // Since this function is called many times, it is a good idea to not
check the database every time
        // And use some sort of time interval...

        // Get name of the table (does not work if based on view)
        String tableName = getTableName(hierarchy);
        Connection jdbcConnection = null;

        DataSource dataSource =
        hierarchy.getRolapSchema().getInternalConnection().getDataSource();

        try {
            jdbcConnection = dataSource.getConnection();
            if (jdbcConnection != null) {
                // Check database whether hierarchy data source has changed
                // ...
            }
        }
    }

    public synchronized boolean isAggregationChanged(Aggregation aggregation) {

          // The first time, register star and bitKey and remember first time of
access...
        RolapStar star = aggregation.getStar();
        BitKey bitKey = aggregation.getConstrainedColumnsBitKey();
```

```
            // The first time this function is called, only the bitKey is set,
            // the columns are not filled up yet.
            RolapStar.Column[] columns = aggregation.getColumns();
            if (columns != null) {
                // Check database...
            }
        }
    }
```

6.9 Fonte de dados dinâmica xmla servlet

O DynamicDatasourceXmlaServlet estende DefaultXmlaServlet, adicionando a capacidade de carregar dinamicamente o arquivo datasources.xml. Para cada solicitação de cliente que recebe, ele verifica se há atualizações para o conteúdo de datasources.xml. Ela limpa seletivamente cache para catálogos que foram alterados ou não existem mais no datasources.xml. O servlet considera um catálogo como alterado quando uma das suas propriedades (DataSourceInfo, propriedades de definição sobre DataSourcesConfig.Catalog) são diferentes. Ele identifica catálogo pelo nome.

Este servlet complementa a capacidade de carga catálogo dinâmico com base em UseContentChecksum. Ele não verifica o conteúdo do catálogo de atualizações. Não há sobreposição na funcionalidade. Os dois juntos vai dar origem de dados dinâmica completa e capacidade de configuração catálogo.

Para usar DynamicDatasourceXmlaServlet, altere a definição do MondrianXmlaServlet servlet no web.xml:

```
<servlet>
   <servlet-name>MondrianXmlaServlet</servlet-name>
   <servlet-class>mondrian.xmla.impl.DynamicDatasourceXmlaServlet</servlet-class>
       ...
</servlet>
```

Esta implementação tem uma limitação. Exige nome de catálogo para ser único em todas as fontes de dados, caso contrário pode não funcionar corretamente.

7. Internacionalização

Uma aplicação Mondrian internacionalizada teria um esquema para cada língua, onde a legenda de cada objeto aparece no idioma local. Por exemplo, o [Produto] dimensão teria a legenda "Product" em Inglês e "Produit" em francês.

Não é prudente para traduzir os nomes reais dos objetos de esquema, porque então as instruções MDX precisaria ser alterado também. Tudo o que você precisa mudar é a legenda. Cada objeto de esquema (schema, cube, virtual cube, dimension, hierarchy, level, measure, named set) em um atributo de legenda e interfaces de usuário, tais como JPivot e Pentaho Analyzer exibir a legenda em vez do nome real. Além disso:

- Cada objeto de esquema tem um atributo descrição.
- A hierarquia pode ter um atributo allMemberCaption que mostrará o valor do membro "All".
- Para o esquema, podemos definir um valor da dimensão "measures" mostradas pelo atributo measuresCaption.
- Um membro calculado tem as propriedades CAPTION e DESCRIPTION that appear as caption and description que aparecerá como legenda e descrição, se o membro é uma medida (ou seja, um membro da dimensão Medidas).

Uma maneira de criar uma aplicação internacionalizada é criar uma cópia do arquivo de esquema para cada idioma, mas estes são difíceis de manter. A melhor maneira é usar a classe LocalizingDynamicSchemaProcessor para realizar a substituição dinâmica em um único arquivo de esquema.

7.1 Localizando processador do esquema

Primeiro, escreva o seu esquema utilizando variáveis como valores para subtítulo, descrição, allMemberCaption e measuresCaption da seguinte forma:

