InterSystems IRIS Business Intelligence の上級モデリング

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

InterSystems IRIS Business Intelligence の上級モデリング
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

計算ディメンジョンの定義

計算ディメンジョンは強力で、メンバを実行時にクエリ経由で定義できる高度な InterSystems IRIS Business Intelligence モデル・オプションです。ここでは、これらを定義する方法を説明します。

計算ディメンジョンとその他の基本 Business Intelligence モデル要素との比較については、"モデル・オプションの概要" を参照してください。

重要　　計算ディメンジョンには計算メンバとの関連付けはありません。計算ディメンジョンは Business Intelligence に固有です。計算メンバは、MDX の標準的な概念です。

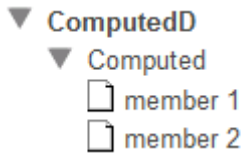
1.1 SQL 計算ディメンジョンの定義

SQL 計算ディメンジョンには、実行時に SQL クエリによって決定されるメンバが含まれます。メンバごとにクエリを指定します。

注釈　　NULL 置換オプションは、計算ディメンジョンに影響しません。

1.1.1 計算ディメンジョンの例

Patients キューブには `ComputedD` ディメンジョンがあります。これは、2 つのメンバ (`member 1` および `member 2`) を定義します。これらの各メンバは、特定の患者の ID を取得する SQL クエリによって定義されます。以下は、アナライザの左の領域に示すように、これらのメンバを示します。



このディメンジョン、`Computed` レベル、およびメンバは、キューブの他の要素を使用する場合と同様に使用できます。

同様に、HoleFoods キューブには、`Comments` ディメンジョンがあります。これは、2 つのメンバ `Complaints` および `Compliments` を定義します。これらのメンバは、SQL クエリによっても定義されます。

1.1.2 SQL 計算ディメンジョンの定義

以下の手順を使用すると、容易に SQL 計算ディメンジョンを定義できます。

1. アーキテクトを使用して、必要な名前でディメンジョン、階層、およびレベルを追加します。

   後で変更するため、選択するディメンジョンのタイプは重要でありません。

   ソース・プロパティやソース式は指定しないでください。

   これらの変更を完了したら、[保存] を選択します。

   この手順では、スタジオでの編集が容易なシェルを作成します。またアーキテクトでは、便宜上、表示名を初期化して名前と同じにします。

2. スタジオでキューブ・クラスを開きます。

3. 計算ディメンジョンに対応する <dimension> 要素を変更します。以下のように変更します。

   - type 属性を以下のように編集します。

     ```
     type="computed"
     ```

   - <dimension> 要素に以下のいずれかを追加します。

     ```
     dimensionClass="SQL"
     dimensionClass="%DeepSee.ComputedDimension.SQL
     ```

     dimensionClass 属性は、ヘルパー・クラスを参照します。完全なパッケージおよびクラス名を指定しないと、このクラスは %DeepSee.ComputedDimension パッケージ内にあると見なされます。%DeepSee.ComputedDimension.SQL クラスは、SQL ベースの計算ディメンジョンのヘルパー・クラスです。その他のタイプの計算ディメンジョンは、このドキュメントの対象ではありません。

   以下に例を示します。

   ```
   <dimension name="New Computed Dimension" displayName="New Computed Dimension"
       disabled="false"
       hasAll="false"
       type="computed"
       dimensionClass="SQL" >
   ```

   既定では、このようには改行されません。この例では読みやすくするために改行が追加されています。

4. レベルを定義するセクションを見つけます。

   ```
   <level name="New Computed Level" displayName="New Computed Level"
       disabled="false"
       list="false"
       useDisplayValue="true">
   </level>
   ```

5. </level> 行の直前で、以下のように行を追加します。

   ```
   <member name="" displayName=""  spec="" />
   ```

   以下のように編集します。

属性	値
name	メンバ名。
displayName	このメンバのオプションの表示名。
description	アナライザの左側の領域にツールヒントとして表示するオプションのテキスト。
spec	このキューブで使用されるファクト・テーブルの 1 つ以上のレコードの ID を返す SQL SELECT クエリ。ファクト・テーブルを参照するには、トークン $$$TABLE または完全なテーブル名を使用します。ファクト・テーブルの詳細は、"ファクト・テーブルおよびディメンジョン・テーブルの詳細" を参照してください。"spec のバリエーション" も参照してください。

以下に例を示します。

```
member name="example member 1" displayName="member 1"
spec="select ID from BI_Model_PatientsCube.Fact WHERE MxAge&lt;50 AND DxHomeCity->DxHomeCity='Elm
 Heights'" />
```

これらの属性で XML 予約文字を使用することはできません。置換については、"XML 予約文字" を参照してください。

6. その他の <member> 要素を必要に応じて追加します。

   これらの要素の順序によって、対応するメンバの既定の並べ替え順序が決まります。

7. クラスを保存し、リコンパイルします。

   これを実行するとすぐに、新しいディメンジョンとそのメンバを使用できるようになります。

1.1.3 サンプルの詳細

Patients キューブでは ComputedD ディメンジョンに以下のメンバが含まれています。

- member 1 は、以下の SQL クエリによって定義されます。

  ```
  select ID from BI_Model_PatientsCube.Fact WHERE MxAge&lt;50 AND DxHomeCity->DxHomeCity='Elm Heights'
  ```

  このクエリは、キューブのファクト・テーブルを使用します。詳細は、"ファクト・テーブルおよびディメンジョン・テーブルの詳細" を参照してください。

- member 2 は、以下の SQL クエリによって定義されます。

  ```
  select ID from $$$TABLE WHERE MxAge=40 AND DxHomeCity->DxHomeCity='Juniper'
  ```

  このクエリは $$$TABLE を使用します。これはファクト・テーブルの実際の名前に置き換えられます。前のクエリとは異なり、このクエリは $$$TABLE トークンを変換するための情報がないため、管理ポータルでは有効ではありません。

以下のピボット・テーブルに、これらのメンバが示されています。

Computed	Avg Age	Patient Count
member 1	24.50	801
member 2	40	18

例えば、この例の member 1 は、平均年齢 24.50 歳の 801 人の患者で構成されています。

1.1.4 spec のバリエーション

可能な spec のバリエーションは以下のとおりです。

- キューブのベース・クラスにクエリに適したインデックスがある場合は、代わりにそのクラスをクエリで参照することもできます。このクラスの ID は、ファクト・テーブルで使用される ID と同じです。

  これは、頻繁に変更されるグループを反映するディメンジョンが必要な場合やキューブの再構築や同期化を回避したい場合に役立ちます。

- 省略表現として、SELECT 文の代わりに WHERE 節を使用することができます。この場合は、WHERE 節を使用する SELECT 文が生成されます。この文では、ファクト・テーブルから ID を選択します。

  例は、このセクションで後述する計算ディメンジョンの例の By Industry レベルのメンバを参照してください。

- 省略表現として、トークン $$$FACT を使用してファクト・テーブル内のフィールドを参照できます。そのためには、レベルで、factName 属性にそのフィールドの名前を指定します。以下に例を示します。

  ```
  <level name="By Allergy Count" factName="Mx1968652733I" >
  ```

  次に、SELECT 文 (または WHERE 節) で、$$$FACT を使用してこのフィールドを参照します。このトークンが適切に解析されるように、トークンの前後に必ずスペースを挿入してください。

  例は、このセクションで後述する計算ディメンジョンの例の By Allergy Count レベルのメンバを参照してください。

- SELECT 文または WHERE 節を使用する代わりに、ストアド・プロシージャを呼び出す CALL 文を使用できます。以下に例を示します。

  ```
  spec="CALL MyStoredProcedure()"
  ```

  ストアド・プロシージャでは、メソッド・キーワード SqlProc およびReturnResultsets の両方を必ず True に指定してください。以下に例を示します。

  ```
  ClassMethod GetPatientIsDiabetic() As %Integer [ ReturnResultsets, SqlProc ]
  {
      // Note that %sqlcontext is created when SQLPROC is defined
      Try {
          Set rc = 1

          Set sql = "SELECT f.id id FROM HSAA_PatientCurrentConditionsCube.Fact f, HSAA.Diagnosis d
   "
                    _" WHERE d.Patient = f.%sourceid and d.Diagnosisgroup in ('249', '250', '253')"
          Set st = ##class(%SQL.Statement).%New()
          Set sc = st.%Prepare(sql)
          If ($$$ISERR(sc)) {
                  Set rc = 0
                  Quit
          }
          Set rs = st.%Execute()
          Do %sqlcontext.AddResultSet(rs)
      }
      Catch(ex) {
          Set %sqlcontext.%SQLCODE = ex.AsSQLCODE()
          Set %sqlcontext.%Message = ex.SQLMessageString()
          Set rc = 0
      }

      Quit rc
  }
  ```

以下の計算ディメンジョンは、Patients キューブで機能します。これを使用するには、コピーしてキューブ・クラスに貼り付け、そのキューブ・クラスをリコンパイルします。

```
<dimension name="Other Groups" displayName="Groups (Computed Dimension)"
 hasAll="false" type="computed" dimensionClass="SQL">
  <hierarchy name="H1">
```

```
   <level name="By Industry"  >
    <member name="Retail Trade"
    spec="WHERE DxIndustry->DxIndustry='Retail Trade'">
    </member>
    <member name="Finance and Insurance"
    spec="select ID from $$$TABLE WHERE DxIndustry->DxIndustry='Finance and Insurance'">
    </member>
   </level>
  </hierarchy>
  <hierarchy name="H2">
   <level name="By Allergy Count" factName="Mx1968652733I" >
    <member name="Highly allergic"
    spec="WHERE $$$FACT > 2">
    </member>
    <member name="Not allergic"
    spec="select ID from $$$TABLE WHERE $$$FACT = 0">
    </member>
   </level>
  </hierarchy>
</dimension>
```

1.2 MDX 計算ディメンジョンの定義

MDX 計算ディメンジョンには、実行時にMDX クエリによって決定されるメンバが含まれます。メンバごとにクエリを指定します。

注釈　　NULL 置換オプションは、計算ディメンジョンに影響しません。

1.2.1 MDX 計算ディメンジョンの定義

以下の手順を使用すると、容易に MDX 計算ディメンジョンを定義できます。

1. アーキテクトを使用して、必要な名前でディメンジョン、階層、およびレベルを追加します。

   後で変更するため、選択するディメンジョンのタイプは重要でありません。

   ソース・プロパティやソース式は指定しないでください。

   この手順では、スタジオでの編集が容易なシェルを作成します。またアーキテクトでは、便宜上、表示名を初期化して名前と同じにします。

2. スタジオでキューブ・クラスを開きます。

3. 計算ディメンジョンに対応する <dimension> 要素を変更します。以下のように変更します。

   - type 属性を以下のように編集します。

     ```
     type="computed"
     ```

   - 以下を <dimension> 要素に追加します。

     ```
     dimensionClass="MDX"
     ```

     dimensionClass 属性は、ヘルパー・クラスを参照します。完全なパッケージおよびクラス名を指定しないと、このクラスは %DeepSee.ComputedDimension パッケージ内にあると見なされます。

   以下に例を示します。

   ```
   <dimension name="AgeBuckets" type="computed" dimensionClass="MDX">
       <hierarchy name="H1">
          <level name="Years"/>
       </hierarchy>
   </dimension>
   ```

4. 以下のいずれかの方法でメンバを定義します。
   - 以下の例のように、<level> の子として <member> 要素を追加します。

     ```
     <member name="Boston" spec="[OUTLET].[CITY].[BOSTON]"/>
     ```

     <member> 要素を配置する場所の例については、"SQL 計算ディメンジョンの定義" を参照してください。
   - キューブ・クラスの %OnGetComputedMembers() コールバックを定義します。次のサブセクションを参照してください。

5. キューブ・クラスを保存してリコンパイルします。

   これを実行するとすぐに、新しいディメンジョンとそのメンバを使用できるようになります。

1.2.2 %OnGetComputedMembers()

%OnGetComputedMembers() コールバックには、以下のシグニチャがあります。

```
ClassMethod %OnGetComputedMembers(pDimName As %String,
                                  pHierName As %String,
                                  pLevelName As %String,
                                  ByRef pMemberList,
                                  pRollupKey As %String = "",
                                  ByRef pRange As %String = "") As %Status
```

pDimName、pHierName、および pLevelName はそれぞれ、ディメンジョン、階層、およびレベルの名前です。このメソッドは参照により pMemberList を返します。これは、以下の形式の多次元配列です。

配列ノード	配列値
pMemberList(i)。I は整数です。	指定レベルのメンバを定義する $LISTBUILD リスト。このリストには、以下の項目が以下の順序で含まれている必要があります。 ・ メンバを定義する MDX 式 ・ メンバの表示名 ・ メンバのキー

以下に例を示します。

Class Member

```
ClassMethod %OnGetComputedMembers(pDimName As %String, pHierName As %String, pLevelName As %String,
      ByRef pMemberList, pRollupKey As %String = "", ByRef pRange As %String = "") As %Status
{
    If (pDimName="AgeBuckets") {
        If (pLevelName="Years") {
            // $LB(MDX,Name,Key)
            Set pMemberList($I(pMemberList)) =
              $LB("%OR([DateOfSale].[Actual].[DaySold].[NOW-1y-1d]:[NOW])","1 year(s)","1")
            Set pMemberList($I(pMemberList)) =
              $LB("%OR([DateOfSale].[Actual].[DaySold].[NOW-2y-1d]:[NOW-1y])","2 year(s)","2")
            Set pMemberList($I(pMemberList)) =
              $LB("%OR([DateOfSale].[Actual].[DaySold].[NOW-3y-1d]:[NOW-2y])","3 year(s)","3")
            Set pMemberList($I(pMemberList)) =
              $LB("%OR([DateOfSale].[Actual].[DaySold].[NOW-4y-1d]:[NOW-3y])","4 year(s)","4")
            Set pMemberList($I(pMemberList)) =
              $LB("%OR([DateOfSale].[Actual].[DaySold].[NOW-5y-1d]:[NOW-4y])","5 year(s)","5")
            Set pMemberList($I(pMemberList)) =
              $LB("%OR([DateOfSale].[Actual].[DaySold].[NOW-6y-1d]:[NOW-5y])","6 year(s)","6")
            Set pMemberList($I(pMemberList)) =
              $LB("%OR([DateOfSale].[Actual].[DaySold].[NOW-7y-1d]:[NOW-6y])","7 year(s)","7")
```

```
        }
    }
    Quit $$$OK
}
```

2

共有ディメンジョンおよび複合キューブの定義

ここでは、Business Intelligence で使用するために、共有ディメンジョンおよび複合キューブを定義する方法を説明します。

複合キューブはアーキテクトで定義できますが、共有ディメンジョンについては IDE を使用する必要があります。

背景情報は、"モデル・オプションの概要" を参照してください。

"BI サンプルのアクセス方法" も参照してください。

2.1 概要

このセクションでは、共有ディメンジョンおよび複合キューブの概要について説明します。

2.1.1 共有ディメンジョン

共有ディメンジョンとは、複数のキューブで使用可能なディメンジョンです。共有ディメンジョンにより、以下の操作が可能になります。

- 両方のキューブのピボット・テーブルが組み込まれたダッシュボードを作成する。
- ダッシュボードで、共有ディメンジョンを使用するフィルタを組み込む。

  テーブルがフィルタのターゲットとして構成されている場合、このフィルタは両方のディメンジョンからのピボット・テーブルに作用します。
- 両方のキューブを使用するピボット・テーブルを作成する (次のセクションで説明する複合キューブも定義する場合のみ)。

通常、時間と場所に基づくディメンジョン (下記のメモを参照) は、関係を持たないキューブであっても共有可能です。

他のディメンジョンの共有が可能な場合もあります。例えば、あるキューブがトランザクションを、別のキューブがそのトランザクションを所有する顧客を表すとします。これら 2 つのキューブは、顧客の等級、ブローカなどの共通するディメンジョンを持つ可能性があります。

共有ディメンジョンは、以下の方法のいずれかで定義できます。

- ディメンジョンは、形式的に共有できます。この場合、ディメンジョンはあるキューブで定義され、別のキューブ (複数も可) で参照されます。

このようにする場合は、複合キューブも定義できます。このキューブは形式的にディメンジョンを共有している 2 つ以上のキューブを結合します。これによって、さまざまなキューブからの要素を単一のピボット・テーブルに結合することができます。

- ディメンジョンは非公式に共有できます。この場合、各キューブに独自のディメンジョン定義があり、これらのキューブは互いに独立しています。

  この場合に、キューブを複合キューブ内で一緒に使用することはできません。ただし、前述したように、両方のキューブからピボット・テーブルを含むダッシュボードを作成して、それらにフィルタを適用することができます。

2.1.2 複合キューブ

複合キューブとは、複数のキューブ（通常は 2 つ）を結合するサブジェクト領域です。これらのキューブでは、同名のディメンジョンのすべてが形式的に共有されたディメンジョンである必要があります。これによって、複数のキューブの要素が含まれるピボット・テーブルを作成できます。

複合キューブから作成されたピボット・テーブルを以下に示します。

ZIP	Doctor Count	Patients Per Week	Avg Patient Allergy Count	Avg Patient Test Score
32006	5	600	1.05	73.65
32007	9	1,035	1.11	76.48
34577	12	1,473	0.98	75.15
36711	5	747	1.03	76.01
38928	9	987	1.14	75.44

このピボット・テーブルの説明を以下に示します。

- `Doctor Count` メジャーと `Patients Per Week` メジャーは `Doctors` キューブから得たものです。

  `Patients Per Week` メジャーは、指定された医師のセットの診察を受ける 1 週間の患者数です。

- `Avg Patient Allergy Count` メジャーと `Avg Patient Test Score` メジャーは `Patients` キューブから得たものです。

  `CompoundCube` サブジェクト領域ではオーバーライドを定義しているため、これらのメジャーは、そのキューブ内とは異なる名前が付けられています。

- `ZIP` レベルは、これらのキューブの両方により使用される共有ディメンジョン内にあります。

複合キューブでは、最初にリストしたキューブのディメンジョンと、形式的に共有されたすべてのディメンジョンが、使用可能なディメンジョンになります。利用可能なメジャーには、すべてのキューブのすべてのメジャーが含まれます。以下のルールが適用されます。

- 複合キューブで使用されるすべてのキューブで名前が同じメジャーの場合、そのメジャーは共通のメジャーになります。このメジャーでは、値はすべてのキューブ全体で集約されます。例えば、あるキューブを Employees、他のキューブを Patients と仮定します。両方のキューブに Count メジャーがある場合、それらのカウントは一緒に集約されます。

  システムには Count メジャーの名前を変更するオプションが用意されているため、その集約が適切でない場合には、この集約の発生を回避できます。

- 1 つのキューブの中にのみ存在するメジャーの場合は、通常どおりに扱われます。

- 形式的に共有されるレベルの場合、そのレベルのメンバと任意のキューブを使用して、それらのレコードのサブセットを選択することができます。前述の例では、32,006 のメンバはこの ZIP コードを持つすべての医師と、この ZIP コードを持つすべての患者に対応します。

  これは、すべてのキューブのメジャーが、潜在的にそのレベルのメンバに対して異なる値を保持している可能性があることを意味します。例えば、メジャー `Patients Per Week`（医師固有）とメジャー `Avg Patient Allergy Count`（患者固有）では、ZIP コードごとに保持している値が異なります。

- 形式的に共有されないレベルの場合、そのレベルのメンバは、それを所有するキューブからレコードのサブセットを選択しますが、レコードはすべて他のキューブから選択します。

  これは、このレベルを定義するキューブのメジャーは、そのレベルのメンバに対して異なる値を潜在的に保持している可能性がありますが、その他のキューブのメジャーは常に同じ値を保持していることを意味します。以下の例では、Doctor Type ディメンジョンは共有されません。

Doctor Type	Doctor Count	Avg Patients Per Week	Patient Count	Patient Avg Age
Allergist	3	137.33	1,000	35.90
Anesthesiologist	1	148	1,000	35.90
Dermatologist	1	148	1,000	35.90
Emergency Physician	1	159	1,000	35.90
Gastroenterologist	4	153.75	1,000	35.90
General Physician	7	146	1,000	35.90
Internist	4	163	1,000	35.90
OB/GYN	6	141.50	1,000	35.90
Pediatrician	8	150.29	1,000	35.90
Radiologist	3	150.67	1,000	35.90
Surgeon	2	165.50	1,000	35.90

  Doctor Count メジャーと Avg Patients Per Week メジャー（どちらも医師に固有）は、医師のタイプごとに異なる値を持っています。

  その他のメジャーは患者に固有です。それらは各医師のタイプに対して同じ値を持ちます。この値は、すべての患者全体で集約されます。

2.2 形式的に共有されるディメンジョンの定義

形式的にディメンジョンを共有するには、以下の手順を実行します。

1. あるキューブ定義で通常どおりディメンジョンを定義します。

   そのキューブを構築するときに、システムは、通常の方法でそのディメンジョンのすべてのレベルの初期メンバを決定します。ソース・クラスが追加データを受信してキューブが更新されると、システムは通常の方法でレベルのメンバを追加します。

2. アーキテクトで別のキューブ定義を開き、以下のように共有ディメンジョンを追加します。

   a. **[要素を追加]** を選択します。

   b. **[共有ディメンジョン]** を選択します。

   c. 最初のドロップダウン・リストで、ディメンジョンを定義するキューブを選択します。

   d. 2 つ目のドロップダウン・リストで、ディメンジョンを選択します。

e. [OK] を選択します。

このディメンジョンのすべてのレベルについて、このキューブのファクト・テーブルは、もう一方のキューブのディメンジョン・テーブルをポイントします。

3. 必要に応じて、2 つ目のキューブで、共有ディメンジョンのレベルのソース・データ定義をオーバーライドします。

既定では、共有ディメンジョンは、　　で使用された、同じソース・プロパティまたはソース式を使用します。これらをオーバーライドするには、スタジオでクラスを編集し、該当する <dimension> 要素を検索し、子の <hierarchy> 要素および <level> 要素を必要に応じて追加します。"キューブ・クラスのリファレンス情報" を参照してください。この場合は、ディメンジョン名、階層名およびレベル名が、他のキューブでの名前と同じになる必要があります。

以下の制約が適用されます。

- ディメンジョンを所有するキューブは、最初に構築する必要があります。このプロセスでは、そのキューブで定義されたディメンジョンのテーブルが作成されます。他のキューブを構築すると、システムがレコードを処理する際に、あらゆる共有ディメンジョンのディメンジョン・テーブルにレコードが追加されます。
- 2 つ目のキューブのソース・データ定義をオーバーライドしない限り、同じレベル定義が両方のキューブに対して適切である必要があります。そのため、同一のソース・プロパティやソース式は、両方のキューブに適用できる必要があります。例えば、定義を所有するキューブがソース式 %source.Item.Category を使用する場合、このソース式は他方のキューブにも対応している必要があります。
- そのディメンジョンを共有するキューブは、共有ディメンジョンのレベルでソース値 (メンバ・キー) が同じである必要があります。

  例えば、それぞれが市区町村名が組み込まれたテーブルに基づく 2 つのキューブについて考えてみましょう。これらのキューブが市区町村名に基づくレベルを共有するには、両方のソース・テーブルで市区町村名がまったく同じ (大文字と小文字の区別も含めて) であることが必要です(そうでない場合、Jonesville と JONESVILLE のような、複数の類似したメンバが存在するようになります)。

  ただし、双方のソース・テーブルが同じ値のセットを持つ必要はありません。例えば、一方のソース・テーブルで、他方にない市区町村をリストしていてもかまいません。ディメンジョン・テーブルには、値の上位集合すべてが含まれます。

また、これらのレベルを使用するフィルタのすべてで、ディメンジョンを共有するすべてのキューブのすべてのメンバがメンバのリストに含まれます。このため、例えば指定のダッシュボードで、フィルタ・ドロップダウンに見慣れない市区町村名、つまりそのダッシュボードで使用されるデータにはない市区町村名が表示される場合があります。この市区町村名は選択できますが、一致するデータは見つかりません。

HoleFoodsBudget キューブおよび CompoundCube/Doctors キューブの両方に共有ディメンジョンの例が含まれています。これらの例は、相互に関連性はありません。

2.3 非公式に共有されるディメンジョンの定義

非公式に共有されるディメンジョンを定義するには、論理ディメンジョン名、その階層名、レベル名、およびメンバ・キーが、関連するすべてのキューブで同じであることを確認します (ソース式や変換オプションなどの基礎となる詳細は重要ではありません。論理名が一致すること、およびメンバ・キーが一致することが重要です)。

このとき、これらのキューブのそれぞれでピボット・テーブルを定義し、そのピボット・テーブルを同じダッシュボードに配置することができます。共有ディメンジョンのいずれかを使用するフィルタ・ウィジェットを組み込むと、すべてのピボット・テーブルに影響する場合があります。

2.3.1 例

Patients キューブ (クラス BI.Model.PatientsCube 内にあります) には、HomeD ディメンジョンが含まれています。このディメンジョンには、H1 階層が含まれています。この階層には、ZIP レベルと City レベルが含まれています。