```
<Schema measuresCaption="%{foodmart.measures.caption}">
      <Dimension name="Store" caption="%{foodmart.dimension.store.caption}"
      description="%{foodmart.dimension.store.description}">
            <Hierarchy hasAll="true" allMemberName="All Stores"
            allMemberCaption="%{foodmart.dimension.store.allmember.caption =All
            Stores}" primaryKey="store_id"
            caption="%{foodmart.hierarchy.store.country.caption}"
            description="%{foodmart.hierararchy.store.country.description}>
                  <Table name="store"/>
                  <Level name="Store Country" column="store_country"
                  uniqueMembers="true"
                  caption="%{foodmart.dimension.store.country.caption}"
                  description="%{foodmart.dimension.store.country.description}"/>
```

```
                <Level name="Store State" column="store_state"
                uniqueMembers="true"
                caption="%{foodmart.dimension.store.state.caption}"
                description="%{foodmart.dimension.store.state.description}"/>
                <Level name="Store City" column="store_city"
                uniqueMembers="false"
                caption="%{foodmart.dimension.store.city.caption}"
                description="%{foodmart.dimension.store.city.description}"/>
                <Level name="Store Name" column="store_name"
                uniqueMembers="true"
                caption="%{foodmart.dimension.store.name.caption}"
                description="%{foodmart.dimension.store.name.description}">
                        <Property name="Store Type" column="store_type"
                        caption="%{foodmart.dimension.store.
                        name.property_type.caption}"
                        description="%{foodmart.dimension.store.
                        name.property_type.description}"/>
                        <Property name="Store Manager" column="store_manager"
                        caption="%{foodmart.dimension.store.
                        name.property_manager.caption}"
                        description="%{foodmart.dimension.store.
                        name.property_manager.description}"/>
                        <Property name="Store Sqft" column="store_sqft"
                        type="Numeric" caption="%{foodmart.dimension.store.
                        name.property_storesqft.caption}"
                        description="%{foodmart.dimension.store.
                        name.property_storesqft.description}"/>
                        <Property name="Grocery Sqft" column="grocery_sqft"
                        type="Numeric"/>
                        <Property name="Frozen Sqft" column="frozen_sqft"
                        type="Numeric"/>
                        <Property name="Meat Sqft" column="meat_sqft"
                        type="Numeric"/>
                        <Property name="Has coffee bar" column="coffee_bar"
                        type="Boolean"/>
                        <Property name="Street address"
                        column="store_street_address" type="String"/>
                </Level>
        </Hierarchy>
</Dimension>
```

```
        <Cube name="Sales" caption="%{foodmart.cube.sales.caption}"
        description="%{foodmart.cube.sales.description}">
                …
                <DimensionUsage name="Store" source="Store" foreignKey="store_id"
                caption="%{foodmart.cube.sales.name.caption}"
                description="%{foodmart.cube.sales.name.description}"/>
                …
                <Measure name="Unit Sales" column="unit_sales"
                caption="%{foodmart.cube.sales.measure.unitsales.caption}"
                description="%{foodmart.cube.sales.measure.unitsales.description}"/>
        </Cube>
</Schema>
```

Como de costume, a legenda padrão para qualquer cube, measure, dimension or level sem um atributo de Caption é o nome do elemento. O Caption padrão de uma hierarquia é o Caption da sua dimensão; por exemplo, a hierarquia [Store] não tem Caption definido, então ele herda o atributo caption attribute do seu parente, a dimensão [Store].

Em seguida, adicione o processador esquema dinâmico e a localidade no seu connect string. Por exemplo,

```
Provider=mondrian; Locale=en_US;
DynamicSchemaProcessor=mondrian.i18n.LocalizingDynamicSchemaProcessor;Jdbc=jdbc:odb
c:MondrianFoodMart; Catalog=/WEB-INF/FoodMart.xml
```

Agora, para cada localidade que deseja apoiar, forneça arquivo locale_{locale}.properties. Por exemplo,