CityRainfall キューブ (クラス BI.Model.RainfallCube 内にあります) にも、HomeD ディメンジョンが含まれています。このディメンジョンは Patients キューブ内のディメンジョンと以下の点のみが異なっています。

- HomeD ディメンジョンには表示名 CityD がある (内部名と同じではない)。
- HomeD ディメンジョンには All メンバがある。
- City レベルは City.Name ソース・プロパティを使用する (HomeCity.Name ではない)。
- ZIP レベルは City.PostalCode ソース・プロパティを使用する (HomeCity.PostalCode ではない)。

これらの定義は、同じダッシュボード上の異なるピボット・テーブルでこれらのキューブを使用できること、および HomeD ディメンジョンを使用するどのフィルタに対しても同様の反応を得られることを意味します。このことを、ダッシュボード Dashboards/Demo Two Subject Areas Together で示します。ここには、Patients キューブを使用するピボット・テーブルと、CityRainfall キューブを使用する別のピボット・テーブルがあります。このダッシュボードには、両方のピボット・テーブルに作用するフィルタ・コントロールが組み込まれています。

同様に、Cities キューブ (クラス BI.Model.CityCube 内) には、HomeD という名前のディメンジョンがあり、このディメンジョンには H1 階層があり、この階層には ZIP および City レベルがあります。HomeD の表示名は CityD で、このディメンジョンにはこのキューブで別の名前があるように見えます。前の説明と同様、Cities キューブと Patients キューブでは、レベルで使用されるソース・プロパティが異なります。

2.4 複合キューブの定義

複合キューブを作成するには、スタジオを使用する必要があります。複合キューブを作成するには、以下の手順をすべて実行します。

- [**ベース・キューブ**] オプションにキューブのコンマ区切りのリストを指定して、サブジェクト領域を作成します。例えば、サンプル・サブジェクト領域 CompoundCube/CompoundCube の場合、[**ベース・キューブ**] は以下のようになります。

  ```
  CompoundCube/Patients,CompoundCube/Doctors,CompoundCube/CityRainfall
  ```

  また、右側の詳細領域の [**依存**] オプションを編集します。値には、すべてのキューブ・クラスの完全なパッケージおよびクラス名を指定します。

  どのサブジェクト領域クラスも常に、基にしている 1 つまたは複数のキューブ・クラスの後に、コンパイルする必要があります。これを制御するには、[**依存**] 設定が役に立ちます。

- 複合キューブで使用するキューブで、必要に応じて Count メジャーを再定義します。これを実行するには、キューブの定義内で countMeasureName 属性と、countMeasureCaption 属性 (オプション) を指定します。以下に例を示します。

  ```
  <cube xmlns="http://www.intersystems.com/deepsee"
  name="Doctors"
  displayName="Doctors"
  sourceClass="BI.Study.Doctor"
  countMeasureName="DoctorCount"
  countMeasureCaption="Doctor Count">
  ...
  ```

この変更には、これらのキューブの再構築は必要ありません。

- 複合キューブではオプションとして、メジャー名の表示名を、複合キューブで使用する際により特定しやすい名前に変更することができます。以下に例を示します。

XML

```
<subjectArea xmlns="http://www.intersystems.com/deepsee/subjectarea"
name="CompoundCube" displayName="CompoundCube"
baseCube="Doctors,Patients">

<measure name="Allergy Count" displayName="Patient Allergy Count"/>
<measure name="Avg Allergy Count" displayName="Patient Avg Allergy Count"/>
<measure name="Age" displayName="Patient Age"/>
<measure name="Avg Age" displayName="Patient Avg Age"/>
<measure name="Test Score" displayName="Patient Test Score"/>
<measure name="Avg Test Score" displayName="Patient Avg Test Score"/>
<measure name="Encounter Count" displayName="Patient Encounter Count"/>
<measure name="Avg Enc Count" displayName="Patient Avg Enc Count"/>

</subjectArea>
```

変更したすべてのキューブの定義をリコンパイルします。最後に、複合キューブをリコンパイルします。

複合キューブでは、最初にリストしたキューブのディメンジョンと、形式的に共有されたすべてのディメンジョンが、使用可能なディメンジョンになります。利用可能なメジャーには、すべてのキューブのすべてのメジャーが含まれます。

注釈　両方のキューブで同じ名前を持つディメンジョンは、形式的に共有される必要があります。両方のキューブで同じ名前を持つメジャーは、1 つに集約されます。

2.4.1 複合キューブの詳細リスト

複合キューブの詳細リストを定義するには、すべての参加キューブで同一の詳細リストを定義します。システムでこれらのリストの SQL UNION が生成されます。

リストは直接 SQL に基づく必要があります。データ・コネクタ経由の詳細リストは、複合キューブでは機能しません。

2.4.2 複合キューブの例

複合キューブの例は、クラス BI.Model.CompoundCube.CompoundCube を参照してください。このクラスは以下のように定義されています。

Class Definition

```
Class BI.Model.CompoundCube.CompoundCube Extends %DeepSee.SubjectArea
[ DependsOn = (BI.Model.CompoundCube.Patients, BI.Model.CompoundCube.Doctors,
BI.Model.CompoundCube.CityRainfall) ]
{

/// This XData definition defines the SubjectArea.
XData SubjectArea [ XMLNamespace = "http://www.intersystems.com/deepsee/subjectarea" ]
{
<subjectArea name="CompoundCube/CompoundCube"  displayName="CompoundCube/CompoundCube"
    baseCube="CompoundCube/Patients,CompoundCube/Doctors,CompoundCube/CityRainfall"  >
</subjectArea>
}

}
```

キューブ CompoundCube/Patients (BI.Model.CompoundCube.Patients で定義されている) は、すべてのディメンジョンを定義しています。

他のキューブ (CompoundCube/Doctors と CompoundCube/CityRainfall) は、CompoundCube/Patients と共有するディメンジョンを定義しています。すべてのディメンジョンがすべてのキューブで定義されているわけではないことに注意してください。以下のテーブルに、各キューブで使用可能なディメンジョンを示します。

ディメンジョン	CompoundCube/Patients キューブ	CompoundCube/Doctors キューブ	CompoundCube/CityRainfall キューブ
BirthD	✓		✓
DocD	✓	✓	
DocTypeD	✓	✓	
HomeD	✓	✓	✓

HomeD ディメンジョンは、3 つのキューブすべてで定義されているので、このディメンジョンは 3 つのキューブすべてのメジャーに影響します。例えば、ダッシュボード Demo Compound Cube には、以下のピボット・テーブルが含まれます。

Pivot table using dimension shared by all cubes

ZIP	City	Patient Count	Doctor Count	Avg Monthly Rainfall Inches
32006	Juniper	119	7	1.58
	Spruce	116	4	1.61
32007	Redwood	99	4	1.61
34577	Cypress	99	5	1.59
	Magnolia	125	2	1.63
	Pine	115	2	1.60
36711	Centerville	118	4	1.61
38928	Cedar Falls	101	6	1.59
	Elm Heights	108	6	1.59

Patient Count メジャーは CompoundCube/Patients で定義され、Doctor Count メジャーは CompoundCube/Doctors で定義され、Avg Monthly Rainfall Inches メジャーは CompoundCube/CityRainfall で定義されています。その値は、各市区町村の各メジャーで異なることに注意してください。

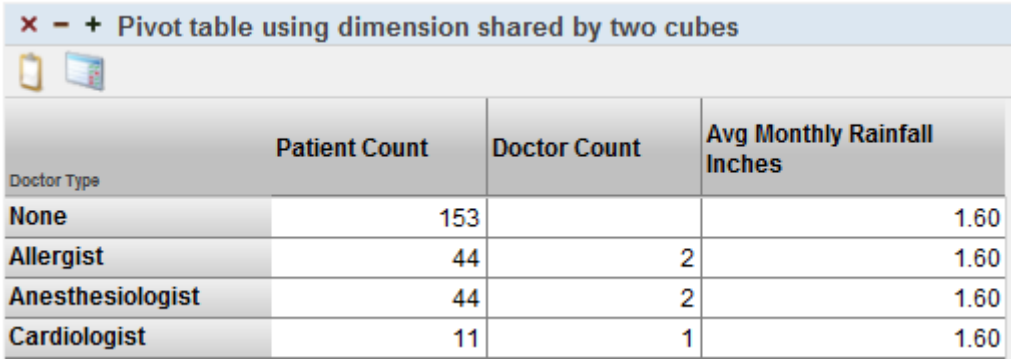
同じダッシュボードに、BirthD を行に使用するピボット・テーブルも含まれています。

Pivot table using dimension shared by two cubes

Decade	Patient Count	Doctor Count	Avg Monthly Rainfall Inches
1900s		40	1.61
1910s	7	40	1.61
1920s	19	40	1.63
1930s	60	40	1.63

CompoundCube/Doctors では BirthD ディメンジョンは定義されていないので、メジャー Doctor Count を、出生年代ごとに取得することはできません。Doctor Count 列の数値がすべてのセルで同じであることに注意してください。これは、すべての患者の 出生年代にわたる医師の合計数です。

最後に、Demo Compound Cube ダッシュボードには、DoctTypeD を行に使用するピボット・テーブルも含まれています。



Doctor Type	Patient Count	Doctor Count	Avg Monthly Rainfall Inches
None	153		1.60
Allergist	44	2	1.60
Anesthesiologist	44	2	1.60
Cardiologist	11	1	1.60

CompoundCube/CityRainfall では DocTypeD ディメンジョンは定義されていないので、メジャー Avg Monthly Rainfall Inches を、医師のタイプごとに取得することはできません。このメジャーは、すべての患者にわたって集約されます (メジャーでの定義に従い、平均によって)。

3

キューブ間のリレーションシップの定義

ここでは、Business Intelligence で使用するために、キューブ間のリレーションシップを定義する方法を説明します。

背景情報は、"モデル・オプションの概要" を参照してください。

キューブ・バージョンとリレーションシップの相互作用の詳細は、"キューブのバージョンとリレーションシップ" を参照してください。

"BI サンプルのアクセス方法" も参照してください。

3.1 リレーションシップの概要

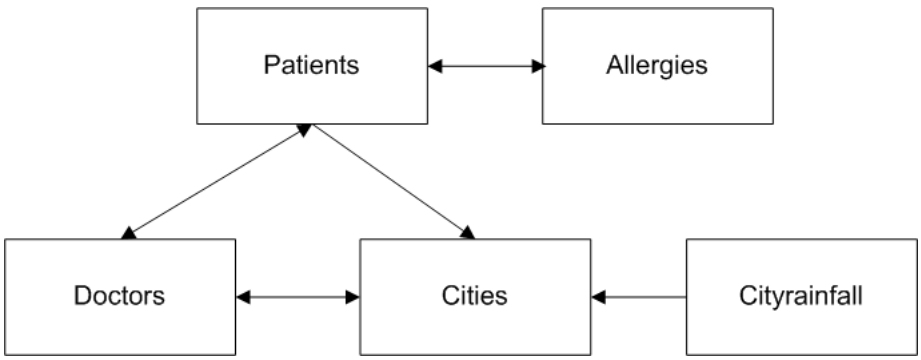
キューブ間のリレーションシップは、以下のように定義できます。

- 一対多のリレーションシップ。両方のキューブで、いずれかのキューブのレベルを使用できます。

  一方のキューブ (片側) では、このリレーションシップはリスト・ベース・レベルとよく似た動作を行います。
- 一方向、一対多のリレーションシップ。一方のキューブで、このリレーションシップはリスト・ベース・レベルとよく似た動作を行います。他方のキューブでは、このリレーションシップは表示できません。

リレーションシップを定義する際は、レベルを複数回定義するのではなく、1 回だけ定義することで、ファクト・テーブルとインデックスのサイズを最小にできます。

Patients サンプルでは、`RelatedCubes` フォルダ内に 5 つの関連キューブが提供されています。下図は、これらのキューブの関連状況をまとめたものです。

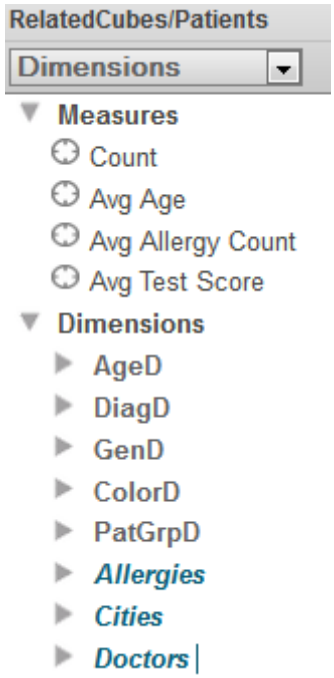


単方向の矢印は単方向のリレーションシップを示します。矢印を出してポイントする側のキューブは、相手方のキューブのレベルを認識できます。双方向矢印は双方向リレーションシップを示します。

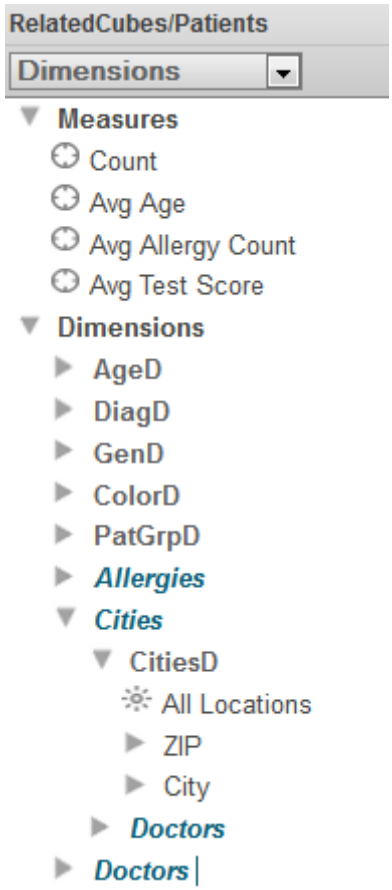
3.1.1 単方向リレーションシップの確認

ここでは、単方向リレーションシップの機能を調べます。

`RelatedCubes/Patients` キューブには、`RCities` キューブへの単方向リレーションシップがあります。`RelatedCubes/Patients` キューブをアナライザで開くと、以下のキューブ・コンテンツが表示されます。



`Cities` フォルダは、`RelatedCubes/Cities` キューブへのリレーションシップです。展開すると、そのキューブで定義されたレベルが表示されます。

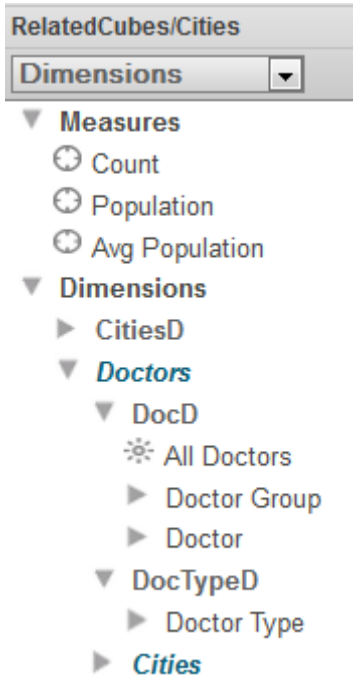


RelatedCubes/Patients キューブを操作する際は、これらすべてのレベルを、このキューブで直接定義されているレベルを使用するのと同じように使用できます。例えば、[ZIP] を [行] にドラッグ・アンド・ドロップできます。

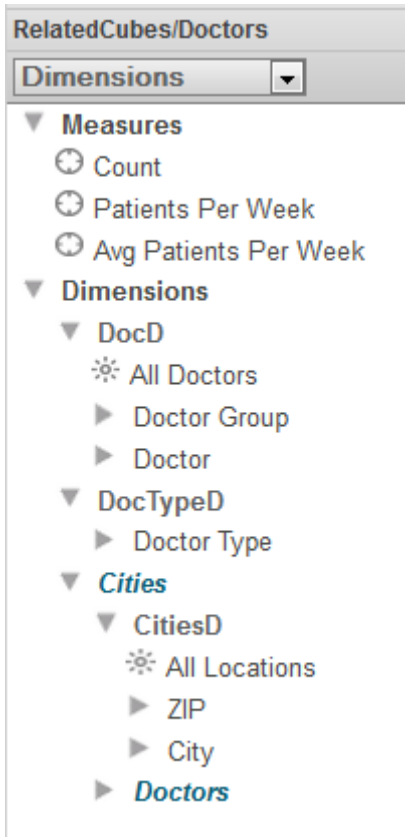
Cities 内の Doctors フォルダは、RelatedCubes/Cities キューブから RelatedCubes/Doctors キューブへのリレーションシップです。結果の混乱を招きやすいため、リレーションシップを重ねて使用することは推奨されません。

3.1.2 双方向リレーションシップの確認

RelatedCubes/Cities キューブと RelatedCubes/Doctors キューブの間には双方向リレーションシップがあります。RelatedCubes/Cities キューブを開くと、以下のキューブ・コンテンツが表示されます。



同様に、`RelatedCubes/Doctors` キューブを開くと、以下のキューブ・コンテンツが表示されます。



3.2 キューブ間のリレーションシップを定義する際の考慮事項

キューブ間の新たなリレーションシップを定義する前に、影響を受けるいずれかまたは両方のキューブが既にキューブ・マネージャに登録されているグループのメンバであるかどうかを考慮する必要があります。

登録したグループ内のキューブにリレーションシップを追加しようとすると、エラーが発生し、そのエラーが解決されるまでキューブ・マネージャのタスクが一時停止されることがあります。この動作を防ぐために、リレーションシップを追加する前に、影響を受けるキューブが属するグループをキューブ・マネージャから登録解除する必要があります。リレーションシップを追加したら、グループを再登録し、元の設定を再作成します。

リレーションシップへのこのような変更はすべて、開発システムで行うことが推奨されます。これにより、キューブ・レジストリが適切に設定されているかが検証され、その後、キューブ定義とキューブ・レジストリ・クラスが一緒にプロダクション・システムにインポートされます。

3.3 単方向リレーションシップの定義

あるキューブから別のキューブへの単方向リレーションシップを定義するには、最初のキューブで単一のリレーションシップを定義します。そのためには、最初のキューブで以下の変更を行います。

1. **[要素を追加]** を選択します。

   ダイアログ・ボックスが表示されます。

2. **[新規項目名の入力]** に、リレーションシップ名を入力します。これにより、リレーションシップの論理名が決まります。これを、他方のキューブの論理名と同じ名前にすると便利です。

3. **[リレーションシップ]** を選択します。

4. [OK] を選択します。

5. 右側の詳細領域で、以下の値を指定します。

属性	目的
表示名	リレーションシップの表示名。これを、他方のキューブの表示名と同じ名前にすると便利です。
プロパティ	これらのいずれかを指定します。この値は、他方のキューブで使用されるベース・クラスのレコードの ID にする必要があります。
式	
カーディナリティ	one
関連キューブ	他方のキューブの論理名。
ヌル置換文字列	(オプション) リレーションシップのソース・データが NULL の場合にメンバ名として使用する文字列 (例えば None) を指定します。 リレーションシップの既定の NULL 置換はありません。

   [逆] は指定しないでください。

6. アーキテクトでキューブ定義を選択します。次に、右側の詳細領域の **[依存]** オプションを編集します。このオプションでは、このクラスをコンパイルする前に実行可能にする必要がある 1 つ以上のクラスを指定します。

   既定では、新しいキューブを作成すると、システムは DependsOn キーワードを、キューブのソース・クラスの名前と等しくなるように設定します。

このオプションを指定するには、コンマで区切られたクラスのリストを指定して、そのリスト内の各クラスの完全なパッケージ名とクラス名を指定します。このリストには、キューブのソース・クラスと、このキューブが依存するキューブ・クラスを含める必要があります。以下に例を示します。

```
[ DependsOn = (MyApp.CubeBaseClass, MyApp.OtherCubeClass)]
```

7. 必要に応じて、スタジオでキューブ・クラスを編集して、このリレーションシップの、同じキューブで定義されている別のリレーションシップへの依存関係を定義します。

   場合によっては、2 つのリレーションシップの間に仮想依存関係が存在します。例えば、Country リレーションシップと Product リレーションシップを持つキューブがあるとします。これらのリレーションシップは、論理的に相互に独立しています。理論上は、任意の製品を任意の国で販売できます。しかし、特定の製品の販売先が特定の国々に限定される場合は、これらのリレーションシップ間に仮想依存関係が存在します。ユーザが国を選択したとき、望ましいのは、その国で販売可能な製品のみが表示されることです。

   そのような場合、リレーションシップ間に依存関係を追加できます。そのためには、"異なる階層に含まれるレベル間での依存関係の定義" の説明に従って、**[依存]** オプションを指定します。値には、同じキューブで定義されている別のリレーションシップの論理名を指定します。(または、リレーションシップがレベルに依存する場合は、そのレベルの MDX 識別子を指定します。)

   [依存] 属性は、DependsOn コンパイラ・キーワードとはまったく関係ないことに注意してください。

3.4 双方向リレーションシップの定義

キューブ間の双方向リレーションシップを定義するには、各キューブに 1 つずつ、計 2 つの補完的な `<relationship>` 要素を定義します。これらのキューブの一方が依存キューブで、もう一方が独立キューブです。

キューブ A (ソース・クラス A のレコードに基づく) とキューブ B (ソース・クラス B のレコードに基づく) について考えてみましょう。これらのキューブ間の双方向リレーションシップを定義するには、以下の手順を使用します。

1. ソース・クラス間のリレーションシップを評価します。これを使用し、以下のようにして、どちらのキューブが依存キューブなのかを判断します。
   - クラス A 内の 1 つのレコードがクラス B の複数のレコードに対応しているかどうか。

     この場合、キューブ B が依存キューブです。
   - クラス B 内の 1 つのレコードがクラス A の複数のレコードに対応しているかどうか。

     この場合、キューブ A が依存キューブです。
   - クラス間に1 対 1 のリレーションシップが存在するかどうか。

     この場合、どちらのキューブも依存キューブである可能性があります。

2. アーキテクトで、依存キューブに以下の変更を加えます。
   a. **[要素を追加]** を選択します。

      ダイアログ・ボックスが表示されます。
   b. **[新規項目名の入力]** に、リレーションシップ名を入力します。これにより、リレーションシップの論理名が決まります。これを、他方のキューブの論理名と同じ名前にすると便利です (例 : `IndependentCubeName`)。
   c. **[リレーションシップ]** を選択します。
   d. **[OK]** を選択します。
   e. 右側の詳細領域で、以下の値を指定します。

属性	目的	例
表示名	リレーションシップの表示名。これを、他方のキューブの表示名と同じ名前にすると便利です。	
プロパティ 式	これらのいずれかを指定します。この値は、他方のキューブで使用されるベース・クラスのレコードの ID にする必要があります。	
カーディナリティ	one	
逆	他方のキューブの逆リレーションシップの値。リレーションシップの名前を、ポイントするキューブの論理名と同じにすると便利であるため、このキューブの論理名を使用します。	DependentCubeName
関連キューブ	他方のキューブの論理名。	IndependentCubeName
ヌル置換文字列	(オプション) リレーションシップのソース・データが NULL の場合にメンバ名として使用する文字列を指定します。 リレーションシップの既定の NULL 置換はありません。	

3. アーキテクトでキューブ定義を選択します。つまり、中央の領域の最上行を選択します。次に、右側の詳細領域の [依存] オプションを編集します。このオプションでは、このクラスをコンパイルする前に実行可能にする必要がある 1 つ以上のクラスを指定します。

   既定では、新しいキューブを作成すると、システムは DependsOn キーワードを、キューブのソース・クラスの名前と等しくなるように設定します。

   このオプションを指定するには、コンマで区切られたクラスのリストを指定して、そのリスト内の各クラスの完全なパッケージ名とクラス名を指定します。このリストには、キューブのソース・クラスと、このキューブが依存するキューブ・クラスを含める必要があります。以下に例を示します。

   ```
   [ DependsOn = (MyApp.CubeBaseClass, MyApp.OtherCubeClass)]
   ```

4. アーキテクトで、独立キューブに以下の変更を加えます。

   a. [要素を追加] を選択します。

      ダイアログ・ボックスが表示されます。

   b. [新規項目名の入力] に、リレーションシップ名を入力します。これにより、リレーションシップの論理名が決まります。このときに、他方のキューブの論理名と同じ名前にすると便利です (例 : DependentCubeName)。

   c. [リレーションシップ] を選択します。

   d. [OK] を選択します。

   e. 右側の詳細領域で、以下の値を指定します。

属性	目的	例
表示名	リレーションシップの表示名。これを、他方のキューブの表示名と同じ名前にすると便利です。	
カーディナリティ	many	

属性	目的	例
逆	他方のキューブの逆リレーションシップの値。リレーションシップの名前を、ポイントするキューブの論理名と同じにすると便利であるため、このキューブの論理名を使用します。	`IndependentCubeName`
関連キューブ	他方のキューブの論理名。	`DependentCubeName`

f. 必要に応じて、スタジオでキューブ・クラスを編集して、このリレーションシップの、同じキューブで定義されている別のリレーションシップへの依存関係を定義します。

場合によっては、2 つのリレーションシップの間に仮想依存関係が存在します。例えば、Country リレーションシップと Product リレーションシップを持つキューブがあるとします。これらのリレーションシップは、論理的に相互に独立しています。理論上は、任意の製品を任意の国で販売できます。しかし、特定の製品の販売先が特定の国々に限定される場合は、これらのリレーションシップ間に仮想依存関係が存在します。ユーザが国を選択したとき、望ましいのは、その国で販売可能な製品のみが表示されることです。

そのような場合、リレーションシップ間に依存関係を追加できます。そのためには、"ディメンジョン、階層およびレベルの定義" の説明に従って、**[依存]** オプションを指定します。値には、同じキューブで定義されている別のリレーションシップの論理名を指定します。(または、リレーションシップがレベルに依存する場合は、そのレベルの MDX 識別子を指定します。)

[依存] オプションは、DependsOn コンパイラ・キーワードとはまったく関係ないことに注意してください。

3.5 リレーションシップを持つキューブの構築

リレーションシップを持つキューブを構築する際は、独立キューブを最初に構築します。これはリレーションシップのソース・プロパティまたはソース式を定義しないほうのキューブです。より一般的には、独立したキューブを再構築したときには、必ずその次に、依存するキューブを再構築する必要があるということです。推奨される最善の手段は、適切な順序でキューブを構築するユーティリティ・メソッドまたはルーチンを作成することです。

プロダクション・システムでは、キューブ・マネージャを使用し、指定に従ってキューブを構築または同期する自動化タスクを作成することをお勧めします。詳細は、"キューブ・マネージャを使用する方法" を参照してください。

3.5.1 関連キューブの構築順序の決定

キューブ・マネージャを使用していない場合は、正しい構築順序を決定する必要があります。そのためには、前述のルールに従うか、%DeepSee.CubeManager.Utils の GetCubeGroups() メソッドを使用します。1 つ目の引数は、参照によって返され、指定されたネームスペース内のキューブの自然なグループ分けを示す配列です。次に例を示します。

```
SAMPLES>d ##class(%DeepSee.CubeManager.Utils).GetCubeGroups(.groups)

SAMPLES>zw groups
groups=11
groups(1,"CITIES")=1
groups(2,"CITYRAINFALL")=1
groups(3,"COMPOUNDCUBE/CITYRAINFALL")=1
groups(4,"COMPOUNDCUBE/DOCTORS")=1
groups(5,"COMPOUNDCUBE/PATIENTS")=1
groups(6,"CONNECTORCUBE")=1
groups(7,"HOLEFOODS")=1
groups(8,"HOLEFOODSBUDGET")=1
groups(9,"PATIENTS")=1
groups(10,"PATIENTSQUERYCUBE")=1
groups(11,"RELATEDCUBES/ALLERGIES")=5
groups(11,"RELATEDCUBES/CITIES")=1
groups(11,"RELATEDCUBES/CITYRAINFALL")=2
groups(11,"RELATEDCUBES/DOCTORS")=3
groups(11,"RELATEDCUBES/PATIENTS")=4
```

この配列の説明は以下のとおりです。

- 1 つ目の添字は、このネームスペース内のキューブの一意のグループを識別します。1 つ目のグループには 1、2 つ目のグループには 2 というように番号が付けられています。これらの番号は任意です。
- 2 つ目の添字は、このネームスペース内のキューブを識別します。このキューブは、1 つ目の添字で識別されたグループに含まれています。
- このノードの値は、このグループ内の指定されたキューブの構築順序です。

この例では、1 つのグループが RELATEDCUBES/ALLERGIES、RELATEDCUBES/CITIES、RELATEDCUBES/CITYTRAINFALL、RELATEDCUBES/DOCTORS、および RELATEDCUBES/PATIENTS というキューブで構成されています。このグループでは、RELATEDCUBES/CITIES を最初に構築し、その後、RELATEDCUBES/CITYTRAINFALL、RELATEDCUBES/DOCTORS、RELATEDCUBES/PATIENTS と続き、最後に RELATEDCUBES/ALLERGIES を構築する必要があります。

3.5.2 関連キューブを間違った順序で構築した場合のエラー

関連キューブを間違った順序で構築した場合、%BuildCube() は以下のようなエラーを生成します。

```
ERROR #5001: Missing relationship reference in RelatedCubes/Patients: source ID 1 missing reference to
 RxHomeCity 4
```

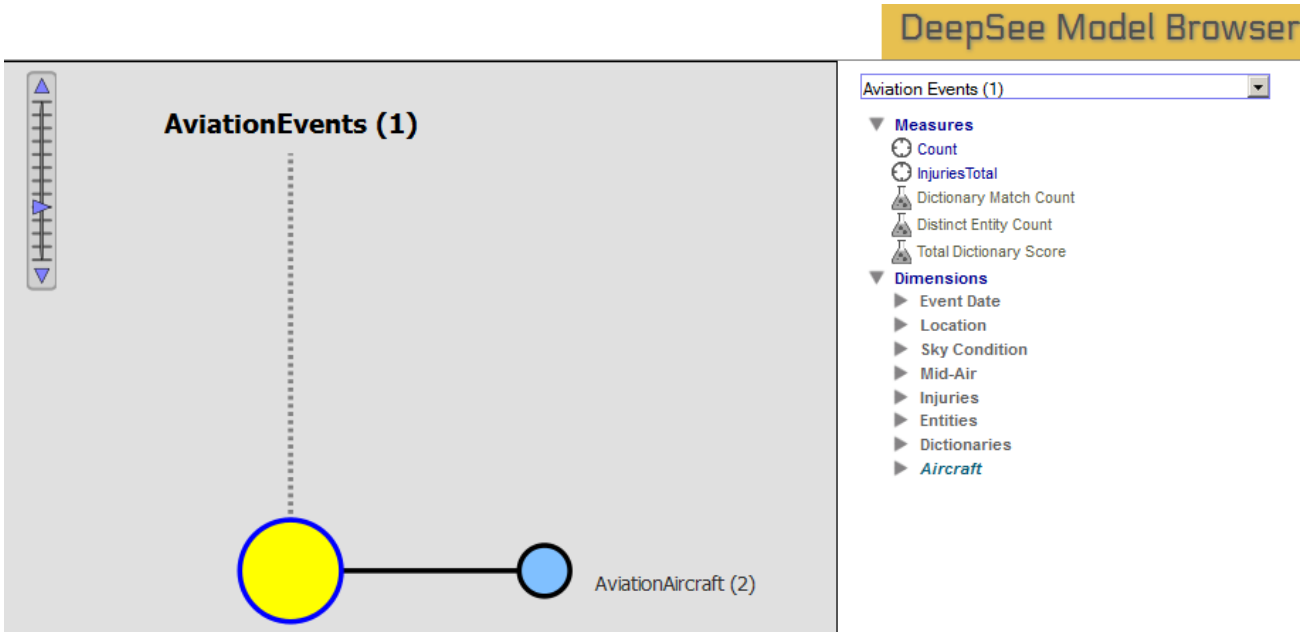
構築エラーおよびそれらを確認できる場所に関する一般情報は、″構築エラー″ を参照してください。

3.6 モデル・ブラウザの使用

システムには、キューブ定義を表示するために使用できる補助ツール (モデル・ブラウザ) が用意されています。モデル・ブラウザは、キューブ間のリレーションシップの表示および移動が可能であるため、リレーションシップを持っているキューブに対して特に役に立ちます。

このツールにアクセスするには、[Analytics]、[**ツール**]、[**モデル・ブラウザ**] の順に選択します。

モデル・ブラウザにアクセスしたら、右側のドロップダウン・リストを使用して、キューブの名前を選択します。モデル・ブラウザに次のような画面が表示されます。



中央の黄色の円は、選択したキューブを表します。図の上部のラベルは、このキューブの名前 (この例では `AviationEvents`) を示しており、続いてかっこ内に関連キューブの数 (1) が示されています。中央の円の周囲にある円それぞれは、選択したキューブに関連する 1 つのキューブを表します。これらの円に対して、ラベルがキューブ名および関連キューブの数を示します。例えば、`AviationAircraft` というラベルが付いている円は、AviationAircraft キューブを表します。Referrals に関連するキューブは 2 つあります。

任意の円を選択すると、新しく選択したキューブが中央に表示されるように図が変化し、残りの図も適宜更新されます。

ページの右側には、選択したキューブの詳細が表示されます。この領域には、アナライザの左側の領域での表示方法とまったく同じ方法でキューブのコンテンツが表示されます。

3.7 リレーションシップの削除

後でリレーションシップを削除する場合は、以下の手順を実行します。

1. そのリレーションシップを両方のキューブ定義から削除します。
2. 両方のキューブ定義をリコンパイルします。
3. すべての変更済みキューブを正しい順序で再構築します。

4

キューブでの Text Analytics の使用法

重要　InterSystems IRIS Business Intelligence 内での Text Analytics の統合では、InterSystems IRIS 自然言語処理 (NLP) が使用されていますが、インターシステムズではこれは非推奨となっています。このため、これはインターシステムズ製品の今後のバージョンから削除される可能性があります。以下のドキュメントは、リファレンスとして既存ユーザのみに提供されます。代わりのソリューションを見つけるサポートを希望する既存ユーザは、インターシステムズのサポート窓口 にお問い合わせください。

ここでは、Business Intelligence キューブ内に Text Analytics (テキスト・データを調査する分析) を含める方法について説明します (ここでは、テキスト・データを構造化されていないデータと呼びます)。

"Text Analytics で使用するセカンダリ・キューブの生成" も参照してください。

"BI サンプルのアクセス方法" も参照してください。

4.1 Text Analytics との統合の概要

Analytics エンジンには、テキスト・データ (構造化されていないデータ) を分析する機能が組み込まれています。テキスト・データとは、英語やフランス語などの人間の言語で記述されたテキストのことです。この機能の概要は、"コンセプトの概要 (NLP)" を参照してください。

構造化されていないデータを含むプロパティがキューブのソース・テーブルに存在する場合 (例えば、テキストを含む文字列フィールド)、キューブ内で構造化されていないデータを使用できます。その際、NLP (自然言語処理) ディメンジョンを使用するピボット・テーブルを定義することができ、さらにそれらのピボット・テーブルは通常どおりダッシュボードで使用できます。

例えば、キューブのソース・テーブルに構造化されているデータと構造化されていないデータの両方が含まれているとします。このページで後述する Aviation デモはその例です。このデモの場合、ソース・テーブルは、航空イベントのレコードで構成されます。航空イベントごとに、出来事の発生場所、航空機型式などを示す一連の短い文字列フィールドが存在します。長いテキスト・フィールドには、イベントの詳細レポートが格納されます。

(InterSystems IRIS Business Intelligence KPI メカニズムによって、Text Analytics クエリを公開する KPI も定義できます。"KPI とダッシュボード" を参照してください。)

4.1.1 用語

構造化されていないデータの分析時に、Analytics エンジンは、そのデータ内のエンティティを識別して、同類の単語および文の中での役割を特定します。エンティティとは、テキストの最小限の論理ユニット、つまり単語または単語のグループです。エンティティの例は、`clear skies` や `clear sky` です (これらは別のエンティティであることに注意してください。語幹解析は実行されません)。言語モデルでは、以下の 2 種類のエンティティが識別されます。

- 関係とは、2 つの概念の関係を指定することでそれらの概念を結合する単語や単語のグループです。関係は、通常は動詞ですが、必ずしもそうとは限りません。
- 概念とは、関係によって関連付けられる単語や単語のグループです。概念は、通常は名詞または名詞句ですが、必ずしもそうとは限りません。

エンジンは、各文を概念およびリレーションシップの論理的な順序に変換します。こうすることによって、同類の単語 (エンティティ) および文内でのその役割を識別します。これは、はるかに構造化された形式のデータで、分析の基礎となります。これには、関連メトリック、自動化された要約およびカテゴリ化が含まれ、既存の分類法またはオントロジと照合します。

概念およびリレーションシップの識別 (ならびにこれらのエンティティの関連性の測定) の主要なアクティビティに関して、エンジンは、テキストで説明されているトピックの情報を必要としません。つまり、トピックの理解ではなく言語の理解に基づいているため、このエンジンはドメインまたはトピックで適切に機能する真のボトムアップ・テクノロジといえます。

ただし、テキストで説明されるトピックに関する知識がある場合、これら既知の用語 (ディクショナリ用語と呼ばれる) との一致をエンジンに検出させることができます。エンジンは、エンティティの役割およびエクステントを理解するため、テキストで発見されたエンティティ (またはエンティティ・シーケンス) の既存の用語との一致率の高低を判定し、一致スコアを算出することも可能です。例えば、ディクショナリ用語 `runway` に対して、エンティティ `runway` (テキスト内) は完全一致ですが、`runway asphalt` は部分一致であるためスコアが低くなります。

ディクショナリ用語は、ディクショナリ項目にグループ化されます。これは、識別する一意のものを表します。例えば、`runway` および `landing strip` は、滑走路を意味する汎用ディクショナリ項目の 2 つの用語です。

ディクショナリの概要は、"スマート・マッチング : ディクショナリの作成" および "スマート・マッチング : ディクショナリの使用" を参照してください。

4.1.2 NLP (自然言語処理) のメジャーとディメンジョンについて

NLP メジャーは、その他の種類のメジャーと異なり、アナライザには表示されません。また、ピボット・テーブルで直接使用することはできません。NLP メジャーは、テキスト・データとしてエンジンで処理する必要があるプロパティごとに定義します。そのメジャーは、NLP ディメンジョンの基礎として使用できます。

NLP ディメンジョンは、その他のディメンジョンと似ています。1 つ以上のレベルを含み、それらはメンバを含みます。メンバは、キューブのソース・クラスの一連のレコードで構成されます。

NLP ディメンジョンには、以下の 2 種類があります。

- エンティティ・ディメンジョン。エンティティ・ディメンジョンには、1 つのレベルが含まれます。そのレベルの各メンバは、構造化されていないデータでエンジンが検出したエンティティに対応します。

  このレベルのメンバは、分散 (このエンティティを含むレコードの数) の降順に並べ替えられます。アナライザの左の領域でこのレベルを展開すると、最も一般的なエンティティ 100 件が表示されます。ただし、このレベルをフィルタとして使用すると、検索によってどのエンティティにもアクセスできます。
- ディクショナリ・ディメンジョン。ディクショナリ・ディメンジョンには、通常は以下のように 2 つのレベルが含まれます。
  - 上位レベルのディクショナリ・レベルには、ディクショナリごとに 1 つのメンバが含まれます。このメンバは、指定されたディクショナリの項目と一致するすべてのレコードで構成されます。

    例えば、　　メンバは、ディクショナリ　　の項目と一致するすべてのレコードで構成されます。これには、 、 、　 などの項目が含まれます。
  - オプションの下位レベルである項目レベルには、ディクショナリ項目ごとに 1 つのメンバが含まれます。このメンバは、指定されたディクショナリ項目の用語と一致するすべてのレコードで構成されます。

    例えば、 メンバは、 、 、　 など、冬メンバの用語のいずれかと一致する各レコードで構成されます。

通常はソース・テキストに複数のエンティティが含まれるので、ソース・レコードは、あるレベルの複数のメンバに属する可能性が高くなります。

4.1.3 生成された Text Analytics ドメイン

このページで説明している機能を使用すると、1 つまたは複数の Text Analytics ドメインが作成されます。各キューブ・レベルおよび各メジャーは、Text Analytics クエリで使用できる擬似メタデータ・フィールドとして使用できます。レベルでは、等値演算子および不等値演算子がサポートされます。メジャーでは、すべての演算子がサポートされます。Text Analytics クエリの詳細は、"InterSystems IRIS 自然言語処理 (NLP) の使用法" を参照してください。

このページで後述する "Text Analytics ドメインの管理" も参照してください。

4.2 Aviation Events デモの設定

ここでは、Samples-Aviation サンプル (https://github.com/intersystems/Samples-Aviation) を使用します。サンプルは、SAMPLES という名前の専用のネームスペースを作成して、そのネームスペースにロードすることをお勧めします。一般的な手順は、"サンプルのダウンロード" を参照してください。

Aviation デモには、いくつかのキューブ定義、条件リストの例、およびダッシュボードが含まれます。

このデモでは、プライマリ・キューブ (Aviation Events) はテーブル Aviation.Event に基づいていて、これは航空イベントのレコードで構成されています。航空イベントごとに、出来事の発生場所、航空機型式などを示す一連の短い文字列フィールドが存在します。長いテキスト・フィールドには、イベントの詳細レポートが格納されます。

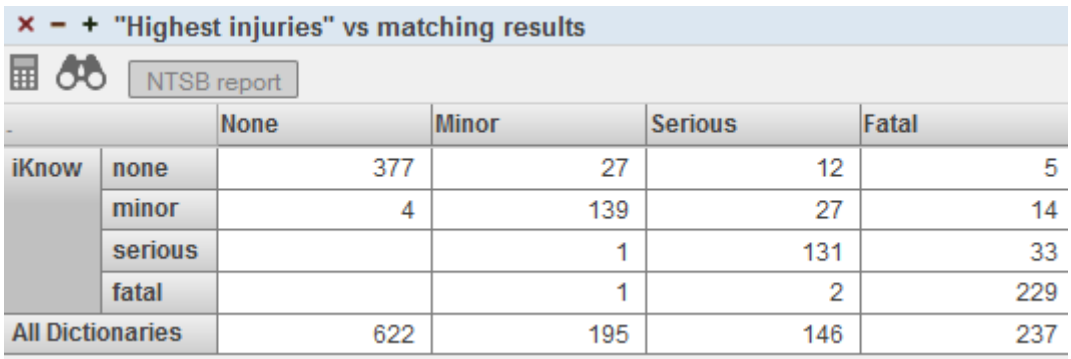
4.2.1 サンプル・ダッシュボード

サンプル・ダッシュボードを表示する手順は以下のとおりです。

1. 管理ポータルで、このサンプルをロードしたネームスペースにアクセスします。
2. [Analytics]→[**ユーザポータル**] を選択します。

   ユーザ・ポータルが表示され、既存のパブリック・ダッシュボードとピボット・テーブルがこのネームスペースにリストされます。ユーザ・ポータルの上部で、[**表示**:] 表示オプションを選択できます。
3. [Aviation event reports] ダッシュボードを選択します。

[Aviation event reports] ダッシュボードには、以下のピボット・テーブルが含まれます。



		None	Minor	Serious	Fatal
iKnow	none	377	27	12	5
	minor	4	139	27	14
	serious		1	131	33
	fatal		1	2	229
All Dictionaries		622	195	146	237

このピボット・テーブルは、以下のように定義されています。

- メジャーは、Count (イベント数) です。

- 最後の行を除いて、Analytics エンジンが検出し Text Analytics ディクショナリの項目に一致したエンティティを各行は表します。
- 最後の行は、ディクショナリの項目に一致したすべてのレコードを表します。
- 列には、`Highest Injury` レベル (レポートに対して用意されている直接分類に基づく標準データ・ディメンジョン内のレベル) のメンバが表示されます。

つまり、行には、(テキストのレポートで Analytics エンジンが検出した) 構造化されていないデータによって決定されたグループが表示され、列には、構造化されているデータによって決定されたグループ (直接分類) が表示されます。このようなピボット・テーブルを使用して、構造化されていないデータと構造化されているデータの間の不一致を検出できます。

例えば、最も高い負傷レベルとして正式に `None` に分類されているレポートの情報を提供する `None` 列について考えてみます。この設定で、この列のセルでは、次の情報が提供されます。

- `iKnow –> none` セルは、377 個のレポートで、Analytics エンジンが負傷ディクショナリの `none` 項目と一致するエンティティを検出したことを示しています。これは妥当です。
- `iKnow –> minor` セルは、4 レポートで、Analytics エンジンが負傷ディクショナリの `minor` 項目と一致するエンティティを検出したことを示しています。つまり、4 レポートで、構造化されていないデータは、(これらのレポートは最も高い負傷レベルとして `None` が分類されているにもかかわらず) 軽傷があったことを示唆しています。

  このセルの値は、構造化されていないデータと構造化されているデータの間に不一致があることを表しています。このセルでは、さらに調査を進め、完全なレポートを読むことが有用です。
- `iKnow > serious` セルおよび `iKnow > fatal` セルは空です。これらのセルは、Analytics エンジンが `serious` または `fatal` という負傷エンティティを検出したレコードがないことを示しています。これは妥当です。

列 `None` のセル `iKnow > minor` の詳細リストを表示し、これらのインシデントのレポートを ( アイコンを使用して) 表示する場合、これらのレポートの分類が間違っていて、すべてのレポートに軽傷があることがわかります。例えば、最初のレポートには、"The private pilot reported minor injuries" (プライベート・パイロットが軽傷を報告しました) という文が含まれています。

同様に、`Minor` 列で、`iKnow > serious` セルと `iKnow > fatal` セルが他の不一致を示しています。`Serious` 列では、`iKnow > fatal` セルが他の不一致を示しています。

4.2.2 Aviation キューブの詳細

これらのデモ・キューブの詳細を確認するには、アーキテクトおよびアナライザを使用します。Aviation Events キューブには、以下の要素が含まれています。

- `Count` メジャー。イベント・レポート数です。
- `InjuriesTotal` メジャー。負傷の合計です。
- `Report` メジャー。構造化されていないデータを使用する NLP メジャーです。このメジャーは、NLP ディメンジョンでのみ使用することを意図されているため、アナライザにはリストされません。
- `Event Date`、`Location`、`Sky Condition`、`Mid-Air`、および `Injuries` の各ディメンジョン。構造化されているデータを使用する標準ディメンジョンです。
- `Entities` ディメンジョン。エンティティ・ディメンジョンです。
- `Dictionaries` ディメンジョン。ディクショナリ・ディメンジョンです。
- `Aircraft` ディメンジョン。Aircraft キューブとのリレーションシップです。

Aircraft キューブは、航空機の属性（タイプ、モデルなど）でレコードをグループ化するために使用できるディメンションを提供します。Aircraft キューブには、航空機の職員と関連付けられたレベルを提供する、Crew キューブとのリレーションシップも含まれます。

4.3 NLP メジャーの定義

NLP メジャーを追加する手順は以下のとおりです。

1. **[要素を追加]** を選択します。

   ダイアログ・ボックスが表示されます。

2. **[新規項目名の入力]** に、メジャー名を入力します。
3. **[メジャー]** を選択します。
4. [OK] を選択します。
5. アーキテクトの中央領域でメジャーを選択します。
6. 以下のオプションを指定します。

   - **[プロパティ]** または **[式]** ― 構造化されていないデータを含むソース値を指定します。

     または、プレーン・テキスト・ファイル（処理するテキストが格納されているファイル）のフル・パスを含む値を指定します。

   - **[タイプ]** ― [iKnow] を選択します。
   - **[iKnow ソース]** ― ソース値のタイプを指定します。`string`、`stream`、または `file` を選択します。例えば、選択したソース・プロパティのタイプが %Stream.GlobalCharacter の場合は、`stream` を選択します。または、値がファイルへのパスの場合は、`file` を選択します。

     このオプションは、Analytics エンジンに対して、**[プロパティ]** または **[式]** に指定された値の処理方法を指示します。

     値 `domain` は、上級者向けです。"代替手法：既存の Text Analytics ドメインの使用" を参照してください。

   例として、Aviation キューブは `Aviation.Event` クラスに基づきます。NLP メジャーの `Report` は、そのクラスの `NarrativeFull` プロパティに基づきます。このメジャーでは、**[iKnow ソース]** は `string` です。

   [集計] オプションは、NLP メジャーに影響しません。

7. アーキテクトにキューブ定義を保存します。
8. このメジャーを使用するディクショナリ・レベルを 1 つまたは複数定義することを予定している場合、以下のように **[ディクショナリ]** オプションも指定します。

   a. **[ディクショナリ]** の下にあるボタンを選択します。

      ダイアログ・ボックスが表示されます。

   b. **[利用可能なディクショナリ]** リストで適切なディクショナリを選択してから [>] をクリックして、そのディクショナリを **[選択されたディクショナリ]** リストに移動します。

      [利用可能なディクショナリ] に必要なディクショナリがリストされていない場合は、このページで後述する "ディクショナリのロードおよび更新" を参照してください。

   c. 必要に応じて、この手順を繰り返します。

   d. [OK] を選択します。

各ディクショナリは実際には条件リストです。ここで説明した手順を実行すると、Analytics エンジンは指定した条件リストを自動的に検出して、ディクショナリとしてロードし、マッチングを実行します。(この属性を追加しなくても、代わりにこれらのタスクを実行するメソッドを起動できます。)

NLP メジャーは、キューブのファクト・テーブルには格納されず、アナライザに表示されないことに注意してください。NLP メジャーの主な目的は、Text Analytics ドメインを定義することと、NLP ディメンジョンの基礎として使用することです。次のセクションを参照してください。

メジャーの既定のパラメータをオーバーライドできることにも注意してください。このページで後述する "メジャーの Text Analytics ドメイン・パラメータの指定" を参照してください。このオプションは上級者向けです。