```
# locale.properties: Recurso padrão
foodmart.measures.caption=Measures
foodmart.dimension.store.country.caption=Store Country
foodmart.dimension.store.name.property_type.column= store_type
foodmart.dimension.store.country.member.caption= store_country
foodmart.dimension.store.name.property_type.caption =Store Type
foodmart.dimension.store.name.caption =Store Name
foodmart.dimension.store.state.caption =Store State
foodmart.dimension.store.name.property_manager.caption =Store Manager
foodmart.dimension.store.name.property_storesqft.caption =Store Sq. Ft.
foodmart.dimension.store.allmember.caption =All Stores
foodmart.dimension.store.caption =Store
foodmart.cube.sales.caption =Sales
foodmart.dimension.store.city.caption =Store City
foodmart.cube.sales.measure.unitsales =Unit Sales
```

and

```
# locale_hu.properties: Resources for the 'hu' locale.
```

```
foodmart.measures.caption=Hungarian Measures
foodmart.dimension.store.country.caption=Orsz\u00E1g
foodmart.dimension.store.name.property_manager.caption =\u00C1ruh\u00E1z
vezet\u0151
foodmart.dimension.store.country.member.caption =store_country_caption_hu
foodmart.dimension.store.name.property_type.caption =Tipusa
foodmart.dimension.store.name.caption =Megnevez\u00E9s
foodmart.dimension.store.state.caption =\u00C1llam/Megye
foodmart.dimension.store.name.property_type.column =store_type_caption_hu
foodmart.dimension.store.name.property_storesqft.caption =M\u00E9ret n.l\u00E1b
foodmart.dimension.store.allmember.caption =Minden \u00C1ruh\u00E1z
foodmart.dimension.store.caption =\u00C1ruh\u00E1z
foodmart.cube.sales.caption =Forgalom
foodmart.dimension.store.city.caption =V\u00E1ros
foodmart.cube.sales.measure.unitsales =Eladott db
```

8. Tabelas Agragadas

Tabelas agregadas são uma maneira de melhorar o desempenho do Mondrian quando a tabela de verdade contém um grande número de linhas: um milhão ou mais. Uma tabela agregada é, essencialmente, um resumo pré-computada dos dados na tabela de fatos.

Vamos olhar para uma simples tabela agregada.

```
<Cube name="Sales">
        <Table name="sales_fact_1997">
                <AggName name="agg_c_special_sales_fact_1997">
                        <AggFactCount column="FACT_COUNT"/>
                        <AggMeasure name="[Measures].[Store Cost]" column="STORE_COST_SUM"/>
                        <AggMeasure name="[Measures].[Store Sales]" column="STORE_SALES_SUM"/>
                        <AggLevel name="[Product].[Product Family]" column="PRODUCT_FAMILY"/>
                        <AggLevel name="[Time].[Quarter]" column="TIME_QUARTER"/>
                        <AggLevel name="[Time].[Year]" column="TIME_YEAR"/>
                        <AggLevel name="[Time].[Quarter]" column="TIME_QUARTER"/>
                        <AggLevel name="[Time].[Month]" column="TIME_MONTH"/>
                </AggName>
        </Table>

<!-- Restante das definições do cubo -->
</Cube>
```

O elemento <AggForeignKey>, não mostrado aqui, permite fazer referência a uma tabela de dimensão diretamente, sem incluir as colunas na tabela agregada. É descrito no guia de agregação de tabelas.

Na prática, um cubo baseado em uma tabela de fatos muito grande pode ter várias tabelas agregadas. É inconveniente declarar cada tabela agregada explicitamente no arquivo XML schema, e felizmente há uma maneira melhor. No exemplo a seguir, Mondrian localiza tabelas agregadas por pattern-matching.