4.3.1 代替手法 : 既存の Text Analytics ドメインの使用

既存の Text Analytics ドメインは、再利用できます。前述の手順を以下のように変更して、実行します。

- [iKnow ソース] に `domain` と指定します。
- ソース式またはソース・プロパティを指定する際、これがファクト・テーブルのレコードに対応するソースの外部 ID として評価されるように指定します。
- スタジオで、`iKnowDomain` 属性をメジャー定義に追加します。その値は、既存の Text Analytics ドメインの名前である必要があります。
- 手順 8 をスキップします。つまり、[ディクショナリ] オプションを指定しないでください。

重要　この場合、Text Analytics ドメインは、キューブのコンパイル、構築、および同期の処理とは別に管理されます。Analytics エンジンは、構築時に Text Analytics レコードの削除またはロードを一切実行しません。カスタム・コードでは、ファクト・レベルで外部 ID プロパティ/式によって表現され、特定されるすべてのデータが正しくロードされることを保証する必要があります。Analytics エンジンは、実行時にのみ Text Analytics ロジックを使用します。また、Analytics エンジンは、キューブのコンパイル、構築、および同期の処理時にデータを自動的にロードすることはない点に注意してください。データのロード、パラメータの指定、またはそれ以外にこのドメインの管理を実行するには、"InterSystems IRIS 自然言語処理 (NLP) の使用法" の説明に従ってドメイン API を直接使用します。

4.3.2 代替手法 : 他の場所からの構造化されていないテキストの取得

一部のシナリオでは、構造化されていないテキストを Web ページから取得する必要が生じることがあります。例えば、追加情報 (ニュース記事など) を検索できる URL を含むフィールドのある、構造化された情報のテーブルがある場合があります。そのような場合、NLP メジャーとしてテキストを使用する最も簡単な方法は、以下のとおりです。

- ユーティリティ・メソッドを記述してテキストを URL から取得します。
- NLP メジャーのソース式でそのユーティリティ・メソッドを参照します。

例えば、ニュース記事に関する概要情報のあるクラスでキューブをベースにするとします。クラス内の各レコードには、報道会社の名前、日付、見出し、および `Link` という名前のプロパティ (ニュース全文の URL が格納されている) が格納されています。それらの URL にあるニュース本文を使用する NLP メジャーを作成します。

そのためには、以下のように `GetArticleText()` メソッドをキューブ・クラスで定義できます。

Class Member

```
ClassMethod GetArticleText(pLink As %String) As %String
{
    set tSC = $$$OK, tStringValue = ""
    try {
        set tRawText = ..GetRawTextFromLink(pLink, .tSC)
```

```
        quit:$$$ISERR(tSC)

        set tStringValue = ..StripHTML(tRawText, .tSC)
        quit:$$$ISERR(tSC)

    } catch (ex) {
        set tSC = ex.AsStatus()
    }
    if $$$ISERR(tSC) {
        set tLogFile = "UpdateNEWSARCHIVE"
        set tMsg = $system.Status.GetOneErrorText(tSC)
        do ##class(%DeepSee.Utils).%WriteToLog("UPDATE", tMsg, tLogFile)
    }
    quit tStringValue
}
```

GetRawTextFromLink() メソッドは、以下のように Raw テキストを取得します。

Class Member

```
ClassMethod GetRawTextFromLink(pLink As %String, Output pSC As %Status) As %String
{
    set pSC = $$$OK, tRawText = ""
    try {
        // derive server and path from pLink
        set pLink = $zstrip(pLink,"<>W")
        set pLink = $e(pLink,$find(pLink,"://"),*)
        set tFirstSlash = $find(pLink,"/")
        set tServer = $e(pLink,1,tFirstSlash-2)
        set tPath = $e(pLink,tFirstSlash-1,*)

        // send the HTTP request for the article
        set tRequest = ##class(%Net.HttpRequest).%New()
        set tRequest.Server = tServer
        set tSC = tRequest.Get(tPath)
        quit:$$$ISERR(tSC)

        set len = 32000
        while len>0 {
            set tString = tRequest.HttpResponse.Data.Read(.len, .pSC)
            quit:$$$ISERR(pSC)
            set tRawText = tRawText _ tString
        }

    } catch (ex) {
        set pSC = ex.AsStatus()
    }
    quit tRawText
}
```

StripHTML() メソッドは、以下のように HTML フォーマットを削除します。

Class Member

```
ClassMethod StripHTML(pRawText As %String, Output pSC As %Status) As %String
{
    set pSC = $$$OK, tCleanText = ""
    try {
        for tTag = "b","i","span","u","a","font","em","strong","img","label","small","sup","sub" {
            set tReplaceTag(tTag) = " "
        }

        set tLowerText = $$$LOWER(pRawText)
        set tStartPos = $find(tLowerText,"<body")-5, tEndTag = ""
        set pRawText = $e(pRawText,tStartPos,*), tLowerText = $e(tLowerText,tStartPos,*)
        for {
            set tPos = $find(tLowerText,"<")
            quit:'tPos // no tag start found

            set tNextSpace = $f(tLowerText," ",tPos), tNextEnd = $f(tLowerText,">",tPos)
          set tTag = $e(tLowerText,tPos,$s(tNextSpace&&(tNextSpace<tNextEnd):tNextSpace, 1:tNextEnd)-2)

            if (tTag="script") || (tTag="style") {
                set tPosEnd = $find(tLowerText,">",$find(tLowerText,"</"_tTag,tPos))
            } else {
                set tPosEnd = tNextEnd
            }
            if 'tPosEnd { //
                set tEndTag = $e(pRawText,tPos-1,*)
                set pRawText = $e(pRawText,1,tPos-2)
```

```
                quit
            }

            set tReplace = $s(tTag="":"", 1:$g(tReplaceTag(tTag),$c(13,10,13,10)))
            set pRawText = $e(pRawText,1,tPos-2) _ tReplace _ $e(pRawText,tPosEnd,*)
            set tLowerText = $e(tLowerText,1,tPos-2) _ tReplace _ $e(tLowerText,tPosEnd,*)
        }
        set tCleanText = $zstrip($zconvert(pRawText, "I", "HTML"),"<>=W")

    } catch (ex) {
        set pSC = ex.AsStatus()
    }
    quit tCleanText
}
```

最後に、NLP メジャーを作成し、それを `%cube.GetArticleText(%source.Link)` ソース式でベースにします。

4.4 ディクショナリのロードおよび更新

このセクションでは、NLP メジャーとディメンジョンで使用するために、ディクショナリをロードおよび更新する方法について説明します。

4.4.1 ディクショナリのロード

InterSystems IRIS にディクショナリをロードするには、以下の手順を実行します。

1. ″条件リスト・マネージャへのアクセス″ の説明に従って、条件リスト・マネージャにアクセスします。
2. ディクショナリ項目およびディクショナリ用語が含まれる新しい条件リストを定義します。この条件リストを以下に示します。
   - 用語リストの簡便な名前を使用します。ディクショナリ名は、条件リスト名に基づき、接頭語が追加されます。
   - 必要に応じて、カスタム・フィールドの URI と language を追加します。これらのフィールドを使用する方法の詳細は、以下の手順を参照してください。

     どの条件リストにも、key と value のフィールドがあるので、使用する条件リストにもそれらのフィールドがあります。

   条件リストの作成に関する一般情報は、″条件リストの定義″ を参照してください。
3. 用語を条件リストに追加します。条件リストの各項目について、以下のように値を指定します。
   - key (必須) は、テキストで検出される可能性がある一意の用語です。
   - value (必須) は、対応するディクショナリ項目です。
   - URI (オプション) は、ディクショナリ項目 (条件リストの value 列) の一意の識別子です。特定のディクショナリ項目を参照する必要がある場合、この識別子を MDX クエリのメンバ・キーとして使用できます。この識別子は、ディクショナリ名とディクショナリ項目の組み合わせごとに一意である必要があります。

     このフィールドを省略した場合、以下の形式の URI が生成されます。

     ```
     :dictionary_name:dictionary_item
     ```

     dictionary_name は定義または更新するディクショナリの名前、dictionary_item は value フィールドの値です。
   - language (オプション) は、すべて小文字の言語タグです (en や es など)。

   以下に例を示します (language フィールドは省略されています)。

Terms

key	value	URI
broken clouds	clouds	:weather:clouds
calm winds	mild wind	:weather:wind
clear of clouds	clear	:weather:visibility
clear skies	clear	:weather:visibility
clear sky	clear	:weather:visibility
cumuliform clouds	clouds	:weather:clouds
drizzle	rain	:weather:rain
extreme turbulence	heavy wind	:weather:heavy wind

4. 条件リストを保存します。
5. ディクショナリとしてこの条件リストを使用する NLP メジャーごとに [**ディクショナリ**] オプションを指定します。"NLP メジャーの定義" を参照してください。

[**ディクショナリ**] オプションは、この NLP メジャーのディクショナリとしてロードする条件リストを指定します。指定された条件リストは、キューブの構築時に自動的にロードされます。

4.4.2 ディクショナリの更新

ディクショナリとして使用する条件リストを作成または変更する場合、ディクショナリを更新する必要があります。この操作には、**%iKnow.DeepSee.CubeUtils** の UpdateDictionary() メソッドを使用します。

```
classmethod UpdateDictionary(pTermList As %String,
                             pCube As %String = "",
                             pMeasure As %String = "",
                             pClearFirst As %Boolean = 0) as %Status
```

以下は、この指定の説明です。

- pTermList は、用語リストの名前です。
- pCube は、キューブの名前です。この引数を省略した場合、このメソッドは、このネームスペースのすべてのキューブに対して起動されます。
- pMeasure は、NLP メジャーの名前です。この引数を省略した場合、このメソッドは、指定されたキューブ (または pCube によってはすべてのキューブ) のすべての NLP メジャーに対して呼び出されます。
- pClearFirst は、再ロードする前に既存のディクショナリを用語リストから削除するかどうかを制御します。条件リストへの追加のみを実行した場合は pClearFirst に 0 を、既存の用語を変更または削除した場合は 1 を使用します。

  pClearFirst が 0 の場合、このメソッドは大幅に高速実行できます。

キューブの構築時に、新しい条件リストを追加することによって、このキューブによって使用されるすべてのディクショナリが更新されます。削除した項目および名前を変更した項目は、影響を受けません。このページで後述する "Text Analytics の更新が実行されるタイミング" を参照してください。

4.5 エンティティ・ディメンジョンの定義

エンティティ・ディメンジョンを追加する手順は以下のとおりです。

1. このページの前半で示した説明に従って、このディメンジョンで使用する NLP メジャーを作成します。

   この操作は、ディメンジョンを定義した後でも実行できます。その場合は、後でディメンジョンを編集して、ディメンジョンがこのメジャーを参照するようにします。

2. **[要素を追加]** を選択します。

   ダイアログ・ボックスが表示されます。

3. **[新規項目名の入力]** に、ディメンジョン名を入力します。

4. **[iKnow ディメンジョン]** を選択し、**[OK]** を選択します。

5. アーキテクトの中央領域でディメンジョンを選択します。

6. 必要に応じて、ディメンジョンに以下の変更を加えます。

   - **[iKnow タイプ]** – **[エンティティ]** を選択します。
   - **[NLP メジャー]** – このディメンジョンで使用する NLP メジャーを選択します。

7. アーキテクトの中央領域でレベルを選択し、必要に応じて **[名前]** と **[表示名]** を変更します。

8. 必要に応じて、このレベルのメンバを手動で指定し、スタジオを使用してそのレベル内に `<member>` 要素を定義します。

   このレベルは、既定ではすべてのエンティティで構成され、分散の降順に並べ替えられています。`<member>` を使用して手動でメンバを指定する場合、それによってこのレベルのメンバとその順序が指定されます。エンティティ・ディメンジョンの場合、アナライザに表示されるメンバ数は 100 個に固定されています。

   "キューブおよびサブジェクト領域の高度な機能の使用" の "レベルのメンバの手動指定" を参照してください。

ディメンジョンまたはレベルのどちらでも、**[ソース値]** には何も指定する必要がないことに注意してください。NLP ディメンジョンの場合、ソース値は関連付けられている NLP メジャーによって指定されます。

4.6 ディクショナリ・ディメンジョンの定義

ディクショナリ・ディメンジョンを追加する手順は以下のとおりです。

1. InterSystems IRIS にディクショナリをロードします。このページで前述した "ディクショナリのロード" を参照してください。

   この操作は、ディメンジョンを定義した後でも実行できます。

2. このディメンジョンで使用する NLP メジャーを作成します。

   この操作は、ディメンジョンを定義した後でも実行できます。その場合は、後でディメンジョンを編集して、ディメンジョンがこのメジャーを参照するようにします。

3. **[要素を追加]** を選択します。

   ダイアログ・ボックスが表示されます。

4. **[新規項目名の入力]** に、ディメンジョン名を入力します。

5. **[iKnow ディメンジョン]** を選択し、**[OK]** を選択します。

6. アーキテクトの中央領域でディメンジョンを選択します。

7. 必要に応じて、以下のように変更します。

   - **[iKnow タイプ]** – **[ディクショナリ]** を選択します。

- [NLP メジャー] ― このディメンジョンで使用する NLP メジャーを選択します。

8. 必要に応じて、このディメンジョンの同じ階層に別のレベルを追加します。

   ディメンジョンにレベルが 1 つだけ存在する場合、そのレベルを使用してディクショナリ項目にアクセスします。ディメンジョンにレベルが 2 つ存在する場合、下位レベルを使用してディクショナリ項目に一致するエンティティにアクセスします。

9. アーキテクトの中央領域で各レベルを選択し、必要に応じて [名前] と [表示名] を変更します。

10. アーキテクトにキューブ定義を保存します。

11. スタジオでキューブ・クラスを開き、このディメンジョンの定義を検索します。例えば、ディメンジョンにレベルが 1 つ存在する場合、以下のように表示されます (この例では改行が追加されています)。

```
<dimension name="MyDictionaryDimension" disabled="false"
 ` hasAll="false" allCaption="MyDictionaryDimension" allDisplayName="MyDict"
  type="iKnow" iKnowType="dictionary"  nlpMeasure="Report"
  hidden="false" showHierarchies="default">
    <hierarchy name="H1" disabled="false">
      <level name="Dictionary" disabled="false" list="false" useDisplayValue="true">
      </level>
    </hierarchy>
</dimension>
```

   または、ディメンジョンにレベルが 2 つ存在する場合、以下のように表示されます。

```
<dimension name="MyDictionaryDimension" disabled="false"
  hasAll="false" allCaption="MyDictionaryDimension" allDisplayName="MyDict"
  type="iKnow" iKnowType="dictionary"  nlpMeasure="Report"
  hidden="false" showHierarchies="default">
    <hierarchy name="H1" disabled="false">
      <level name="Dictionary" disabled="false" list="false" useDisplayValue="true">
      </level>
      <level name="Items" disabled="false" list="false" useDisplayValue="true">
      </level>
    </hierarchy>
</dimension>
```

12. ディクショナリ・レベルで、必要に応じてこのレベルで使用するディクショナリを 1 つまたは複数指定します。レベルが 2 つ存在する場合、高いほうのレベルがディクショナリ・レベルになります。レベルが 1 つ存在する場合、そのレベルがディクショナリ・レベルになります。

   ディクショナリを指定しない場合、すべてのディクショナリが使用されます。

   使用するディクショナリごとに、<level> 要素と </level> の間に以下を追加します。

```
        <member name="dictionary name" />
```

   dictionary name はディクショナリの名前です。

   例えば、ディクショナリを 1 つ使用する場合、以下のようになります。

```
<dimension name="MyDictionaryDimension" disabled="false"
  hasAll="false" allCaption="MyDictionaryDimension" allDisplayName="MyDict"
  type="iKnow" iKnowType="dictionary"  nlpMeasure="Report"
  hidden="false" showHierarchies="default">
    <hierarchy name="H1" disabled="false">
      <level name="Dictionary" disabled="false" list="false" useDisplayValue="true">
         <member name="my dictionary" />
      </level>
      <level name="Items" disabled="false" list="false" useDisplayValue="true">
      </level>
    </hierarchy>
</dimension>
```

13. スタジオにキューブ定義を保存します。

ディメンジョンまたはレベルのどちらでも、[**ソース値**] には何も指定する必要がないことに注意してください。NLP ディメンジョンの場合、ソース値は関連付けられている NLP メジャーによって指定されます。

4.7 項目レベルへのメンバ・オーバーライドの追加

レベルが 2 つ存在するディクショナリ・ディメンジョン内では、既定で、ディクショナリ・レベルによって下位項目レベルのメンバが決定されます。この項目レベルに、親によって決定される定義をオーバーライドする <member> 要素を追加できます。

これは、例えば、ディクショナリのサブセットのみ表示する必要がある場合に便利です。

このようなオーバーライドを作成する場合、各 <member> 要素は、以下の形式で指定する必要があります。

```
<member name="itemURI" displayName="displayName" />
```

itemURI はディクショナリ項目の一意の URI、displayName はディクショナリ項目の表示名です。このページで前述した"ディクショナリのロード" を参照してください。

<member> 要素を、必要に応じて並べ替えてリストします。以下はその例です。

```
<level name="ReportDictInjuriesDimItem" displayName="Injuries" >
   <member name=":injuries:none" displayName="not injured" />
   <member name=":injuries:minor" displayName="minor injuries" />
   <member name=":injuries:serious" displayName="serious injuries" />
   <member name=":injuries:fatal" displayName="killed" />
</level>
```

これらのオーバーライドは以下のように機能します。

- 少なくとも 1 つの <member> 要素が、指定されたディクショナリ項目とマッチング可能である場合、このレベルには、これらの <member> 要素によってリストされるメンバのみが含まれます。
- ディクショナリ項目とマッチング可能な <member> 要素が存在しない場合、これらのオーバーライドはすべて無視されます。

4.8 プラグインを使用するメジャーの追加

プラグインは事実上クエリです。システムには、専用の Text Analytics クエリを実行するプラグインが用意されています。これらのプラグインを使用して、エンティティの出現および一致結果に関する情報を提供する計算メジャーを追加できます。以降のセクションで詳細を説明します。

4.8.1 エンティティ出現を数値化するメジャーの追加

エンティティ出現に関する情報 (合計数、レコードあたりの平均数など) を提供するメジャーを簡単に追加できます。例は、Aviation Events キューブの計算メジャー Distinct Entity Count を参照してください。

独自のメジャーを追加するには、"計算メジャーの定義" の手順に従います。[**式**] には、以下の式を使用します。

```
%KPI("%DeepSee.iKnow","Result",1,"aggregate","total","%CONTEXT")
```

この式は、あらゆるコンテキストにおいて、個別のエンティティの合計数を返します。

"total" の代わりに、以下のいずれかを使用できます。

- "sum" ー この場合、式は、指定したコンテキストにおいて、(個別のエンティティではなく) エンティティの合計数を返します。つまり、エンティティが複数回カウントされる場合があります。
- "average" ー この場合、式は、指定したコンテキストにおいて、レコードあたりのエンティティの平均数を返します。
- "max" ー この場合、式は、指定したコンテキストのあらゆるレコードのエンティティの最大数を返します。
- "min" ー この場合、式は、指定したコンテキストのあらゆるレコードのエンティティの最小数を返します。

この式は、%KPI MDX 関数とプラグイン・クラス %DeepSee.PlugIn.iKnow を使用します。この関数の詳細は、"InterSystems MDX リファレンス" を参照してください。このクラスの詳細は、クラス・リファレンスを参照してください。

重要　"%CONTEXT" を省略すると、あらゆる場合で、計算メジャーは、すべてのコンテキストを無視し、データ・セット全体の結果を返します。

4.8.2 一致結果を数値化するメジャーの追加

ディクショナリ・マッチング結果に関する情報 (合計数、レコードあたりの平均一致スコアなど) を提供するメジャーを簡単に追加できます。例は、Aviation Events キューブの計算メジャー Dictionary Match Count および Total Dictionary Score を参照してください。

独自のメジャーを追加するには、"計算メジャーの定義" の手順に従います。[式] には、以下の式のいずれかを使用します。

- 一致結果 (ディクショナリ項目と一致する結果) の数を取得するには、以下の式を使用します。

```
%KPI("%DeepSee.iKnowDictionary","MatchCount",1,"aggregate","sum","%CONTEXT")
```

  この式は、あらゆるコンテキストにおいて、一致する結果の合計数を返します。

  "sum" の代わりに、前のセクションでリストした代替の集約タイプを使用できます。

- 一致する結果のスコアを取得するには、以下の式を使用します。

```
%KPI("%DeepSee.iKnowDictionary","MatchScore",1,"aggregate","sum","%CONTEXT")
```

  この式は、あらゆるコンテキストにおいて、一致する結果の合計スコアを返します。

  "sum" の代わりに、前のセクションでリストした代替の集約タイプを使用できます。

これらの式は、%KPI MDX 関数とプラグイン・クラス %DeepSee.PlugIn.iKnowDictionary を使用します。この関数の詳細は、"InterSystems MDX リファレンス" を参照してください。このクラスの詳細は、クラス・リファレンスを参照してください。

重要　"%CONTEXT" を省略すると、あらゆる場合で、計算メジャーは、すべてのコンテキストを無視し、データ・セット全体の結果を返します。

4.9 リストに Text Analytics の結果を組み込む方法

以下のように、リストに Text Analytics の結果を組み込むことができます。

- リストにサマリ・フィールドを組み込むことができます。
- リストから構造化されていないテキスト全文にリンクを追加できます。
- コンテンツ分析プラグインで使用する専用リストを定義できます。

4.9.1 リストに Text Analytics サマリ・フィールドを組み込む方法

構造化されていないテキストのサマリをリストに追加できると便利です。このようなサマリを追加するには、リスト・フィールド定義内で $$$IKSUMMARY トークンを使用します。このトークンは、2 つの引数 (角括弧で囲みます) を取ります。

```
$$$IKSUMMARY[nlpMeasure,summarylength] As Report
```

nlpMeasure はサマリを作成する NLP メジャーの名前、summary_length はサマリに追加する文の数 (既定値は 5) です。キューブに NLP メジャーが 1 つしかない場合は、nlpMeasure を省略できます。

As 節は、列のタイトルを指定します。この場合、タイトルは Report です。

$$$IKSUMMARY トークンは、ソースと最も関連性のある文を返し、最大 32000 文字の文字列に連結します。

以下に例を示します。

```
<listing name="Default" disabled="false" listingType="table"
    fieldList="%ID,EventId,Year,AirportName,$$$IKSUMMARY[Report] As Report">
</listing>
```

内部で $$$IKSUMMARY は、%iKnow.Queries.SourceAPI の GetSummary() メソッドを使用します。

リスト・キューブと関連キューブの間に多対一のリレーションシップがある場合は、$$$IKSUMMARY トークンを使用して関連キューブ内の NLP メジャーを参照することもできます。その場合は、$$$IKSUMMARY の最初の引数として、nlpMeasure の代わりに relationshipname.nlpMeasure を使用します。例えば、Observations キューブに、Patients キューブを指す Patient という名前のリレーションシップがあるとします。また、Patients キューブには、History という名前の NLP メジャーがあるとします。Observations キューブ内で、$$$IKSUMMARY[Patient.History] を含むリストを定義できます。

同様に、リレーションシップのリレーションシップを参照することもできます。例 :
$$$IKSUMMARY[Relationship.Relationship.Measure]

4.9.2 リストへの構造化されていないテキスト全文へのリンクの追加

構造化されていないテキスト全文が表示されているページへのリンクをリストに追加することもできます。このようなリンクを追加するには、リスト・フィールド定義内で $$$IKLINK トークンを使用します。このトークンは、1 つの引数 (角括弧で囲みます) を取ります。

```
$$$IKLINK[nlpMeasure]
```

nlpMeasure は、表示する NLP メジャーの名前です。キューブに NLP メジャーが 1 つしかない場合は、nlpMeasure を省略できます。

リスト・キューブと関連キューブの間に多対一のリレーションシップがある場合は、$$$IKLINK トークンを使用して関連キューブ内の NLP メジャーを参照することもできます。その場合は、$$$IKLINK の最初の引数として、nlpMeasure の代わりに relationshipname.nlpMeasure を使用します。同様に、リレーションシップのリレーションシップを参照することもできます。例 : $$$IKLINK[Relationship.Relationship.Measure]

前のサブセクションの例を参照してください。

4.9.3 コンテンツ分析プラグインで使用する専用リストの作成

アナライザに用意されている高度な分析オプションの 1 つとして、コンテンツ分析プラグインがあります。このオプションは、詳細リストを使用して、最も一般的なレコード 5 件と最も一般的でないレコード 5 件を表示します。既定では、このプラグインは、キューブの既定リストを使用します。

容量の問題で、ここで使用するためだけのリストを作成するものとします。ShortListing という名前のリストを定義した場合は、プラグインは代わりにこのリストを使用します。

どちらの場合も、プラグインは、リストに定義されている列の右側に Score 列を追加します。

この分析オプションの詳細は、"Text Analytics コンテンツ分析" を参照してください。

4.10 Text Analytics ドメインの管理

このページで説明している機能を使用すると、1 つまたは複数の Text Analytics ドメインが作成されます。こうした Text Analytics ドメインは、Business Intelligence で管理されます ("InterSystems IRIS 自然言語処理 (NLP) の使用法" で説明するように、直接作成する Text Analytics ドメインとは異なります)。これらを変更するには、このページで説明されている API のみを使用する必要があります。