```
<Cube name="Sales">
        <Table name="sales_fact_1997">
                <AggPattern pattern="agg_.*_sales_fact_1997">
                        <AggFactCount column="FACT_COUNT"/>
                        <AggMeasure name="[Measures].[Store Cost]"
                         column="STORE_COST_SUM"/>
                        <AggMeasure name="[Measures].[Store Sales]"
                         column="STORE_SALES_SUM"/>
                        <AggLevel name="[Product].[Product Family]"
                         column="PRODUCT_FAMILY"/>
                        <AggLevel name="[Time].[Quarter]" column="TIME_QUARTER"/>
```

```
                    <AggLevel name="[Time].[Year]" column="TIME_YEAR"/>
                    <AggLevel name="[Time].[Quarter]" column="TIME_QUARTER"/>
                    <AggLevel name="[Time].[Month]" column="TIME_MONTH"/>
                    <AggExclude name="agg_c_14_sales_fact_1997"/>
                    <AggExclude name="agg_lc_100_sales_fact_1997"/>
                </AggPattern>
            </Table>
</Cube>
```

Diga ao Mondrian como tratar todas as tabelas que correspondem ao padrão "agg_.*_sales_fact_1997" como tabelas agregadas, exceto "agg_c_14_sales_fact_1997" e "agg_lc_100_sales_fact_1997". Mondrian usa regras para deduzir os papéis das colunas nessas tabelas, por isso é importante a aderir a convenções de nomenclatura estritas. As convenções de nomenclatura são descritos no guia de agregação de tabelas.

O guia de desempenho tem conselhos sobre como escolher tabelas agregadas.

9. Controle de Acesso

OK, então agora você tem todo este grande quantidade de dados, mas você não que todos sejam capazes de ler tudo isso. Para resolver isso, você pode definir um perfil de controle de acesso, chamado de função, como parte do esquema, e definir este papel ao estabelecer uma conexão.

9.1 Definindo uma regra

Regras são definidas por elementos <Role>, que ocorrem como filhos diretos do elemento <Schema>, depois do último <Cube>. Aqui está um exemplo de regras:

```
<Role name="California manager">
    <SchemaGrant access="none">
        <CubeGrant cube="Sales" access="all">
            <DimensionGrant dimension="[Measures]" 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]">
                <MemberGrant member="[Customers].[USA].[CA]"
                access="all"/>
                <MemberGrant member="[Customers].[USA].[CA].[Los
                Angeles]" access="none"/>
            </HierarchyGrant>
            <HierarchyGrant hierarchy="[Gender]" access="none"/>
        </CubeGrant>
    </SchemaGrant>
</Role>
```

O <SchemaGrant> define o padrão de acesso para objetos em um esquema. O atributo de acesso pode ser "all" ou "none"; esse acesso pode ser substituído por objetos específicos. Neste caso, porquê o acesso = "none", um usuário só seria capaz de navegar no cubo "Sales", porque ele é explicitamente concedido..

O <CubeGrant> define o acesso a um determinado cubo. Quanto <SchemaGrant>, o atributo de acesso pode ser "all", "custom" ou "none", e pode ser substituído por sub-objetos específicos no cubo.

O <DimensionGrant> define o acesso a uma dimensão. O atributo de acesso pode ser "all", "custom" ou "none". Um nível de acesso de "all" significa que todas os

filhos hierarquias da dimensão vai ter herdado de acesso. Um nível de acesso de "custom" significa que o papel não herdar acesso aos filhos hierarquicos, a menos que o papel seja explicitamente concedido usando um elemento <HierarchyGrant>.

O <HierarchyGrant> define o acesso a uma hierarquia. O atributo de acesso pode ser "all", o que significa que todos os membros são visíveis; "None", ou seja, a própria existência da hierarquia é escondido do usuário; e "custom". Com acesso personalizado, você pode usar o atributo de nível superior para definir o nível superior que é visível (impedindo que os usuários possam ver muito do "big picture", como receitas de visualização rolou até o nível loja País); ou usar o atributo bottomLevel para definir o nível de fundo que é visível (aqui, impedindo que os usuários de invadir olhando detalhes dos clientes individuais); ou controle de quais conjuntos de membros que o usuário pode ver, definindo elementos aninhados <MemberGrant>.

Você pode definir um elemento <MemberGrant> se este estiver dentro <HierarchyGrant> com acesso "custom". Membro podem dar (dar ou remover) o acesso a um determinado, e a todos os seus filhos: Aqui estão algumas regras

1. Membros podem herdar o acesso de seus pais. Se você negar o acesso para a Califórnia, você não será capaz de ver San Francisco.
2. As concessões são dependentes de ordem. Se você conceder acesso aos EUA, em seguida, negar o acesso a Oregon, então você não será capaz de ver Oregon, ou Portland. Mas se você fosse para negar acesso a Oregon, em seguida, conceder acesso aos EUA, você pode efetivamente ver tudo.
3. Um membro é visível se qualquer um dos seus filhos são visíveis. Suponha que você negar o acesso aos EUA, em seguida, conceder acesso para a Califórnia. Você será capaz de ver EUA, e na Califórnia, mas nenhum dos outros estados. Os totais contra EUA ainda refletirá todos os estados, no entanto. Se o HierarchyGrant pai especifica um nível superior, apenas os pais iguais ou inferiores a este nível será visível. Da mesma forma, se um nível inferior está especificado, apenas as crianças acima ou igual ao nível são visíveis.
4. Acesso de membros não substituem acesso acima ou abaixo da subvenção hierarquia. Se você definir TopLevel = "[Store]. [Store State]", e conceder acesso a Califórnia, você não será capaz de ver EUA. subsídios dos membros não substituir o nível superior e atributos inferior. Você pode conceder ou negar acesso a um membro de qualquer nível, mas a parte superior e inferior restrições têm precedência sobre as bolsas membro explícito.

No exemplo, o usuário terá acesso a Califórnia, e todas as cidades da Califórnia, exceto Los Angeles. Eles serão capazes de ver EUA (porque o seu filho, Califórnia, é visível), mas há outras nações, e nem todas as lojas (porque está acima do nível superior, Story Country).

9.2 Política de Agragação

Uma política de agregação determina como Mondrian calcula total de um membro, se o papel atual não pode ver todos os filhos desse membro. No âmbito da política de agregação padrão, chamado de "full", o total para esse membro inclui contribuições de filhos que não são visíveis. Por exemplo, suponha que Fred pertence a um papel que pode ver [USA].[CA] and [USA].[OR] but not [USA].[WA]. Se Fred faz uma consulta

```
SELECT {[Measures].[Unit Sales]} ON COLUMNS,
        {[[Store].[USA], Store].[USA].Children} ON ROWS
FROM [Sales]
```

the query returns

[Customer]	[Measures].[Unit Sales]
[USA]	266,773
[USA].[CA]	74,748
[USA].[OR]	67,659

Note that [USA].[WA] não é devolvido, por a política de controle de acesso, mas o total inclui o total dos Washington (124,366) que Fred não pode ver. Para algumas aplicações, esta não é adequada. Em particular, se a dimensão tem um pequeno número de membros, o usuário final pode ser capaz de deduzir os valores dos membros que eles não têm acesso.

Para remediar isso, um papel pode aplicar uma política de agregação diferente de uma hierarquia. A política descreve como um total é calculado para um determinado membro, se o actual papel só pode ver algumas das crianças que membro:

- Full. O total para esse membro inclui todas as crianças. Esta é a política padrão se você não especificar o atributo de rollupPolicy.
- Partial. O total para esse membro inclui crianças só acessíveis.
- Hidden. Se alguma das crianças são inacessíveis, o total é escondido.

No âmbito da política 'partial', o total [USA] é a soma dos filhos [CA] e [OR]:

[Customer]	[Measures].[Unit Sales]
[USA]	142,407
[USA].[CA]	74,748
[USA].[OR]	67,659

Sob a política ‘hidden’, o [USA] total é escondido porque um de seus filhos não é acessível:

```
[Customer]    [Measures].[Unit Sales]

[USA]         -

[USA].[CA]    74,748

[USA].[OR]    67,659
```

A política é especificado por papel e hierarquia. No exemplo a seguir, o papel vê totais parciais para o [Store] hierarquia, mas os totais completas para [Product].