InterSystems IRIS によるこれらのドメインの管理は、ユーザによる操作をほとんど、あるいはまったく必要としない方法で行われます。ここでは、Text Analytics ドメインに精通していて、管理方法の詳細に興味があるユーザを対象に説明します。

ユーザがキューブに NLP メジャーを 1 つ追加すると、システムによって Text Analytics ドメインが 1 つ作成されます。このドメインの名前は DeepSee@cubename@measurename です。cubename はキューブの論理名、measurename は NLP メジャーの論理名です。

InterSystems IRIS では、これらのドメインが以下のように管理されます。

- キューブを初めてコンパイルするときに、必要なドメインが作成されます。
- キューブの構築時に、Analytics エンジンは NLP メジャーのテキストを処理して、その結果を保存します。
- キューブを再コンパイルするときは、必要なドメインが存在するかどうかのチェックが行われます。存在する場合、それらが再利用されます。存在しない場合、作成されます。

  特定のドメインが再利用可能かどうかのチェックでは、各 NLP メジャーの (論理名ではなく) ソース値またはソース式が考慮されます。したがって、NLP メジャーの名前を変更した場合も、既存の Text Analytics ドメインが再利用されます。
- NLP メジャーを削除してキューブを再コンパイルすると、対応する Text Analytics ドメインおよび関連付けられていたエンジンのすべての結果が削除されます。
- キューブを削除すると、その Text Analytics ドメインおよび関連付けられていたエンジンのすべての結果が削除されます。

4.11 その他のトピック

このセクションでは、以下のその他のトピックについて説明します。

- 既定の Text Analytics ドメイン・パラメータをオーバーライドする方法
- キューブで使用するために skiplist をロードする方法
- skiplist を更新する方法

4.11.1 メジャーの Text Analytics ドメイン・パラメータの指定

まれに、指定された NLP メジャーで使用する既定の Text Analytics ドメイン・パラメータをオーバーライドする必要がある場合があります。そのためには、スタジオでキューブ・クラスを編集し、該当する NLP メジャーの定義に以下を追加します。

```
iKnowParameters="parameter::value;parameter::value"
```

parameter には、パラメータ名またはパラメータを表すマクロを使用します。パラメータとその値の間に 2 つのコロンを使用します。リストの名前と値のペア同士の間は、セミコロンで区切ります。

以下の例では、`DefaultConfig` パラメータおよび `MAT:SkipRelations` パラメータの既定の値がオーバーライドされます。

```
iKnowParameters="DefaultConfig::Spanish;MAT:SkipRelations::0"
```

Text Analytics ドメイン・パラメータの詳細は、"NLP 環境作成の代替方法" を参照してください。

4.11.2 skiplist のロード

skiplist とは、クエリから返されることを望まないエンティティのリストです。Business Intelligence で使用するために skiplist をロードするには、以下の手順を実行します。

1. skiplist 項目で構成される条件リストを作成します。用語リストの作成の詳細は、"条件リストの定義" を参照してください。
2. この skiplist を使用する NLP メジャーを編集します。そのためには、スタジオでメジャーを編集し、`iKnowParameters` 属性を指定します。この属性には、1 つ以上の名前と値のペアが含まれています。名前は Text Analytics ドメイン・パラメータで、値はそれに対応する値です。`iKnowParameters` の指定に関する一般情報は、このページで前述した "メジャーの Text Analytics ドメイン・パラメータの指定" を参照してください。

   この場合、Text Analytics ドメイン・パラメータは `$$$IKPDSSKIPLIST` で、その値は条件リスト名です。
3. キューブを再構築するか、条件リストを skiplist として手動でロードします。

   条件リストを skiplist として手動でロードするには、以下の %iKnow.DeepSee.CubeUtils のクラス・メソッドを使用します。

   ```
   classmethod LoadTermListAsSkipList(pCube As %String,
                                      pMeasure As %String,
                                      pTermList As %String) as %Status
   ```

   以下は、この指定の説明です。

   - pCube は、この NLP メジャーを使用するキューブの名前です。
   - pMeasure は、この skiplist を使用する NLP メジャーの名前です。
   - pTermList は、用語リストの名前です。

この skiplist は、以下の目的で使用されます。

- アナライザで、または直接 MDX を使用して、エンティティ・ディメンジョンから返されたエンティティをフィルタ処理する。
- [エンティティ分析] 画面に表示される上位グループの派生からのエンティティを除外する ("[ピボット分析] ウィンドウの使用" を参照)。skiplist のエントリ (またはそれらの標準化された形式) は、それら自体でグループになることはありませんが、他のグループのスコアに引き続き寄与します。例えば、`pilot` が skiplist にある場合でも、`helicopter pilot` は、`helicopter` グループにまだ属しています。

- [エンティティ分析] 画面の [エンティティの詳細] タブに表示されるエンティティをフィルタ処理する。

skiplist は、[エンティティ分析] 画面の **[文字列の分析]** オプションの補助テキスト・エントリには影響しません。

4.11.3 skiplist の更新

skiplist を更新するには、対応する条件リストを編集してから、キューブを再構築するか、条件リストを skiplist として手動でロードします (前のサブセクションを参照)。

4.12 NLP の更新が実行されるタイミング

以下の表は、Analytics エンジンがディクショナリ、skiplist、およびマッチング結果を更新するタイミングをまとめたものです。

操作	Analytics エンジンによる NLP の自動更新
キューブのコンパイル	なし
キューブの構築	• 新しい条件リストを追加することによって、このキューブが使用するすべてのディクショナリを更新 (削除した項目および名前を変更した項目は影響を受けません) • キューブが使用する条件リストのすべての skiplist の完全な更新 • 一致する結果の完全な更新
キューブの同期	• ディクショナリまたは skiplist の更新なし • ファクト・テーブル内の新しいレコードの結果の作成
API または管理ポータルを使用した条件リストの更新	なし
%iKnow.DeepSee.CubeUtils の UpdateDictionary() メソッドを使用した条件リストの更新	• 指定した条件リストが表すディクショナリの完全な更新 • 指定した条件リストに対する一致する結果の完全な更新

5
条件リストの定義

用語リストにより、プログラミングせずに InterSystems IRIS® データ・プラットフォームの Business Intelligence モデルをカスタマイズする方法が提供されます。ここでは、これらを定義する方法を説明します。

"BI サンプルのアクセス方法" も参照してください。

5.1 条件リストの概要

条件リストにより、プログラミングせずに Business Intelligence モデルをカスタマイズする方法が提供されます。条件リストは、[Key] と [値] のペアの単純なリスト (ただし、拡張可能) です。キー値はリスト内で一意であることが必要です。カスタム・フィールドを追加することで、条件リストを拡張できます。条件リストを以下の方法で使用できます。

- 条件リストを使用すると、別のデータを定義して、キューブで使用できます。この場合、キューブが構築されると、条件リストが使用されます。例えば、条件リストを使用すると、値のセットを定義して、レベルのプロパティとして使用できます。または、条件リストを使用すると、レベル・メンバの名前に別のセットを提供できます。
- 条件リストを実行時に使用すると、別の値にアクセスできます。(この場合、LOOKUP 関数および %LOOKUP 関数を使用します。)
- 条件リストを実行時に使用すると、メンバ・セットを作成できます。これは一般的にフィルタで使用します。この場合、条件リスト・パターンを指定する必要があります。条件リストを %TERMLIST 関数で使用できます。

Tipヒント 条件リストを使用すると、構造化されていないデータで使用するために、マッチング・ディクショナリを定義できます。"構造化されていないデータのキューブでの使用" を参照してください。

5.2 条件リスト・マネージャへのアクセス

条件リストの定義や変更を行うには、条件リスト・マネージャを使用します。このツールにアクセスする手順は以下のとおりです。

1. [InterSystems ランチャー] を選択し、[**管理ポータル**] を選択します。

   セキュリティの設定によっては、InterSystems IRIS ユーザ名とパスワードを使用してログインするように求められます。

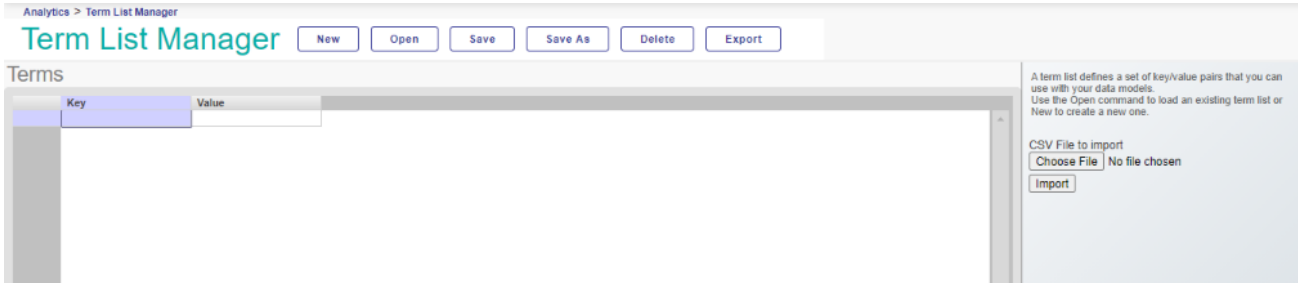
2. 以下のように、適切なネームスペースに切り替えます。

   a. 現在のネームスペースの名前を選択して、使用可能なネームスペースのリストを開きます。

b. リストから、該当するネームスペースを選択します。

3. [Analytics]→[**ツール**]→[**条件リスト・マネージャ**] を選択します。

[**条件リスト・マネージャ**] ページは最初、以下のようになります。



このページでは、次の操作を実行できます。

- 新規条件リストを作成します。[**新規**] を選択し、"条件リストの定義" の説明に従って作業を進めます。
- 既存の条件リストを開いて変更します。[**開く**] を選択して、条件リストを選択し、"条件リストの定義" の説明に従って作業を進めます。
- 条件リストをエクスポートおよびインポートします。"条件リストのエクスポートおよびインポート" を参照してください。

5.3 条件リストの定義

条件リストを定義するには、[**新規**] を選択します。新しい条件リストの名前の入力を求めるダイアログ・ボックスが表示されます。名前を入力して、[OK] を選択します。必要に応じて、以下の作業を行います。

- 条件リストについての情報を、[**詳細**] タブの以下のオプション・フィールドに入力します。
  - [**キャプション**] ― 条件リストのオプション・キャプション
  - [**サブジェクト領域**] ― 条件リストが使用されるサブジェクト領域を選択します。
  - [**説明**] ― 条件リストに関する長い説明

  [**パターン**] の詳細は、"条件リストのパターンの指定" を参照してください。
- 用語を追加します。そのためには、右側の [**条件の追加**] を選択します。または、[新しい条件] ボタン を選択します。

  新しい行が条件リストの一番下に追加されます。
- カスタム・フィールドを追加します。そのためには、[**フィールド名**] に値を入力してから、右側の [**フィールドの追加**] ボタン を選択します。または、[新しいフィールド] ボタン を選択します。

  新しい列が条件リストの右側に追加されます。
- 値をセルに入力します。値を入力するには、セルを選択して値を入力します。次に、カーソルを別のセルに移動し、Enter キーを押すか、[OK] を選択します。
- 用語を削除します。そのためには、その条件の行を選択してから、[**詳細**] タブの [**条件の削除**] を選択します。
- カスタム・フィールドを削除します。そのためには、カスタム・フィールドの列を選択してから、右側の [**フィールドの削除**] ボタン を選択します。

その後、条件リストを保存します。そのためには、[保存] または [名前を付けて保存] を選択します。[名前を付けて保存] を選択した場合、新しい名前を指定してから、[OK] を選択します。

5.4 条件リストのパターンの指定

%TERMLIST 関数は、既定で、条件リストのキー値によって指定されるメンバと条件リスト・パターンの組み合わせで構成されるセットを返します。

条件リストを %TERMLIST 関数で使用する場合、条件リストを以下のように定義します。

- 用語ごとに [値] フィールドをメンバ名か数値キーとして指定します。
- [パターン] には、以下のいずれかの形式の式を指定します。
  - [値] フィールドにメンバ名がある場合、以下の形式を使用します。

    ```
    [dimension name].[hierarchy name].[level name].[*]
    ```

  - [値] フィールドにメンバ・キーがある場合、以下の形式を使用します。

    ```
    [dimension name].[hierarchy name].[level name].&[*]
    ```

どちらの場合も、dimension name、hierarchy name、および level name は、メンバが属するディメンジョン、階層、およびレベルの名前です。アスタリスクは [値] フィールドの値を表します。例については、"%TERMLIST" を参照してください。

5.5 条件リストのエクスポートおよびインポート

条件リストをコンマ区切りファイルにエクスポートしたり、条件リストをそのようなファイルからインポートすることができます。

5.5.1 条件リストのエクスポート

条件リストをエクスポートする手順は以下のとおりです。

1. 条件リストを条件リスト・マネージャで表示します。
2. [エクスポート] を選択します。システムによってファイルが生成されます。

   ブラウザの設定に応じて、ブラウザでファイルを自動的に保存する (このブラウザの既定ダウンロード・ディレクトリ内) か、ファイルを既定のアプリケーションで開きます。

5.5.2 サンプル条件リスト・ファイル

以下にサンプル条件リスト・ファイルの内容を示します。

```
%%NAME,My Term List
%%CAPTION,My Caption
%%DESCRIPTION,My Description
%%SUBJECTAREA,HoleFoods
%%MODDATE,2014-06-06 11:31:10
Atlanta,Braves
Boston,Red Sox
New York,Yankees
```

エクスポートの最初の行には、先頭が %% の項目があります。これらの項目には条件リストに関する説明的な情報が記載されています。

5.5.3 条件リストのインポート

条件リストをインポートする手順は以下のとおりです。

1. 右側の領域にある [**インポートする CSV ファイル**] で、[**参照...**] を選択してファイルを探して選択します。
2. [**インポート**] を選択します。

   InterSystems IRIS で、このファイルに含まれている条件リストの名前が調べられます。次に、以下の操作を行います。

   - 保存された条件リストを同名でシステムに格納している場合は、条件リストを上書きするかどうかを尋ねられます。条件リストを上書きする可能性がある場合は、[**はい**] を選択します。そうでない場合は、[**キャンセル**] を選択してインポート操作をキャンセルします。

     [**はい**] を選択すると、システムによってファイルがインポートされ、条件リストの新しい定義が表示されますが、変更内容は保存されません。
   - 保存された条件リストが同じ名前でまだシステムに保存されていない場合は、ファイルがインポートされ、新しい条件リストが表示されますが、新しい条件リストは保存されません。
3. 必要に応じて、新規条件リストや変更した条件リストを保存します。そのためには、[**保存**] または [**名前を付けて保存**] を選択します。[**名前を付けて保存**] を選択した場合、新しい名前を指定してから、[OK] を選択します。

5.6 条件リストの削除

条件リストを削除する手順は以下のとおりです。

1. 条件リストを条件リスト・マネージャで表示します。
2. [**削除**] を選択します。
3. [OK] を選択します。

5.7 プログラムによる条件リストへのアクセス

%DeepSee.TermList クラスを使用すると、プログラムにより条件リストにアクセスできます。このクラスは、以下のようなクラス・メソッドを提供します。

- %ExportCSV()
- %GetTermLists()
- %GetValueArray()
- %Lookup()
- %TermListExists()
- その他

″インターシステムズ・クラス・リファレンス″ を参照してください。

6
ワークシートの定義

管理ポータルには、[**ワークシート・ビルダ**] オプションがありますが、このオプションに関する文書による説明はなくなりました。ワークシートは非推奨になりました。

7

品質メジャーの定義

ここでは、Business Intelligence で使用するために、品質メジャーを定義する方法を説明します。品質メジャーは、計算メジャーに似ています (″計算メンバの定義″ を参照してください)。

品質メジャーとその他の基本 Business Intelligence モデル要素との比較については、″モデル・オプションの概要″ を参照してください。

7.1 品質メジャーの概要

品質メジャーは、複数のコンテキストで再利用可能な計算メジャーに似ています。品質メジャーは、MDX 式を組み合わせた数式で定義できます。数式を使用可能なサブジェクト領域を 1 つまたは複数指定すると、品質メジャーをパブリッシュするかどうか (およびそれによってアナライザで使用できるようにするかどうか) を制御できます。

各品質メジャーは、InterSystems IRIS® データ・プラットフォームのクラス定義であり、具体的には %DeepSee.QualityMeasure.QualityMeasure のサブクラスです。

品質メジャーは、品質メジャー・マネージャまたは IDE のどちらかで定義できます。ここでは、品質メジャー・マネージャを説明します。

品質メジャーは、以下のように使用できます。

- パブリッシュされている品質メジャーは、アナライザの左側の領域の [品質メジャー] セクションに表示されます。これらはドラッグして、ピボット・テーブルにドロップできます。
- 品質メジャーは、MDX クエリで使用できます。MDX で品質メジャーを参照するには、以下の構文を使用します。

  ```
  [%QualityMeasure].[catalog/set/qm name]
  ```

  catalog は品質メジャーが属するカタログ、set はそのカタログ内のセット、qm name は品質メジャーの略称です (品質メジャーのフルネームは catalog/set/qm name になります)。

  また、以下のように品質メジャーでグループ (group name) を参照することもできます。

  ```
  [%QualityMeasure].[catalog/set/qm name/group name]
  ```

7.2 品質メジャー・マネージャの概要

品質メジャー・マネージャには、指定したネームスペースのすべてのコンパイル品質メジャー・クラスが表示されます。品質メジャー・マネージャでは、品質メジャー・クラスを作成、変更、および削除できます。品質メジャー・マネージャにアクセスする手順は以下のとおりです。

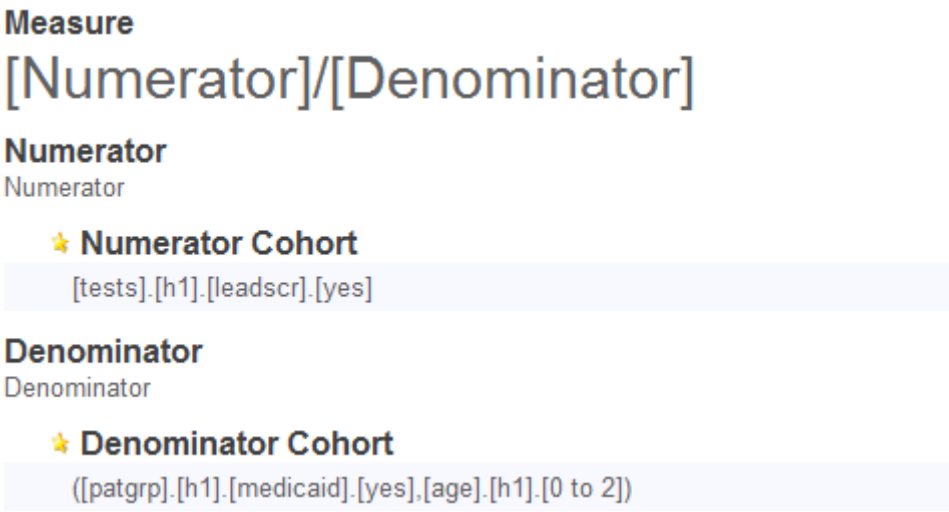
1. 管理ポータルで、[Analytics] を選択します。
2. [**ツール**]→[**品質メジャー**] を選択します。

ここでは次の操作を実行できます。

- このネームスペースの既存の品質メジャーのサマリ情報を表示します。そのためには、[**参照**] を選択します。

  中央の領域に、以下のように表示されます。

- 品質メジャーの定義を表示します。そのためには、[**開く**] を選択し、その品質メジャーが見つかるまでフォルダを展開して、見つかったメジャーを選択します。中央の領域に表示される詳細の中に、メジャーの定義が表示されたセクションが含まれています。通常は、以下のように表示されます。



このメジャーを編集または削除できます。

- 品質メジャーを作成します。そのためには、**[新規作成]** を選択し、次のセクションの説明に従って操作を続行します。

7.3 品質メジャーの作成

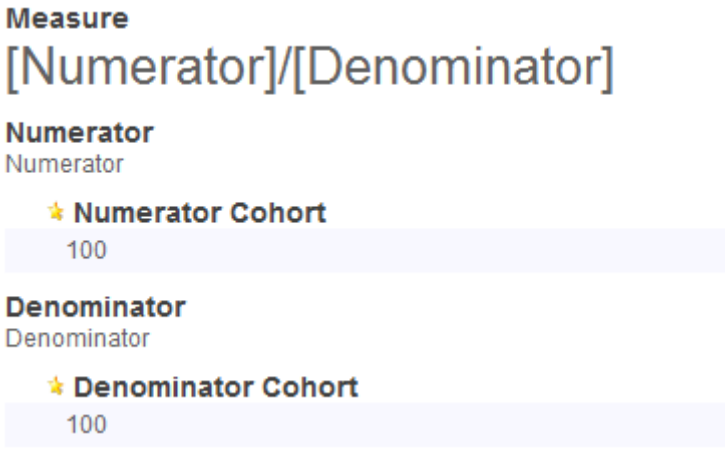
品質メジャーを作成する手順は以下のとおりです。

1. **[新規作成]** を選択します。

   ダイアログ・ボックスが表示されます。

2. 以下の値を指定します。これらはすべて必須です。
   - **[カタログ]** で、既存のカタログを選択するか、または新しいカタログの名前を入力します。
   - **[セット]** で、既存のセットを選択するか、または新しいセットの名前を入力します。
   - **[名前]** で、名前を入力します。
   - **[品質メジャーのクラス名]** で、完全修飾クラス名 (パッケージおよびクラス) を入力します。
3. [OK] を選択します。

   ダイアログ・ボックスが閉じ、ページの中央の領域に、品質メジャーの初期定義が以下のように表示されます。



ここに示すように、品質メジャーはすべて、**[メジャー]** フィールドで数式として表されます。この場合、数式は以下のとおりです。

```
[Numerator]/[Denominator]
```

Numerator と Denominator は、この品質メジャーで定義されているグループの名前です。

各グループは 1 つまたは複数の項目で構成され、各項目は MDX 式によって定義されます。

品質メジャーの初期定義では、これらのグループはそれぞれ定数 (100) として定義されます。この品質メジャーの初期値は 1 です。

他の情報は、ページの下部領域に表示されます。

4. **[編集]** を選択します。

   これにより、数式を変更したり、他の変更を実行できます。この後のセクションを参照してください。

5. 操作が完了したら、**[保存]** を選択して定義を保存するか、**[名前を付けて保存]** を選択して新しい名前で定義を保存します。

   定義を保存すると、自動的にクラスがコンパイルされ、品質メジャー定義がシステム・グローバルに書き込まれます。

7.4 品質メジャーの式の指定

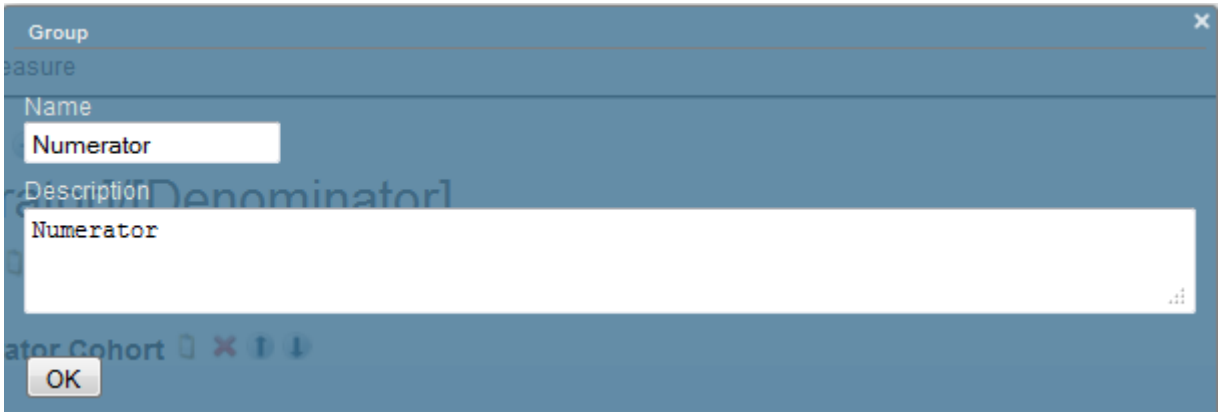
リンクされた品質メジャー以外の品質メジャーを編集モードで表示した場合、式全体を編集できます。初期状態では以下のように表示されます。

ここでは次の操作を実行できます。

- 式全体を変更します。そのためには、編集ボタン を選択します。新しい式を指定して、[OK] を選択します。式は以下のように指定します。
  - 式で使用するグループの名前は角括弧で囲みます。例えば、品質メジャーで `Numerator` グループが定義されている場合、このグループを参照するには `[Numerator]` と指定します。
  - 標準の算術演算子を使用します。
  - 必要に応じて数値定数を使用します。
  - 必要に応じて括弧を使用して優先順位を制御します。