```
<Role name="South Pacific manager">
      <SchemaGrant access="none">
            <CubeGrant cube="Sales" access="all">
                  <HierarchyGrant hierarchy="[Store]" access="custom"
                     rollupPolicy="partial" 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"
                     rollupPolicy="full" topLevel="[Customers].[State Province]"
                     bottomLevel="[Customers].[City]">
                  <MemberGrant member="[Customers].[USA].[CA]" access="all"/>
                  <MemberGrant member="[Customers].[USA].[CA].[Los Angeles]"
                     access="none"/>
                  </HierarchyGrant>
                  <HierarchyGrant hierarchy="[Gender]" access="none"/>
            </CubeGrant>
      </SchemaGrant>
</Role>
```

Este exemplo mostra também recursos existentes, tais como a forma de subvenções hierarquia pode ser restrita usando atributos de nível superior e / ou inferior, e como uma função pode ser impedido de ver uma hierarquia usando access = "none".

9.3 Regras de União

As regras de união combina várias funções, e tem a soma de seus privilégios.

As regras de união pode ver um objeto de esquema particular se um ou mais de seus papéis constituintes pode vê-lo. Da mesma forma, a política de agregação de um papel de união com respeito a uma hierarquia particular, é menos restritiva de todas as políticas de agregação dos papéis.

Aqui está um exemplo que mostra a sintaxe de um papel união.