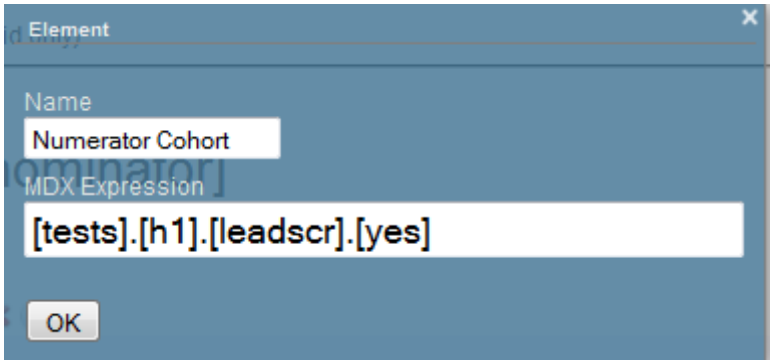
- グループを追加します。そのためには、**[メジャー]** という見出しの横にある追加ボタン を選択します。新しいグループを編集します。このグループは、定義の最後に追加されます。

- グループを編集します。そのためには、編集ボタン を選択します。必要に応じて **[名前]** と **[説明]** に新しい値を指定して、[OK] を選択します。



- 要素をグループに追加します。そのためには、グループ名の横にある追加ボタン を選択します。新しい要素を編集します。

- グループの要素を編集します。そのためには、要素名の横にある編集ボタン を選択します。必要に応じて **[名前]** と **[MDX 式]** に新しい値を指定して、[OK] を選択します。



[MDX 表現] の詳細は、"許可される MDX 式" を参照してください。

- メジャー定義内のグループの順序を変更するか、またはグループ内の要素の順序を変更します。そのためには、移動する項目の横にある上下の矢印を選択します。
- グループまたは要素を削除します。そのためには、削除する項目の横にある [X] ボタンを選択します。システムから、このアクションの確認を求められます。

7.4.1 許可される MDX 式

[MDX 式] には、以下の種類の式のいずれかを指定します。

- 例に示したような、メンバへの参照。これは、品質メジャーで使用される最も一般的な種類の式です。メンバは一連のレコードで、品質メジャーは、通常、異なるレコード・セット間の比較で構成されます。
- 他の種類のメンバ式のほとんど。
- 42 などの数値リテラルおよびその他の種類の数値式。例えば、AVG や COUNT などの関数を使用します。
- "my label" などの文字列リテラルおよびその他の種類の文字列式の一部。PROPERTIES 関数および IIF 関数はここではサポートされていません。
- タプル式。

[MDX 式] をセット式にすることはできません。ただし、%OR を使用してセットを囲み、単一のメンバとして返すことは可能です。

検索式は、品質メジャー内ではサポートされていません。

MDX 式の作成の詳細は、"InterSystems MDX の使用法" および "InterSystems MDX リファレンス" を参照してください。

7.4.2 グループおよび要素の結合方法

品質メジャーの数式 ([**メジャー**] の見出しの下に表示されている) により、グループの結合方法が決まります。

どのグループ内でも、要素は MDX セットとして結合されます (すなわち、論理 OR によって結合されます)。要素を論理 AND によって結合する必要がある場合は、必要な部分を結合した MDX タプル式を持つ要素を 1 つ作成します。例：

```
([patgrp].[h1].[medicaid].[yes],[age].[h1].[0 to 2])
```

特定のコンテキストにおける品質メジャーの値を計算するために、Business Intelligence によって以下の処理が行われます。

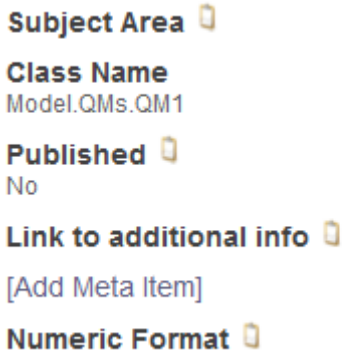
1. グループごとに、そのグループの要素を結合する MDX セット式を作成します。
2. 各グループを評価し、その値を決定します。

3. 数式の指定に従ってグループ値を結合します。

品質メジャーでは、数式で使用されないグループを定義することができます。これは、このページで前述したように、特殊な [%QualityMeasure] ディメンジョンを使用してグループを参照する場合に役立つことがあります。

7.5 品質メジャーの他の情報の編集

品質メジャーを編集モードで表示した場合、[**キャプション**] (ページの上部) およびページの下部の以下の領域も編集できます。



ここでは次の操作を実行できます。

- 項目を編集します。そのためには、その項目の横にある編集ボタン を選択し、詳細を指定します。
- 項目を追加します。そのためには、[**メタ項目の追加**] を選択します。

ここで編集可能な項目は以下のとおりです。

- [**サブジェクト領域**] ― この品質メジャーを使用可能なキューブまたはサブジェクト領域を指定します。

  複数のキューブまたはサブジェクト領域を指定するには、スタジオで品質メジャー・クラスを編集します。

  このリストの任意のキューブに基づく任意のサブジェクト領域で、自動的に品質メジャーが使用可能になることに注意してください。
- [**パブリッシュ**] ― この品質メジャーがアナライザで使用可能かどうかを指定します。
- [**他の情報へのリンク**] ― 他の情報を含む URL へのリンクを指定します。
- カスタム項目 ― 品質メジャー・マネージャに表示する他の情報を指定します。これらの項目は使用されません。

7.6 リンクされた品質メジャー (品質メジャーのエイリアス) の定義

リンクされた品質メジャーは、別の品質メジャー (マスタ・メジャー) のエイリアスです。マスタ・メジャーは、パブリッシュされる場合もされない場合もあります。また、別のサブジェクト領域に属していることもあります。

リンクされた品質メジャーを作成する手順は以下のとおりです。

1. [**新規作成**] を選択します。

ダイアログ・ボックスが表示されます。

2. 以下の値を指定します。これらはすべて必須です。
   - **[カタログ]** で、既存のカタログを選択するか、または新しいカタログの名前を入力します。
   - **[セット]** で、既存のセットを選択するか、または新しいセットの名前を入力します。
   - **[名前]** で、名前を入力します。
   - **[品質メジャーのクラス名]** で、完全修飾クラス名 (パッケージおよびクラス) を入力します。
3. [OK] を選択します。

   ダイアログ・ボックスが閉じ、ページの中央の領域に、品質メジャーの初期定義が表示されます。
4. **[編集]** を選択します。
5. **[リンク先]** で、親メジャーを選択し、[OK] を選択します。

   既存の品質メジャーに対して **[リンク先]** を指定すると、そのメジャーのオプションのプロパティの値は破棄されます。
6. 必要に応じて、以下のオプションも指定します。
   - **キャプション**
   - **サブジェクト領域**
   - **パブリッシュ**

   その他のオプションは、マスタ・メジャーから継承されます。
7. 操作が完了したら、**[保存]** を選択して定義を保存するか、**[名前を付けて保存]** を選択して新しい名前で定義を保存します。

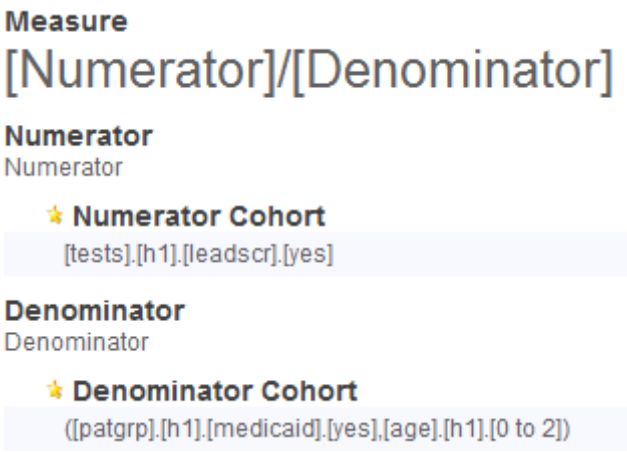
   定義を保存すると、自動的にクラスがコンパイルされ、品質メジャー定義がシステム・グローバルに書き込まれます。

7.7 品質メジャーの式のチェック

品質メジャーを定義する式全体を、特に定義が複雑な場合、チェックすることがあります。そのためには、品質メジャーの %GetExpression インスタンス・メソッドを使用します。以下に例を示します。

```
SAMPLES>set qm=##class(QM.Preventive.Child.QM7).%New()

SAMPLES>w qm.%GetExpression()
[tests].[h1].[leadscr].[yes]/([patgrp].[h1].[medicaid].[yes],[age].[h1].[0 to 2])
```

これは、以下の品質メジャーの式です。



7.8 品質メジャーの削除

品質メジャーを削除する手順は以下のとおりです。

1. その定義を表示します。
2. [削除] を選択します。
3. [OK] を選択します。

8

基本的な KPI の定義

ここでは、Business Intelligence の重要業績評価指標 (KPI) の概要を示し、ハードコードされたクエリを使用する KPI の定義方法を説明します。

KPI とその他の種類のモデル要素との比較については、"モデル・オプションの概要" を参照してください。

"フィルタおよびリストを含む KPI の定義" および "高度な KPI の定義" も参照してください。

Text Analytics に基づく KPI の定義に関する詳細は、"KPI とダッシュボード" を参照してください。

"BI サンプルのアクセス方法" も参照してください。

8.1 KPI の概要

KPI は、**%DeepSee.KPI** をベースにしたクラスです。多くの場合、KPI はクエリを使用して、結果セットを表示します (それ以外は、KPI はアクションのみ定義します。"カスタム・アクションの定義" を参照してください)。

8.1.1 KPI の使用法

KPI は、ピボット・テーブルと同様に、ダッシュボードで、ウィジェット内に、表示できます。

また、MDX %KPI 関数を使用して、KPI の値を取得することもできます。その結果として、KPI に基づいた計算メンバを定義できます。

ObjectScript から KPI の値にアクセスするには、KPI クラスの %GetKPIValueArray() メソッドを使用します。例については、"ハードコードされたクエリを使用した KPI の定義" を参照してください。

8.1.2 ピボット・テーブルとの比較

KPI は、さまざまな点でピボット・テーブルと似ていますが、ピボット・テーブルにはないオプションが追加されています。違いの 1 つとして、KPI は SQL クエリを使用できます。SQL クエリと MDX クエリは適したシナリオが異なるので、これは重要な違いです。場合によっては SQL クエリのほうが効果的であり、そのような場合は KPI で SQL クエリを使用する必要があります。

KPI とピボット・テーブルの他の相違点と類似点は、"モデル・オプションの概要" を参照してください。

8.1.3 KPI クエリの要件

多くの場合、KPI は MDX クエリまたは SQL クエリのどちらかを使用します。クエリの形式にはルールがあります。このルールは、KPI 結果セットの構造によって定められるものです (後続のセクションを参照してください)。

- クエリで MDX を使用する場合、以下の要件に注意してください。
  - クエリは行に MEMBERS 関数を使用する必要があります。ネストした行を使用できます。
  - クエリは列にメジャーを使用する必要があります。
  - クエリでは、列にネストを使用できません。
- クエリは数値を返す必要があります。
- クエリが 1000 行を超える行を返す場合は、最初の 1000 行のみが使用されます。

これらのルールに従わないクエリを使用できます。そのためには、結果セットを解析して、直接 KPI インスタンスのプロパティを指定する必要があります。詳細は、"高度な KPI の定義" を参照してください。

メータで KPI を表示する場合、KPI の最初の行のみが使用されることにも注意してください。

8.2 MDX と SQL のどちらかを選択する方法

SQL クエリと MDX クエリは、適したシナリオが異なり、場合によっては SQL クエリのほうが効果的です。

MDX は一般に、大量のレコードを集約する場合のほうが適しています。一方で、集約をまったく行わない場合、または下位レベルでのみ集約する場合は、SQL のほうが適しています。例えば、次のようなピボット・テーブルがあるとします。

PatientID	Patient Count	Age	Allergy Count
SUBJ_100301	1	2	
SUBJ_100302	1	79	
SUBJ_100303	1	17	2
SUBJ_100304	1	72	
SUBJ_100305	1	66	1
SUBJ_100306	1	44	
SUBJ_100307	1	66	1

このピボット・テーブルでは、各行がソース・テーブルの 1 行を表しています。同等の SQL クエリのほうが高速です。

8.3 KPI 結果セットの構造

KPI の結果セットは、系列とプロパティで構成されています。

KPI 系列は行です。以下の例は、9 つの系列を示しています (このページで後述する KPI テスト・ページに表示されます)。各系列には名前があり、ここでは最初の列に表示されています。

KPI Values

Series	PatCount	Population
Cedar Falls	1153	90000
Centerville	1120	49000
Cypress	1170	3000
Elm Heights	1163	33194
Juniper	1035	10333
Magnolia	1147	4503
Pine	1147	15060
Redwood	1086	29192
Spruce	1084	5900

KPI プロパティはデータ列です。前述の例は、2 つのプロパティを持つ KPI を示しています。

MDX クエリに基づく KPI の場合、系列は通常レベルのメンバに対応し、プロパティは通常メジャーに対応します。

8.4 ハードコードされたクエリを使用した KPI の定義

ハードコードされたクエリを使用する単純な KPI を作成するには、スタジオで以下の手順を実行します。

1. **[ファイル]→[新規作成]** を選択してから、**[カスタム]** タブ、**[Business Intelligence KPI の新規作成]** の順に選択します。
2. 以下の必須の値を指定します。
   - **Package Name** ― KPI クラスを含むパッケージ。
   - **Class Name** ― KPI クラスの略称。
3. 必要に応じて、以下に示す他の値を指定します。
   - **[KPI キャプション]** ― 使用されていません。
   - **[KPI 名]** ― KPI の論理名。
   - **[説明]** ― KPI の説明。クラスのコメント行として保存されます。
   - **[ドメイン]** ― この KPI が属するローカライズ・ドメイン。詳細は、"InterSystems Business Intelligence の実装" を参照してください。
   - **[リソース]** ― この KPI を保護するリソース。この使用法については、"セキュリティの設定" を参照してください。
   - **[ソース・タイプ]** ― この KPI のデータのソースを指定します。[mdx] または [sql] のどちらかを選択します (**[手動]** の詳細は、"高度な KPI の定義" を参照してください)。
   - **[プロパティ]** ― この KPI のプロパティの名前 (結果セットの列名) を入力します。1 行につき 1 つのプロパティを入力します。
   - **[フィルタ]** ― KPI クエリで使用するフィルタの名前を入力します。"フィルタおよびリストを含む KPI の定義" を参照してください。1 行につき 1 つのフィルタ名を入力します。
   - **[アクション]** ― KPI で定義するアクションの名前を入力します。"カスタム・アクションの定義" を参照してください。1 行につき 1 つのアクション名を入力します。

   これらの値はすべて、後で同じように編集できます。
4. **[完了]** を選択します。

ウィザードにより、以下のようなクラス定義が生成されます。

```
Class MyApp.KPI.MyKPI Extends %DeepSee.KPI
{

Parameter DOMAIN = "MyAppDomain";

Parameter RESOURCE = "KPI_Resource";

/// This XData definition defines the KPI.
XData KPI [ XMLNamespace = "http://www.intersystems.com/deepsee/kpi" ]
{
<kpi xmlns="http://www.intersystems.com/deepsee/kpi"
 name="MyKPI" sourceType="mdx"
 caption="MyCaption"
>
<property name="PatCount" displayName="PatCount" columnNo="1"/>
<property name="AvgAge" displayName="AvgAge" columnNo="2"/>
</kpi>
}
```

XData ブロックは、KPI を定義します。XData ブロックの <kpi> は XML 要素です。この要素は、<kpi で始まり、右山括弧で終了します。xmlns、name、sourceType、および caption は XML 属性です。各属性に値があります。この例では、sourceType 属性の値は mdx です。

このクラスには、さまざまなメソッドのスタブ定義も含まれています。既定では、これらは何も実行しません。詳細は、"フィルタおよびリストを含む KPI の定義" および "高度な KPI の定義" も参照してください。

5. <kpi> 要素内に、以下の属性指定のどちらかを追加します。

```
mdx="MDX query"
```

または以下のようになります。

```
sql="SQL query"
```

MDX query は MDX SELECT クエリ、SQL query は SQL SELECT クエリです。(ウィザードで mdx を選択した場合は mdx オプションを使用し、sql を選択した場合は sql オプションを使用します)。

例えば以下のようになります。

```
<kpi xmlns="http://www.intersystems.com/deepsee/kpi"
 name="MyKPI" sourceType="mdx"
 mdx="SELECT {MEASURES.[%COUNT],MEASURES.[Avg Age]} ON 0, HomeD.H1.City.MEMBERS ON 1 FROM patients"

 caption="MyCaption"
>
```

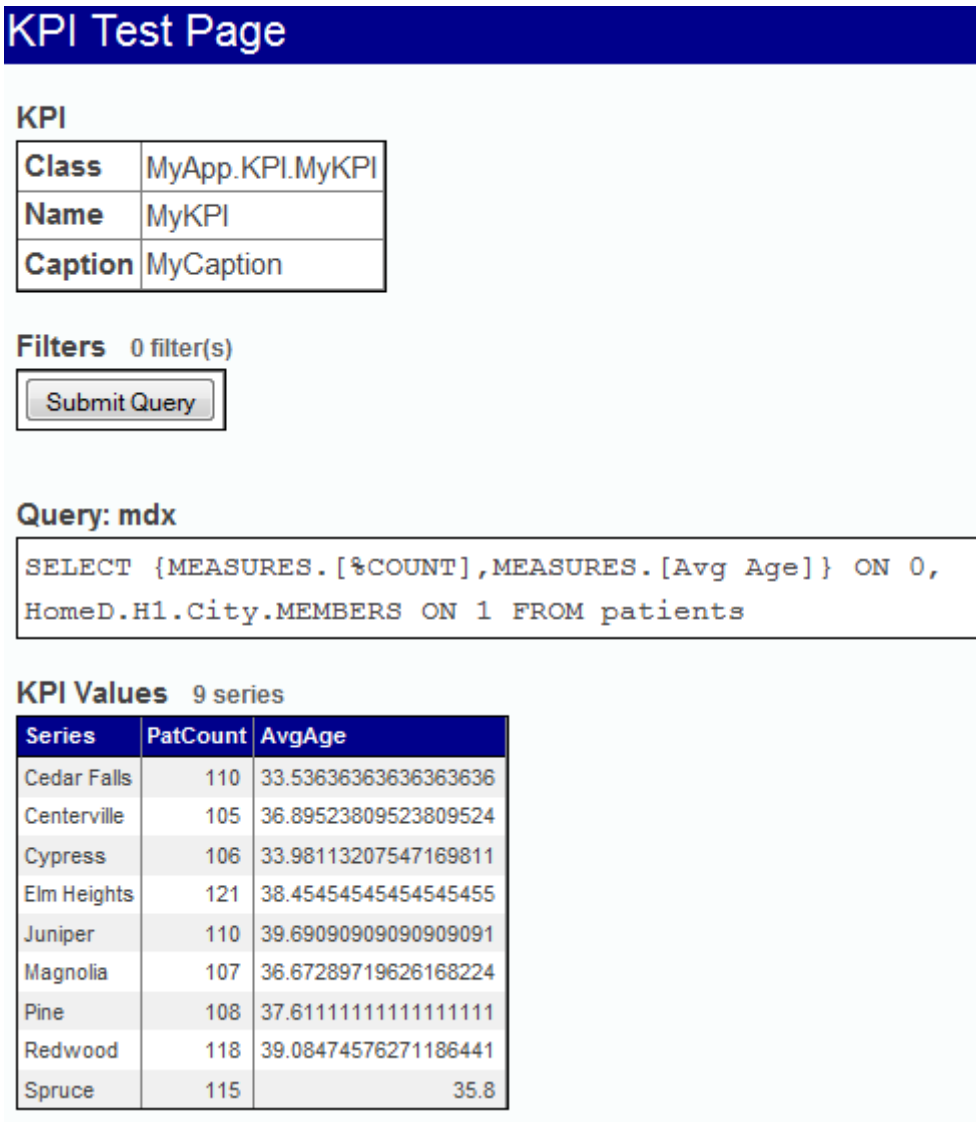
この属性指定は、始まりの <kpi と右山括弧の間の任意の場所に追加できます。属性指定は、上記のように 1 行に独立して記述することも、他の属性と同じ行に記述することもできます。XData ブロック内では、スタジオにより入力支援機能が提供されます。

要件は、このページで前述の "KPI クエリの要件" を参照してください。

MDX の詳細は、"InterSystems MDX の使用法" および "InterSystems MDX リファレンス" を参照してください。

6. 必要に応じて、次のセクションの説明に従って、クラス・パラメータを指定します。
7. クラスをコンパイルします。
8. [ビュー]→[ウェブページ] を選択します。

これにより、表示は以下のようになります。



KPI Test Page

KPI

Class	MyApp.KPI.MyKPI
Name	MyKPI
Caption	MyCaption

Filters 0 filter(s)

Submit Query

Query: mdx

```
SELECT {MEASURES.[%COUNT],MEASURES.[Avg Age]} ON 0,
HomeD.H1.City.MEMBERS ON 1 FROM patients
```

KPI Values 9 series

Series	PatCount	AvgAge
Cedar Falls	110	33.53636363636363636
Centerville	105	36.89523809523809524
Cypress	106	33.98113207547169811
Elm Heights	121	38.45454545454545455
Juniper	110	39.69090909090909091
Magnolia	107	36.67289719626168224
Pine	108	37.61111111111111111
Redwood	118	39.08474576271186441
Spruce	115	35.8

[**系列**] 列は、各系列の名前を示します。この名前は、この KPI をスコアカードに表示する際にラベルとして使用できます。

このテーブルには、これらの列の右に、KPI の `<property>` ごとに 1 つの列があります。この列には、KPI の行ごとにそのプロパティの現在の値が表示されます。

KPI テスト・ページでは、使用前の KPI を簡単にテストできます。KPI クラスの %GetKPIValueArray() メソッドを使用することもできます。以下に例を示します。

```
SAMPLES>set
status=##class("HoleFoods.KPIYears").%GetKPIValueArray("HoleFoods.KPIYears",.pValues,$LB("Value"))

SAMPLES>w status
1
SAMPLES>set
status=##class("HoleFoods.KPIYears").%GetKPIValueArray("HoleFoods.KPIYears",.pValues,$LB("Value"))

SAMPLES>w status                                                                1
SAMPLES>zw pValues                                                             pValues(1)=$lb("2010")
pValues(2)=$lb("2011")
pValues(3)=$lb("2012")
pValues(4)=$lb("2013")
pValues(5)=$lb("2014")
pValues(6)=$lb("2015")
```

詳細は、%DeepSee.AbstractKPI のクラスリファレンスを参照してください。

8.5 クラス・パラメータの指定

KPI クラスで以下のクラス・パラメータの一部または全部を指定できます。

DOMAIN

Class Member

```
Parameter DOMAIN = "MyAppDomain";
```

この KPI が属するローカライズ・ドメインを指定します。詳細は、"InterSystems Business Intelligence の実装" を参照してください。

FORCECOMPUTE

この KPI が MDX クエリ内で使用される場合 (つまり、%KPI 関数を介して)、システムで常にこの KPI の値を再計算するかどうかを指定します。既定は false です。そのクエリが再実行されると、代わりにキャッシュの値が使用されます。

KPI で外部データを使用する場合、FORCECOMPUTE を true に設定すると便利な場合があります。

LABELCONCAT

CROSSJOIN または NONEMPTYCROSSJOIN を行に使用する MDX ベースの KPI で、ラベルを連結するために使用する文字を指定します。既定値はスラッシュ (/) です。

PUBLIC

KPI が、スコアカードおよび他のダッシュボード・ウィジェットで使用可能かどうか、および MDX %KPI 関数で使用可能かどうかを制御します。KPI をダッシュボードで使用できないようにする場合、クラスに PUBLIC クラス・パラメータを追加し、値を 0 に設定します。

RESOURCE

Class Member

```
Parameter RESOURCE = "KPI_Resource";
```

この KPI を保護するリソースを指定します。この使用法については、"セキュリティの設定" を参照してください。

ASYNC クラス・パラメータの詳細は、"高度な KPI の定義" を参照してください。

8.6 速度計の範囲としきい値の指定

KPI の定義内で、速度計で使用する範囲としきい値を指定できます。これらの値を指定するには、<kpi> 要素を編集して、以下の属性を指定します。

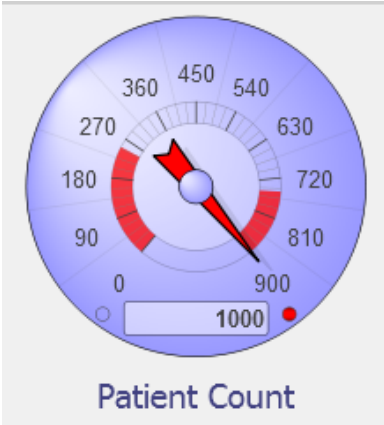
- rangeLower ー メータに表示される下限値の既定値。
- rangeUpper ー メータに表示される上限値の既定値。

- `thresholdLower` ― この KPI の下限しきい値の既定値。しきい値領域は、対比色で表示されます。
- `thresholdUpper` ― 上限しきい値の既定値。

以下に例を示します。

```
<kpi name="KPIForRangeDemos"
sourceType="mdx"
mdx='SELECT MEASURES.[%COUNT] ON 0, AgeD.[All Patients] ON 1 FROM PATIENTS'
rangeLower="0"
rangeUpper="900"
thresholdLower="20"
thresholdUpper="800"
>

<property name="Patient Count" columnNo="1" />

</kpi>
```

この KPI を速度計で表示すると、(既定では) 以下のようになります。



速度計の値ボックスには、`rangeUpper` の値を超えていても、実際の KPI 値 (1000) が表示されていることがわかります。

プログラムによって範囲としきい値を設定することもできます。値をハードコードすることが不適切な場合は、その方法が便利です。"高度な KPI の定義" を参照してください。

注釈　ダッシュボードでスコアカードを構成する際、[**下限のしきい値**]、[**上限のしきい値**]、[**範囲の下限**]、および [**範囲の上限**] の各オプションがあります。KPI 属性 `rangeLower`、`rangeUpper`、`thresholdLower`、および `thresholdUpper` は、これらのスコアカード・オプションに影響しないことに注意してください。

8.7 %CONTEXT フィルタの無効化

前述したように、MDX %KPI 関数を使用して、KPI の値を取得できます。MDX ベースの KPI の場合、%KPI 関数には、KPI にコンテキスト情報を渡すオプション・パラメータ (**%CONTEXT**) があります。既定では、このコンテキスト情報は、フィルタ節として MDX クエリに適用されます。この自動動作を無効にするには、以下のように、%GetMDXContextFilter() メソッドをオーバーライドします。

```
Method %GetMDXContextFilter() As %String
{
    Quit ""
}
```

9

フィルタおよびリストを含む KPI の定義

ここでは、フィルタおよびリストを含む Business Intelligence KPI を定義するオプションを説明します。

このページを読む前に、"基本的な KPI の定義" を参照してください。高度な KPI の詳細は、"高度な KPI の定義" を参照してください。

KPI には、アクションも定義できます。"カスタム・アクションの定義" を参照してください。

"BI サンプルのアクセス方法" も参照してください。

9.1 フィルタの概要

通常のダッシュボードには、1 つ以上のフィルタ・コントロールが組み込まれ、ユーザはこれを使用してダッシュボードのウィジェットと対話します。通常、各フィルタ・コントロールは、ドロップダウン・リストとして表示されます。フィルタで、ウィジェットによって表示されるデータをフィルタ処理できます。または、他の方法でデータ・ソースに影響を与えることができます。

ピボット・テーブルでは、キューブまたはサブジェクト領域の任意のレベルをフィルタとして使用できます。ただし、KPI の場合、既定ではフィルタは定義されていません。フィルタを KPI に追加するには、以下のシステムを使用します。

- 各フィルタを定義します。すなわち、フィルタ・リストのフィルタ名と項目を指定します。

  ピボット・テーブルによって提供されるフィルタを使用する場合は、この手順をスキップできます。

- プログラムによって、KPI クエリでフィルタを使用するように指定します。そのためには、KPI にコールバック・メソッドを実装し、ハードコードされたクエリをオーバーライドするようにします。このメソッド内で、選択したフィルタ値にアクセスできます。

  一般的な例では、クエリで使用されるレコードを制限する WHERE 節 (MDX クエリでは %FILTER) を追加します。その後でユーザがダッシュボードを更新すると、クエリが再実行され、その結果が表示されます。

  KPI クエリは自身で構成するので、より汎用的にフィルタを使用することができます。つまり、クエリの WHERE (または %FILTER) 節に作用するフィルタ選択を使用することは、必須ではありません。極端な例として、フィルタによってオプション 1、2、および 3 が表示されるとします。KPI はクエリのバージョン 1、2、または 3 を実行できますが、これらがまったく別のバージョンである場合もあります。別の例として、フィルタでメジャー名のリストが表示され、KPI がそのクエリに選択されたメジャーを組み込む場合もあります。

9.2 相互運用可能なフィルタの作成

ダッシュボードに KPI とピボット・テーブルの両方を追加して、スコアカード、ピボット・テーブル・ウィジェット、または他のウィジェットに表示できます。ダッシュボード内では、複数のウィジェットに作用するように、フィルタを構成できます。

KPI とピボット・テーブルの両方に作用するフィルタを定義できます。

ピボット・テーブルは、次のサブセクションで説明する固有の構文で、フィルタ値を送受信します。相互運用可能なフィルタを作成するには、この要件を考慮する必要があります。2 つ目のサブセクションで、2 つの考えられる方法について説明します。

9.2.1 ピボット・テーブルのフィルタ構文

以下のテーブルは、ピボット・テーブルで使用および期待されるフィルタ構文を示しています。この構文は、あらゆる種類のダッシュボード・ウィジェットに表示されるピボット・テーブルに適用されます。

	シナリオ	構文	例
フィルタ名	All	MDX レベル識別子	`[Region].[H1].[Country]`
フィルタ値	ユーザが 1 つのメンバを選択する	そのレベルのメンバの MDX キー。これは `&[keyval]` 形式の式であり、keyval はメンバのキー値。	`&[USA]`
フィルタ値	ユーザがメンバの範囲を選択する	以下の形式の式 : `&[keyval1]:&[keyval2]`	`&[2006]:&[2009]`
フィルタ値	ユーザが複数のメンバを選択する	以下の形式の式 (例) : `{&[keyval1],&[keyval2]}`	`{&[2006],&[2007],&[2008]}`
フィルタ値	ユーザが 1 つのメンバを選択して [除外] を選択する	以下の形式の式 : `%NOT&[keyval1]`	`%NOT &[2010]`
フィルタ値	ユーザが複数のメンバを選択して [除外] を選択する	以下の形式の式 (例) : `%NOT{&[keyval1],&[keyval2}`	`%NOT {&[2006],&[2007]}`

オプションの `%NOT` 文字列を除いて、フィルタ名およびフィルタ値は大文字と小文字が区別されません。

9.2.2 相互運用可能なフィルタを作成する方法

KPI とピボット・テーブルの両方に作用可能なフィルタを作成する汎用的な方法として、以下の 2 つがあります。

- ピボット・テーブルが要求する形式のフィルタを定義して使用する KPI を作成できます。
- ピボット・テーブル形式から KPI クエリで必要な形式にフィルタ名と値を変換する KPI を定義できます。

これらの方法は、MDX ベースまたは SQL ベースのどちらの KPI でも使用できますが、SQL ベースの KPI を使用する場合は、値の変換により多くの処理が必要になります。

MDX の例は、サンプル・ダッシュボード Demo Filter Interoperability を参照してください。このダッシュボードは、1 番目の方法のデモです。このダッシュボードには、2 つのピボット・テーブル・ウィジェットが表示され、上側にピボット・テーブル、下側に KPI が表示されます。このダッシュボードの左側の [フィルタ] ワークリストには、以下のフィルタが含まれています。

- Favorite Color フィルタは、上側のウィジェットの一部として構成されており、そのウィジェットに表示されるピボット・テーブルによって定義されています。

  Favorite Color は、このピボット・テーブルの基となるキューブのレベルです。

- Doctor Group フィルタは、下側のウィジェットの一部として構成されており、そのウィジェットに表示される KPI によって定義されています。

  このフィルタは、この KPI 内にプログラムによって定義されています。このフィルタは、前のセクションで示されているピボット・テーブル形式の値を使用するように、定義されています。

どちらのフィルタの場合も、ターゲットは * (このダッシュボードのすべてのウィジェットを示します) です。試してわかるように、どちらのフィルタも、両方のウィジェットに作用します。

KPI には別のフィルタ (Yaxis) も含まれています。このフィルタは、KPI で使用する MDX クエリの y 軸を制御します。このフィルタは、ピボット・テーブルには作用しません。

これが作用する方法の詳細は、このページで後述する "その他の MDX KPI の例" を参照してください。

9.3 MDX ベースの KPI でのフィルタの定義

MDX ベースの KPI でフィルタを定義するには、以下の手順を実行します。

1. フィルタのリストを指定します。各フィルタには、論理名 (必須) および表示名 (既定では論理名と同じ) があります。

   1 つのオプションは、フィルタのリストをハードコードすることです。そのためには、一連の <filter> 要素を <kpi> 要素に追加します。以下に例を示します。

   ```
   <kpi name="sample KPI"...>
    <filter name="[aged].[h1].[age group]" displayName="Age Group"/>
    <filter name="[gend].[h1].[gender]" displayName="Gender"/>
    <filter name="[homed].[h1].[zip]" displayName="ZIP Code"/>
   ...
   ```

   この例では、論理フィルタ名は MDX レベル識別子です。この方法で論理名を指定すると便利です (ただし、必須ではありません)。

   代わりに、実行時にフィルタ名を指定することもできます。このページで後述する "フィルタ名および項目を定義するその他のオプション" を参照してください。

2. <filter> 要素を追加する場合、必要に応じて以下のオプションを指定します。

   - ユーザがフィルタ・リストから要素を 1 つのみ選択できるように設定できます (既定では複数の要素)。そのためには、<filter> 要素に multiSelect="false" を追加します。以下に例を示します。

     ```
      <filter name="[homed].[h1].[zip]" displayName="ZipCode" multiSelect="false" />
     ```

     複数選択を無効にすると、[除外] オプションも無効になります。

   - フィルタ項目が日数の場合、既定のドロップダウン・リストではなくカレンダー・コントロールを表示できます。そのためには、<filter> 要素に searchType="day" を追加します。以下に例を示します。

     ```
      <filter name="[birthd].[h1].[day]" displayName="Day"
     filterProperty="Day" searchType="day"/>
     ```

- 他のフィルタを基にするフィルタを作成できます。そのためには、オプションの dependsOn 属性を使用します。以下に例を示します。

```
<filter name="[homed].[h1].[zip]" displayName="ZIP Code"/>
<filter name="[homed].[h1].[city]" displayName="City" dependsOn="[homed].[h1].[zip]"/>
```

3. 各フィルタにフィルタ項目を定義します。各フィルタ項目には、論理名 (必須) および表示名 (既定では論理名と同じ) があります。

   論理名が MDX メンバのキーと同じだと便利です。そうでない場合、このページで後述するように、フィルタ節を構築するためにより多くの作業が必要になります。

   1 つ目のオプションは、%OnGetFilterMembers() メソッドを実装する方法です。フィルタの論理名が MDX レベル識別子の場合、以下の単純な実装を使用できます。

   Class Member

```
ClassMethod %OnGetFilterMembers(pFilter As %String, Output pMembers As %List,pSearchKey As %String
 = "") As %Status
{
    set status = $$$OK

    try {
        do ..%GetMembersForFilter("Patients.cube",pFilter,.pMembers,pSearchKey)
    }
    catch(ex) {
        set status = ex.AsStatus()
    }

    quit status
}
```

   単純に Patients をキューブの名前に置き換えます。以下に例を示します。

```
 do ..%GetMembersForFilter("YourCube.cube",pFilter,.pMembers,pSearchKey)
```

   %OnGetFilterMembers() の詳細は、最初のサブセクションを参照してください。ここに示す例では、%GetMembersForFilter() メソッドを使用します。詳細は、2 番目のサブセクションを参照してください。

   代わりに、フィルタ項目をハードコードできます。このページで後述する "フィルタ名および項目を定義するその他のオプション" を参照してください。

4. クエリを変更し、ユーザが選択した値を使用します。このページで後述する "MDX クエリを変更してフィルタ値を使用する" を参照してください。

9.3.1 %OnGetFilterMembers() の詳細

KPI クラスで、%OnGetFilterMembers() メソッドを実装する場合、以下のシグニチャを使用します。

```
classmethod %OnGetFilterMembers(pFilter As %String,                                    Output pMembers
As %List,                                  pSearchKey As %String = "") as %Status
```

以下は、この指定の説明です。

- pFilter は、フィルタの論理名です。
- pMembers は、$LISTBUILD リストのメンバを指定します。このリストには、論理名と表示名の両方が含まれます。詳細は、%DeepSee.KPI のクラスリファレンスを参照してください。
- pSearchKey は、ユーザが入力した検索キーです。

Tipヒン 使用するクエリのタイプと関係なく、任意の KPI でこのメソッドを使用できます。
ト

フィルタの論理名が MDX レベル識別子の場合、前述の単純な実装を使用できます。そうでない場合、メンバを取得するためにより多くの作業が必要です。例えば、このページで後述する "カスタム・ロジックを使用して実行時にフィルタ項目のリストを構築" を参照してください。

9.3.2 %GetMembersForFilter() の詳細

KPI クラスでは、以下のシグニチャを使用する %GetMembersForFilter() メソッドを使用できます。

```
classmethod %GetMembersForFilter(pCube As %String,                                    pFilterSpec As
%String,                                  Output pMembers,                                  pSearchKey
 As %String = "") as %Status
```

以下は、この指定の説明です。

- pCube は、キューブの論理名に .cube を付加した名前です。
- pFilterSpec は MDX レベル識別子です (例えば、"[DateOfSale].[Actual].[YearSold]")。
- pMembers は、出力パラメータとして返される、%OnGetFilterMembers() で必須の形式のメンバのリストです。このリストでは、フィルタ項目はメンバ・キーです。
- pSearchKey は、ユーザが入力した検索キーです。

このメソッドは、SQL ベースの KPI よりも MDX ベースの KPI で便利です。SQL ベースの KPI では、フィルタ値をクエリでの使用に適した形式に変換する必要があります。

9.4 SQL ベースの KPI でのフィルタの定義

SQL ベースの KPI でフィルタを定義するには、以下の手順を実行します。

1. フィルタのリストを指定します。各フィルタには、論理名 (必須) および表示名 (既定では論理名と同じ) があります。

   1 つのオプションは、<kpi> 要素の <filter> 要素を指定することです。前のセクションを参照してください。

   代わりに、実行時にフィルタ名を指定することもできます。このページで後述する "フィルタ名および項目を定義するその他のオプション" を参照してください。

2. 各 <filter> 要素内で、必要に応じて以下の multiSelect、searchType、および dependsOn の各属性を指定します。前のセクションを参照してください。
3. 各フィルタにフィルタ項目を定義します。各フィルタ項目には、論理名 (必須) および表示名 (既定では論理名と同じ) があります。

   1 つのオプションは、このページで前述した %OnGetFilterMembers() メソッドを実装することです。

   代わりに、フィルタ項目をハードコードできます。このページで後述する "フィルタ名および項目を定義するその他のオプション" を参照してください。

4. クエリを変更し、ユーザが選択した値を使用します。このページで後述する "SQL クエリを変更してフィルタ値を使用する" を参照してください。

9.5 フィルタ名および項目を定義するその他のオプション

このセクションでは、フィルタ名およびフィルタ項目を定義するその他のオプションについて説明します。以下のトピックについて説明します。

- 実行時にフィルタ名を指定する方法
- valueList 属性でフィルタ項目のリストをハードコードする方法
- sql 属性を使用してフィルタ項目のリストを指定する方法
- カスタム・ロジックを使用して実行時にフィルタ項目のリストを作成する方法

9.5.1 実行時のフィルタ名の指定

フィルタ名を定義する最も簡単な方法は、このページで前述したようにフィルタ名をハードコードすることです。その他のオプションは、実行時に定義することです。そのためには、KPI クラスの %OnGetFilterList() メソッドをオーバーライドします。このメソッドには、以下のシグニチャがあります。

```
classmethod %OnGetFilterList(Output pFilters As %List,pDataSourceName As %String = "") As %Status
```

pFilters は以下のノードの配列です。

- pFilters ー フィルタ数を指定します。
- pFilters(n) ー n 番目のフィルタの詳細を指定します。これは、以下の項目で構成される $LISTBUILD リストです。
  - フィルタの論理名に等しい文字列。
  - フィルタの表示名に等しい文字列。
  - フィルタ・プロパティに等しい文字列。既定値はフィルタの論理名です。
  - このフィルタで複数選択が有効かどうかを示す 1 または 0。1 を使用すると複数選択は有効、0 を使用すると無効。

pDataSourceName は、将来使用するためのものです。

例えば、以下の %OnGetFilterList() は `New Filter` という名前のフィルタを追加します。

Class Member

```
ClassMethod %OnGetFilterList(Output pFilters As %List, pDataSourceName As %String = "") As %Status
{
    set newfilter=$LB("New Filter","New Filter Display Name",,0)
    set pFilters($I(pFilters))=newfilter

}
```

別の例として、以下の %OnGetFilterList() は、ユーティリティ・メソッドを使用して、Patients キューブのすべてのレベルをフィルタとして定義します。ユーティリティ・メソッド %GetFiltersForDataSource() (**%DeepSee.Dashboard.Utils** 内) は、%OnGetFilterList() で必要な形式でフィルタのリストを返します。

Class Member

```
ClassMethod %OnGetFilterList(Output pFilters As %List, pDataSourceName As %String = "") As %Status
{
    set tSC = ##class(%DeepSee.Dashboard.Utils).%GetFiltersForDataSource("patients.cube",.tFilters)
    quit:$$$ISERR(tSC)

    set i = ""
    for {
        set i = $order(tFilters(i), 1, data)
        quit:i=""

        set pFilters($i(pFilters)) = $lb($lg(data,2), $lg(data,1),,1)
    }
    quit $$$OK
}
```

9.5.2 valueList 属性を使用したフィルタ項目のリストのハードコード

フィルタ項目のリストを構築する別の方法は、<filter> 要素の valueList 属性を指定することです。フィルタ項目の論理名のコンマで区切られたリストを使用します。このリストの順序によって、フィルタがフィルタ項目のリストを制御する順序が決まります。

以下に例を示します。

XML

```
<filter name="ZipCode"
valueList="&amp;[36711],&amp;[34577],&amp;[38928],&amp;[32006],&amp;[32007]"
displayList="36711,34577,38928,32006,32007"
/>
```

この属性は、sql 属性よりも優先されます。

この属性を指定する場合、上の例に示すように、displayList 属性も指定できます。指定する場合、表示名のコンマで区切られたリストを使用する必要があります。この属性を指定しない場合、論理名が表示名としても使用されます。

9.5.3 sql 属性を使用したフィルタ項目のリストの取得

フィルタ項目のリストを構築する別の方法は、<filter> 要素の sql 属性を指定することです。指定した場合、これは SQL クエリにする必要があります。このクエリは、1 列または 2 列を返すことができます。返されるデータセットの最初の列は、フィルタ項目の論理名を提供する必要があります。2 番目の列を含める場合、これは対応する表示名を提供します。2 番目の列を含まない場合、論理名が表示名としても使用されます。

以下に例を示します。

XML

```
<filter name="ZipCode1" sql="SELECT DISTINCT PostalCode FROM BI_Study.City"/>
```

sql 属性を指定する場合は、displayList 属性および valueList 属性は指定しないでください (前のサブセクションを参照してください)。

9.5.4 カスタム・ロジックを使用して実行時にフィルタ項目のリストを構築

フィルタ項目のリストを構築する別の方法は、%OnGetFilterMembers() および独自のロジックを実装し、リストを作成することです。以下に例を示します。

Class Member

```
ClassMethod %OnGetFilterMembers(pFilter As %String, Output pMembers As %List, pSearchKey As %String =
 "") As %Status
{
    Set status = $$$OK

    Try {
        If (pFilter = "AgeGroup") {
                set pFilterSpec="[AgeD].[h1].[Age Group]"
        } Elseif (pFilter="Gender") {
                set pFilterSpec="[GenD].[h1].[Gender]"
        } Elseif (pFilter="ZipCode") {
                set pFilterSpec="[HomeD].[h1].[ZIP]"
        }
        do ##class(%DeepSee.KPI).%GetMembersForFilter("Patients",pFilterSpec,.pMembers,pSearchKey)
    }
    Catch(ex) {
        Set status = ex.AsStatus()
    }

    Quit status
}
```

別の方法として、レベル・メンバを保持するディメンジョン・テーブルでクエリを実行することができます。(ディメンジョン・テーブルの詳細は、"ファクト・テーブルおよびディメンジョン・テーブルの詳細"を参照してください。)以下に例を示します。

Class Member

```
ClassMethod %OnGetFilterMembers(pFilter As %String, Output pMembers As %List, pSearchKey As %String =
 "") As %Status
{
  set status = $$$OK

  try {
    if (pFilter = "AgeGroup") {
      //get values from level table
      set sql = "SELECT DISTINCT DxAgeGroup FROM BI_Model_PatientsCube.DxAgeGroup"
      set stmt = ##class(%SQL.Statement).%New()

      set status = stmt.%Prepare(sql)
      if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

      set rs = stmt.%Execute()
      if (rs.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rs.%SQLCODE, ": ", rs.%Message
quit}

      while(rs.%Next()) {
        set display=rs.DxAgeGroup
        set actual="&["_display_"]"
        set pMembers($I(pMembers)) = $LB(display,actual)
      }
      if (rs.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rs.%SQLCODE, ": ", rs.%Message quit}

    }
  }
  catch(ex) {
    set status = ex.AsStatus()
  }

  quit status
}
```

この例では pSearchKey 引数を使用しません。

9.6 MDX クエリを変更してフィルタ値を使用する

MDX ベースの KPI では、クエリを変更してフィルタを使用するには、%OnGetMDX() メソッドを実装します。スタジオで KPI ウィザードを使用して、ソース・タイプとして MDX を選択する場合、生成される KPI クラスには、このメソッドの以下のスタブ定義が含まれます。これを開始点として使用できます。

Class Member

```
/// Return an MDX statement to execute.
Method %OnGetMDX(ByRef pMDX As %String) As %Status
{
    Quit $$$OK
}
```

変数 pMDX には、<kpi> の mdx 属性で指定される MDX クエリが格納されます(この属性を指定しなかった場合、pMDX は Null になります)。

このメソッドを実装する場合、以下の手順を実行します。

1. 初期クエリがない場合、変数 pMDX の初期値を指定します。これは MDX SELECT クエリです。

   ″基本的な KPI の定義″ で説明したように、mdx 属性でクエリを指定しない場合、初期クエリはありません。

2. 該当するフィルタそれぞれを調べ、その値を取得します。

   このためには、最初のサブセクションで説明したように、KPI インスタンスの %filterValues プロパティを使用します。

3. 各フィルタ値を解析し、必要に応じて、MDX %FILTER 節で使用できる形式に変換します (これがフィルタを使用する方法の場合)。適切な形式をリストするテーブルは、2 番目のサブセクションを参照してください。

4. ニーズに合わせて、変数 pMDX を変更して、フィルタ値を使用します。最も一般的な状況では、以下のように NULL でない各フィルタに %FILTER 節を追加します。

   ```
   %FILTER mdx_expression
   ```

   ここで、mdx_expression は、前の手順で作成した、そのフィルタの式です。

   または、式を結合する MDX タプル式を作成し、以下のように 1 つの %FILTER 節を追加します。

   ```
   %FILTER tuple_expression
   ```

   一般的ではない状況では、フィルタ値を使用して、別の方法でクエリ文字列を書き換えることができます。

5. $$$OK (または 1) を返します。

最後のサブセクションに例を示します。

9.6.1 フィルタ値へのアクセス

ユーザがフィルタ項目を選択すると、KPI インスタンスの %filterValues プロパティが設定されます。このプロパティは、システムによって以下のように作成されたオブジェクト (具体的には %ZEN.proxyObject のインスタンス) です。

- KPI のフィルタごとに、フィルタと同じ名前 (大文字と小文字が区別されます) のプロパティが 1 つあります。

  例えば、KPI に filter1、filter2、および filter3 という名前のフィルタがある場合、KPI インスタンスの %filterValues プロパティには filter1、filter2、および filter3 という名前のプロパティがあります。

- %filterValues のプロパティには、以下のテーブルに示す値が含まれます。

ユーザがフィルタで選択する対象	%filterValues.FilterName の値	留意事項
なし	null	
1 つの項目	item	
複数の項目	{item1, item2}	複数選択がオフの場合には該当なし
1 つの項目と [除外] オプション	%NOT item	複数選択がオフの場合には該当なし
複数の項目と [除外] オプション	%NOT {item1, item2}	複数選択がオフの場合には該当なし
範囲	item1 : item2	ピボット・テーブルで定義されるフィルタに対してのみ適用され、KPI に影響します

9.6.2 %FILTER で使用するためにフィルタ値を MDX 式に変換

以下のテーブルは、ユーザの選択に応じて、%FILTER 節での使用に適切な MDX 式をリストしています。

ユーザがフィルタで選択する対象	該当する MDX %FILTER 式
なし	none (この場合、%FILTER 節を適用しません)
1 つの項目	[dimension].[hierarchy].[level].[member]
1 つの項目と [除外] オプション	[dimension].[hierarchy].[level].[member].%NOT
item1 および item2	{item1,item2,...} (各 item の形式は [dimension].[hierarchy].[level].[member])
複数の項目と [除外] オプション	EXCEPT([dimension].[hierarchy].[level].MEMBERS,{item,item,item,...}) (各 item の形式は [dimension].[hierarchy].[level].[member])
範囲	[dimension].[hierarchy].[level].[member]:[member]