```
<Role name="Coastal manager">
	<Union>
		<RoleUsage roleName="California manager"/>
		<RoleUsage roleName="Eastern sales manager"/>
	</Union>
</Role>
```

Os regras constituintes "Califórnia manager" e "Eastern sales manager" pode ser papéis regulares, funções definidas pelo usuário ou funções sindicais, mas eles devem ser declarados no início do arquivo de esquema. O papel de "Coastal manager" será capaz de ver qualquer membro que ou um "California manager" e "Eastern sales manager". Ele será capaz de ver todas as células na intersecção destes membros, mais ele vai ser capaz de ver as células que nem papel pode ver: Por exemplo, se apenas "Califórnia manager" pode ver [USA].[CA].[Fresno], e só "Eastern sales manager" ver a métrica [Sales Target], em seguida, "manager Coastal" será capaz de ver a meta de vendas para Fresno, que nenhum dos papéis constituintes têm acesso.

9.4 Definindo uma regra de ligação

Um papel só tem efeito quando está associado com uma ligação. Por padrão, as conexões têm um papel que lhes dá acesso a todos os cubo no esquema que de conexão.

A maioria dos bancos de dados tem regras de ligações (ou "grupos") com os usuários, e automaticamente atribuí-los quando os usuários fazem login. No entanto, Mondrian não tem a noção de usuários, então você tem que estabelecer a regra de uma maneira diferente. Há duas maneiras de fazer isso:

1. Na String de conexão. Se você especificar a palavra-chave papel na cadeia de ligação, a ligação vai adotar esse papel. Você pode especificar vários nomes de funções separadas por vírgula, e um papel união será criado; se um nome de papel contém uma vírgula, substitua-o com uma vírgula extra. Veja class DriverManager classe para exemplos de sintaxe de cadeia de ligação.
2. Programaticamente. Uma vez que seu pedido tenha estabelecido uma conexão, chamar o método Connection.setRole(Role). Você pode criar uma função programaticamente (veja interface Role e o developer's note link para mais detalhes), ou olhe um nivel acima para o metodo Schema.lookupRole(String).

10. Apêndice A: Elementos XML

Elementos	Descrição
<Schema>	Coleção de Cubes, Virtual cubes, Shared dimensions, and Roles.
Elementos Lógicos	
<Cube>	Uma coleção de dimensões e métricas, tudo centrado numa tabela de fatos.
<VirtualCube>	Um cubo definido por uma combinação de dimensões e metricas de um ou mais cubos. Uma metrica originada de outro cuo pode ser um <CalculatedMember>.
<CubeUsages>	Cubos de base que são importados para um cubo virtual
<CubeUsage>	A utilização de um cubo de base por um cubo virtual.
<VirtualCubeDimension>	Uso de uma dimensão por um cubo virtual.
<VirtualCubeMeasure>	O uso de uma medida por um cubo virtual.
<Dimension>	Dimensão.
<DimensionUsage>	Dimensão
<Hierarchy>	Hierarquia.
<Level>	Nível de uma hierarquia.
<KeyExpression>	expressão SQL utilizada como chave de o nível, em vez de uma coluna.
<NameExpression>	expressão SQL utilizada para calcular o nome de um membro, em vez de Level.nameColumn.
<CaptionExpression>	expressão SQL utilizada para calcular a legenda de um membro, em vez de Level.captionColumn.
<OrdinalExpression>	expressão SQL utilizada para classificar os membros de um nível, em vez de Level.ordinalColumn.

<ParentExpression>	expressão SQL utilizada para calcular uma medida, em vez de Level.parentColumn.
<Property>	propriedade do membro. A definição é contra uma hierarquia ou nível, mas a propriedade estará disponível para todos os membros.
<PropertyExpression>	expressão SQL utilizada para calcular o valor de uma propriedade, em vez de Property.column.
<Measure>	A medida..
<CalculatedMember>	Um membro cujo valor é calculado utilizando uma fórmula, definida como parte de um cubo.
<NamedSet>	Um conjunto cujo valor é calculado utilizando uma fórmula, definida como parte de um cubo.

Elementos Físicos

<Table>	Fato ou tabela dimensão
<View>	Define uma "tabela" usando uma consulta SQL, que pode ter variantes diferentes para diferentes bancos de dados subjacentes.
<Join>	Define uma "tabela" por aderir a um conjunto de consultas.
<InlineTable>	Define uma tabela utilizando um conjunto de dados em linha.
<Closure>	Mapeia uma hierarquia pai-filho em uma mesa de encerramento.

Tabelas Agregadas

<AggExclude>	Excluir uma tabela agregada candidato pelo nome ou correspondência de padrões.
<AggName>	Declara uma tabela agregada a ser correspondido pelo nome.
<AggPattern>	Declara um conjunto de tabelas agregadas por padrão de expressão regular.
<AggFactCount>	Especifica o nome da coluna na tabela agregada o candidato que contém o número de linhas da tabela de fatos.
<AggIgnoreColumn>	Diz Mondrian para ignorar uma coluna em uma tabela agregada.

<AggForeignKey>	Mapas de chave estrangeira na tabela de fatos a uma coluna de chave estrangeira na tabela agregada candidato.
<AggMeasure>	Mapeia uma medida para uma coluna na tabela do agregado candidato.
<AggLevel>	Mapeia um nível para uma coluna na tabela de agregado candidato.

Controle de Acesso

<Role>	Um perfil de controle de acesso.
<SchemaGrant>	Um conjunto de direitos a um esquema.
<CubeGrant>	Um conjunto de direitos a um cubo.
<HierarchyGrant>	Um conjunto de direitos a uma hierarquia e níveis dentro dessa hierarquia.
<MemberGrant>	Um conjunto de direitos a um membro e seus filhos.
<Union>	Definição de um conjunto de direitos como a união de um conjunto de papéis.
<RoleUsage>	Uma referência a uma função.

Extensões

<UserDefinedFunction>	Declara uma função definida pelo usuário.
<CellFormatter>	formatador de célula.
<MemberFormatter>	formatador-Membro.
<PropertyFormatter>	formatador propriedade.
<Script>	fragmento de script para implementar um SPI como a função definida pelo usuário, formatador membro ou formatador de célula.

Outros

<Annotations>	Suporte para anotações.
<Annotation>	propriedade definida pelo usuário ligado a um elemento de metadados.

<Parameter>	Parte da definição de uma hierarquia; passado para um MemberReader, se estiver presente.
<CalculatedMemberProperty>	Propriedade de um membro calculado.
<Formula>	Mantém o texto da fórmula dentro de um <NamedSet> ou <CalculatedMember>.
<ColumnDefs>	Suporte para <ColumnDef> elementos.
<ColumnDef>	Definição de uma coluna em um conjunto de dados <InlineTable>.
<Rows>	Suporte para <row> elementos.
<Row>	Linha em um conjunto de dados <InlineTable>.
<Value>	Valor de uma coluna em um conjunto de dados <InlineTable>.
<MeasureExpression>	expressão SQL utilizada para calcular uma medida, em vez ofMeasure.column.
<SQL>	A expressão SQL para um dialeto do banco de dados particular.