使用例は、サンプル・クラス BI.Utils.MDXAutoFiltersKPI のメソッド BuildString() を参照してください。

複数選択が無効になっている場合、EXCLUDE も無効になり、フィルタ値の使用できる形式が減ることに注意してください。この場合、クエリ構造は比較的単純です。

9.6.3 例

%OnGetMDX() の以下の実装は、複数選択が無効になっている KPI に適用されます。

Class Member

```
Method %OnGetMDX(ByRef pMDX As %String) As %Status
{
 if (..%filterValues."[aged].[h1].[age group]"'="")
 {
    set pMDX = pMDX _ " %FILTER [aged].[h1].[age group]." _..%filterValues."[aged].[h1].[age group]"
 }

 if (..%filterValues."[gend].[h1].[gender]"'="")
 {
    set pMDX = pMDX _ " %FILTER [gend].[h1].[gender]." _..%filterValues."[gend].[h1].[gender]"
 }

 if (..%filterValues."[homed].[h1].[zip]"'="")
 {
    set pMDX = pMDX _ " %FILTER [homed].[h1].[zip]." _..%filterValues."[homed].[h1].[zip]"
 }

 quit $$$OK
}
```

このメソッドは、区切りプロパティ名を使用しているため、読みにくい場合があります。メモ：

- `"[aged].[h1].[age group]"` は、有効なプロパティ名です。
- `..%filterValues."[aged].[h1].[age group]"` は、KPI インスタンスの %filterValues プロパティの `"[aged].[h1].[age group]"` プロパティへの参照です。

9.7 SQL クエリを変更してフィルタ値を使用する

SQL ベースの KPI では、クエリを変更してフィルタを使用するには、%OnGetSQL() メソッドを実装します。スタジオで KPI ウィザードを使用して、ソース・タイプとして SQL を選択する場合、生成される KPI クラスには、このメソッドの以下のスタブ定義が含まれます。これを開始点として使用できます。

Class Member

```
/// Return an SQL statement to execute.
Method %OnGetSQL(ByRef pSQL As %String) As %Status
{
    Quit $$$OK
}
```

変数 pSQL には、`<kpi>` の `sql` 属性で指定される SQL クエリが格納されますこの属性を使用していない場合、pSQL は Null です。

このメソッドを実装する場合、以下の手順を実行します。

1. 初期クエリがない場合、変数 pSQL の初期値を指定します。これは SQL SELECT クエリです。

   ″基本的な KPI の定義″ で説明したように、`sql` 属性でクエリを指定しない場合、初期クエリはありません。

2. 該当するフィルタそれぞれを調べ、その値を取得します。このオプションは以下のとおりです。
   - %GetSQLForFilter() メソッドを使用します。これは、SQL クエリの WHERE 節で使用する便利な形式で値を返します。最初のサブセクションを参照してください。
   - 前のセクションで説明したように、KPI インスタンスの %filterValues プロパティを使用します。

3. ニーズに合わせて、変数 pSQL を変更して、フィルタ値を使用します。最も一般的な状況では、クエリを変更して SQL WHERE 節を組み込みます。

   一般的ではない状況では、フィルタ値を使用して、別の方法でクエリ文字列を書き換えることができます。

2 番目および 3 番目のサブセクションに例を示します。

4. $$$OK (または 1) を返します。

注釈　SQL ベースの KPI は 1000 行を超えることはできません。返される行数は自動的に制限されます。

9.7.1 %GetSQLForFilter()

SQL ベースの KPI では、クエリに組み込むのに便利な形式でフィルタ値にアクセスするメソッドを使用できます。

```
method %GetSQLForFilter(sql_field_reference,filter_name) As %String
```

現在のフィルタ選択を調べて、SQL クエリの WHERE 節で使用できる文字列を返します。

sql_field_expression は SQL フィールド名で、矢印構文を含むことができます。filter_name は、この KPI で定義されるフィルタの名前です。

例えば、以下のメソッド呼び出しを考えてみます。

```
..%GetSQLForFilter("City->Name","City")
```

以下のテーブルに、さまざまなシナリオでこのメソッド呼び出しによって返される値を示します。

シナリオ	メソッドによって返される値
ユーザが PINE を選択する	City->Name = 'PINE'
ユーザが MAGNOLIA で始まり PINE で終わる範囲を選択する	City->Name = ('MAGNOLIA':''PINE')
ユーザが MAGNOLIA と PINE を選択する	City->Name IN ("MAGNOLIA','PINE')
ユーザが PINE を選択して [除外] を選択する	City->Name <> 'PINE'
ユーザが MAGNOLIA と PINE を選択して [除外] を選択する	City->Name NOT IN ('MAGNOLIA','PINE')

9.7.2 SQL KPI の例 1

以下の例は、サンプル・クラス BI.Model.KPIs.DemoSQL のものです。この場合、フィルタは、GROUP BY 節と ORDER BY 節を SQL クエリに追加します。

Class Member

```
Method %OnGetSQL(ByRef pSQL As %String) As %Status
{
    //this is the start of the SQL query for this KPI
    Set pSQL = "SELECT Count(*),AVG(Age) FROM BI_Study.Patient "

    Set where = ""
    //look at %filterValues to see if a filter has been applied to this KPI instance
    If $IsObject(..%filterValues) {
        If (..%filterValues.ZipCode'="")
        {
            // Call utility method that returns filter data in convenient format
            Set sqlstring=..%GetSQLForFilter("HomeCity->PostalCode","ZipCode")
            Set where = "WHERE "_sqlstring

        }
    }

    Set groupby="GROUP BY HomeCity "
    Set orderby="ORDER BY HomeCity "
```

```
    // assemble the SQL statement
    Set pSQL=pSQL_where_groupby_orderby
        Quit $$$OK
}
```

9.7.3 SQL KPI の例 2

以下の例は、サンプル・クラス HoleFoods.KPISQL のものです。

Class Member

```
Method %OnGetSQL(ByRef pSQL As %String) As %Status
{
 If $IsObject(..%filterValues) {
   Set tWHERE = ""
   If (..%filterValues.City'="") {
      Set tWHERE = tWHERE _ $S(tWHERE="":"",1:" AND ") _ " Outlet->City = '" _ ..%filterValues.City
_"'"
   }
   If (..%filterValues.Product'="") {
     Set tWHERE = tWHERE _ $S(tWHERE="":"",1:" AND ") _ " Product = '" _ ..%filterValues.Product _"'"
     }

   If (tWHERE'="") {
     // insert WHERE clase within query
     Set tSQL1 = $P(pSQL,"GROUP BY",1)
     Set tSQL2 = $P(pSQL,"GROUP BY",2)
     Set pSQL = tSQL1 _ " WHERE " _ tWHERE
     If (tSQL2 '= "") {
       Set pSQL = pSQL _ " GROUP BY" _ tSQL2
     }
    }
 }
 Quit $$$OK
}
```

この場合、KPI は、"基本的な KPI の定義" で説明したように、sql 属性内に初期クエリを定義します。%OnGetSQL() メソッドは、そのクエリを変更します。

9.8 その他の MDX KPI の例

このセクションでは、その他の MDX KPI の例について説明します。

9.8.1 DemoMDXAutoFilters KPI

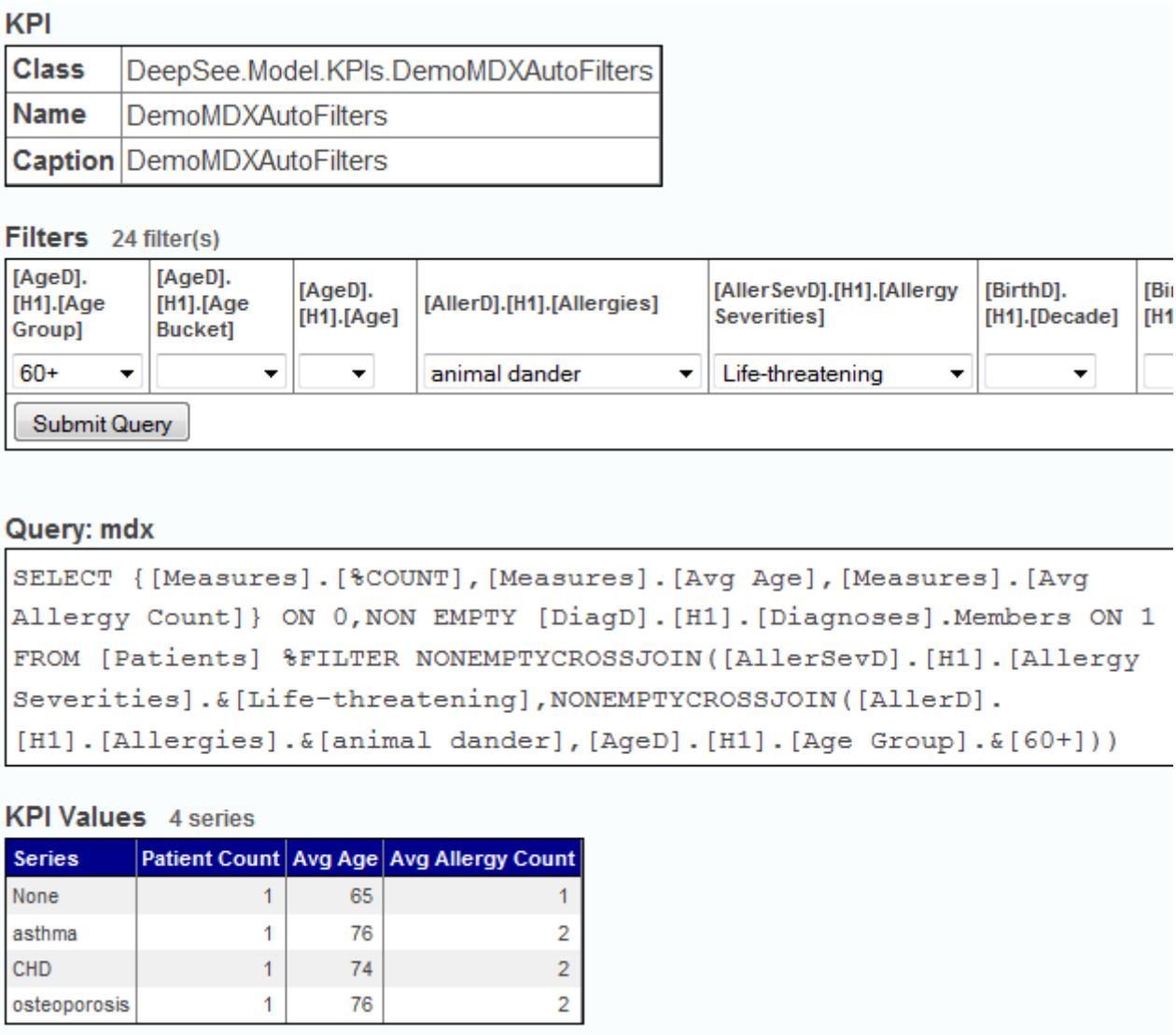
サンプル・クラス BI.Model.KPIs.DemoMDXAutoFilters の KPI は単純ですが、特別なスーパークラスを使用します。

Class Definition

```
Class BI.Model.KPIs.DemoMDXAutoFilters Extends BI.Utils.MDXAutoFiltersKPI
{

Parameter CUBE = "PATIENTS";

Parameter DOMAIN = "PATIENTSAMPLE";

XData KPI [ XMLNamespace = "http://www.intersystems.com/deepsee/kpi" ]
{
<kpi name="DemoMDXAutoFilters" displayName="DemoMDXAutoFilters"
sourceType="mdx"
mdx="SELECT {[Measures].[%COUNT],[Measures].[Avg Age],[Measures].[Avg Allergy Count]} ON 0,
NON EMPTY [DiagD].[H1].[Diagnoses].Members ON 1 FROM [Patients]">

<property name="Patient Count" displayName="Patient Count" columnNo="1" />
<property name="Avg Age" displayName="Avg Age" columnNo="2" />
<property name="Avg Allergy Count" displayName="Avg Allergy Count" columnNo="3" />

</kpi>
```

```
}
}
```

このクラスは、直接フィルタを定義することも、直接フィルタ・メンバを定義することもなく、ハードコードされた MDX クエリのみを定義していることに注意してください。ただし、この KPI のテスト・ページを表示した場合、Patients キューブのすべてのレベルをフィルタとして使用でき、KPI は適切な %WHERE 節をクエリに追加します。以下に例を示します。



KPI

Class	DeepSee.Model.KPIs.DemoMDXAutoFilters
Name	DemoMDXAutoFilters
Caption	DemoMDXAutoFilters

Filters 24 filter(s)

[AgeD].[H1].[Age Group]	[AgeD].[H1].[Age Bucket]	[AgeD].[H1].[Age]	[AllerD].[H1].[Allergies]	[AllerSevD].[H1].[Allergy Severities]	[BirthD].[H1].[Decade]	[Bi [H1
60+			animal dander	Life-threatening		

Submit Query

Query: mdx

```
SELECT {[Measures].[%COUNT],[Measures].[Avg Age],[Measures].[Avg
Allergy Count]} ON 0,NON EMPTY [DiagD].[H1].[Diagnoses].Members ON 1
FROM [Patients] %FILTER NONEMPTYCROSSJOIN([AllerSevD].[H1].[Allergy
Severities].&[Life-threatening],NONEMPTYCROSSJOIN([AllerD].
[H1].[Allergies].&[animal dander],[AgeD].[H1].[Age Group].&[60+]))
```

KPI Values 4 series

Series	Patient Count	Avg Age	Avg Allergy Count
None	1	65	1
asthma	1	76	2
CHD	1	74	2
osteoporosis	1	76	2

この KPI は、サンプル **BI.Utils.MDXAutoFiltersKPI** を拡張します。これは、フィルタを定義し、クエリを書き換えます。

このクラスには、以下のメソッドが存在します。

- %OnGetFilterList() は、CUBE クラス・パラメータで指定されたキューブに定義されているすべてのレベルを取得します。
- %OnGetFilterMembers() が実装されています。各レベルで、ピボット・テーブルで要求される形式で、レベル・メンバを取得します。このページで前述した "ピボット・テーブルのフィルタ構文" を参照してください。
- インスタンス・メソッド FilterBuilder() は、キューブ・ベースのフィルタを繰り返し処理して、それぞれの現在の値を取得し、それらを結合して MDX %FILTER 節としての使用に適した 1 つの文字列を生成します。
- %OnGetMDX() は、ハードコードされたクエリに %FILTER 節を追加します。

9.8.2 DemoInteroperability KPI

サンプル・クラス BI.Model.KPIs.DemoInteroperability の KPI は、前述の例より複雑です。このクラス内で、%OnGetFilterList() は、CUBE クラス・パラメータで指定されたキューブに定義されているすべてのレベルを取得します。次に、Yaxis という名前の別のフィルタを追加します。

Class Member

```
ClassMethod %OnGetFilterList(ByRef pFilters As %List, pDataSourceName As %String = "") As %Status
{
    //call method in superclass so we can get filters of the associated cube
    set tSC=##super(.pFilters,pDataSourceName)
    quit:$$$ISERR(tSC) tSC

    //update pFilters array to include the custom filter
    set pFilters($i(pFilters)) = $lb("Yaxis","Yaxis",,0)

   quit $$$OK
}
```

%OnGetFilterMembers() が実装されています。フィルタ Yaxis の場合、このメソッドはメンバ・セットを提供します。他のフィルタの場合、ピボット・テーブルで要求される形式で、レベル・メンバを取得します。このページで前述した "ピボット・テーブルのフィルタ構文" を参照してください。このメソッドは、以下のとおりです。

Class Member

```
ClassMethod %OnGetFilterMembers(pFilter As %String, Output pMembers As %List, pSearchKey As %String =
 "",
pDataSourceName As %String = "") As %Status
{
    set pMembers=""
    if (pFilter="Yaxis") {
        set pMembers($I(pMembers))=$LB("Home City","[homed].[h1].[city]")
        set pMembers($I(pMembers))=$LB("Favorite Color","[colord].[h1].[favorite color]")
        set pMembers($I(pMembers))=$LB("Profession","[profd].[h1].[profession]")
        set pMembers($I(pMembers))=$LB("Diagnoses","[diagd].[h1].[diagnoses]")
    } else {
        //call method in superclass so we can get filter members for the associated cube
         do ..%GetMembersForFilter(..#CUBE,pFilter,.pMembers)
 }
   quit $$$OK
}
```

最後に、%OnGetMDX() は MDX クエリを構築します。Yaxis フィルタは、行に使用するレベルを決定します。このメソッドは、%FILTER 節をクエリに追加します。%FILTER 節は、前の例と同様に、キューブ・ベースの任意のフィルタを使用します。

Class Member

```
Method %OnGetMDX(ByRef pMDX As %String) As %Status
{
 set yaxis=", NON EMPTY [profd].[h1].[profession].MEMBERS ON 1"
 //check custom filter value
 if (..%filterValues."Yaxis"'="") {
    set yaxis=", NON EMPTY "_..%filterValues.Yaxis_".MEMBERS ON 1"
 }
 set pMDX="SELECT {MEASURES.[%COUNT],MEASURES.[avg age]} on 0"_yaxis_" FROM "_..#CUBE

 /// append a %FILTER clause to handle any other filter values
 Set pMDX = pMDX _ ..FilterBuilder()
 Quit $$$OK
}
```

9.9 KPI のリストの定義

リスト・オプションを含むように KPI を定義できます。この場合、KPI にフィルタも含まれる場合、リスト定義でフィルタ選択を考慮する必要があります。

リストを定義するために、KPI クラスに %OnGetListingSQL() メソッドを実装します。このメソッドには、以下のシグニチャがあります。

```
ClassMethod %OnGetListingSQL(ByRef pFilters As %String,                    ByRef pSelection
 As %String) As %String
```

このメソッドは、リスト・クエリのテキストを返します。引数は以下のとおりです。

- pFilters は、現在のフィルタ値を格納する多次元配列です。この配列には以下のノードがあります。

ノード	ノード値
pFilters(filter_name)。filter_name は、この KPI で定義されるフィルタの名前です。	このフィルタの現在の値

  詳細は、"KPI フィルタの定義" を参照してください。

- pSelection は、現在の選択に関する情報を格納する多次元配列です。この配列には以下のノードがあります。

ノード	ノード値
pSelection("selectedRange")	ピボットで形式 "startRow,startCol,endRow,endCol" (1-based) の文字列として現在選択されているセル。
pSelection("rowValues")	選択された行の値のコンマ区切りのリスト。これらの値では、コンマが円記号 (\) として表示されます。KPI のプロパティでこの値が構成されていない場合は、このノードには代わりに系列名が含まれます。
pSelection("sortColumn")	リストの並べ替えに使用する列の番号を指定します。0 を使用すると並べ替えは行われません。
pSelection("sortDir")	並べ替えの方向を "ASC" または "DESC" で指定します。

このメソッドは、SQL SELECT クエリを返す必要があります。このクエリでは、他のリストと同様に、矢印構文および SQL 関数も使用できます。

または、%OnGetListingResultSet() メソッドをオーバーライドできます。オーバーライドする場合は、結果セットを作成して実行する必要があります。

9.9.1 例

以下の例は、HoleFoods.KPISQL からの抜粋です。

Class Member

```
ClassMethod %OnGetListingSQL(ByRef pFilters As %String, ByRef pSelection As %String) As %String
{
    Set tSQL = "SELECT TOP 1000 %ID,DateOfSale,Product FROM HoleFoods.SalesTransaction"

    // apply sorting, if asked for
    If (+$G(pSelection("sortColumn"))>0) {
        Set tSQL = tSQL _ " ORDER BY " _ pSelection("sortColumn") _ " " _ $G(pSelection("sortDir"))
    }

    Quit tSQL
}
```

10

高度な KPI の定義

ここでは、高度な機能を使用する Business Intelligence KPI を実装する方法を説明します。

"基本的な KPI の定義" および "フィルタおよびリストを含む KPI の定義" も参照してください。

"BI サンプルのアクセス方法" も参照してください。

10.1 手動 KPI の定義

任意の KPI は、**%DeepSee.KPI** のサブクラスのインスタンスです。手動 KPI では、コールバック・メソッドがそのインスタンスのプロパティを設定します。このセクションでは、以下の項目について説明します。

- KPI インスタンスの利用できるプロパティ
- KPI のプロパティをオーバーライドする方法
- 手動クエリを定義する方法

10.1.1 利用できるプロパティ

KPI インスタンスのコールバック・メソッドで、以下のプロパティを利用できます。

- **%seriesCount** ― この KPI の系列 (行) 数を指定します。
- **%seriesNames(n)** ― 系列 n の名前を指定します。n は整数値です。
- **%data(n,propname)** ― 系列 n に対して、特定のプロパティ (propname) の値を指定します。
- **%rangeLower** ― 範囲の下限値を指定します。この値により、この KPI がメータで表示されている場合の既定の下限インジケータが設定されます。
- **%rangeUpper** ― 範囲の上限値を指定します。この値により、この KPI がメータで表示されている場合の既定の上限インジケータが設定されます。
- **%thresholdLower** ― 下限しきい値を指定します。この値により、この KPI がメータで表示されている場合の既定の下限しきい値インジケータが設定されます。
- **%thresholdUpper** ― 上限しきい値を指定します。この値により、この KPI がメータで表示されている場合の既定の上限しきい値インジケータが設定されます。
- **%filterValues** ― 任意のフィルタの値が含まれます。詳細は、"フィルタおよびリストを含む KPI の定義" を参照してください。

10.1.2 KPI プロパティのオーバーライド

%OnLoadKPI() コールバックにより、KPI オブジェクト・インスタンスのプロパティを表示前にオーバーライドできます。これを使用すると、実行時に範囲としきい値を指定できます。このコールバックには、以下のシグニチャがあります。

```
Method %OnLoadKPI() As %Status
```

これらのプロパティを、KPI クラスの他のメソッド内で設定することもできます。

10.1.2.1 例

以下の例は、HoleFoods.KPISalesVsTarget からの抜粋です。

```
Method %OnLoadKPI() As %Status
{
    Set tSC = $$$OK

    // Compute additional values
    Set tFilters = ..%filterValues

    // compute recent history using query
    If ((tFilters.Year'="")&&(tFilters.Year'="*")) {
        // Take &[] off of Year value!
        Set tStartMonth = "Jan-"_$E(tFilters.Year,3,6)
        Set tEndMonth = "Dec-"_$E(tFilters.Year,3,6)
    }
    Else {
        Set tStartMonth = "NOW-12"
        Set tEndMonth = "NOW"
    }

    Set tROWS = ..RowsClause
    Set tMDX = "SELECT "_tROWS_"%LIST(DateOfSale.[MonthSold].["_tStartMonth_"]:["_tEndMonth_"]) "
      _"ON COLUMNS FROM HOLEFOODS WHERE Measures.[Amount Sold] " _ ..FilterClause
    Set tRS = ##class(%DeepSee.ResultSet).%New()
    Set tSC = tRS.%PrepareMDX(tMDX)
    If $$$ISERR(tSC) Quit tSC
    Set tSC = tRS.%Execute()
    If $$$ISERR(tSC) Quit tSC

    For n = 1:1:..%seriesCount {
        Set tValue = tRS.%GetOrdinalValue(1,n)
        Set ..%data(n,"History") = tValue
    }
    Quit tSC
}
```

このメソッドは、この KPI の `History` プロパティに値を入力します。このプロパティは、製品ごとの過去の月別の売上を並べたコンマ区切りリストです。

10.1.3 手動クエリの定義

手動 (カスタム) クエリを KPI の基にするには、以下の手順を実行します。

- `<kpi>` 要素内で、`sourceType="manual"` を指定します。
- KPI クラスの %OnExecute() コールバック・メソッドをオーバーライドします。このメソッドには、以下のシグニチャがあります。

  ```
  method %OnExecute() as %Status
  ```

このメソッドで、必要に応じた任意のロジックを使用してクエリを定義します。次に %seriesCount、%seriesNames、および %data の各プロパティを設定します。

10.1.3.1 例

以下は、ハードコードされた値の単純な例を示しています。

Class Member

```
Method %OnExecute() As %Status
{
    Set ..%seriesCount=3
    Set ..%seriesNames(1)="alpha"
    Set ..%seriesNames(2)="beta"
    Set ..%seriesNames(3)="gamma"
    Set ..%data(1,"property1")=123
    Set ..%data(1,"property2")=100000
    Set ..%data(1,"property3")=1.234
    Set ..%data(2,"property1")=456
    Set ..%data(2,"property2")=200000
    Set ..%data(2,"property3")=2.456
    Set ..%data(3,"property1")=789
    Set ..%data(3,"property2")=300000
    Set ..%data(3,"property3")=3.789
    Quit $$$OK
}
```

10.2 キャッシュ可能な KPI の定義

既定では、MDX クエリを使用する KPI は、(他のすべての MDX クエリと一緒に) キャッシュされます。このキャッシュは、目的に対して十分に最新である場合と最新ではない場合があります。すなわち、このセクションで説明するように、KPI を明示的にキャッシュすることもできます。

既定では、非 MDX KPI はキャッシュされません。

KPI を変更して結果がキャッシュされるようにするには、以下の変更を KPI クラスに適用します。

- CACHEABLE クラス・パラメータを 1 に指定します。
- %OnComputeKPICacheKey() メソッドを実装します。

  ```
  Method %OnComputeKPICacheKey(Output pCacheKey As %String,
                               pQueryText As %String = "") As %Status
  ```

  pQueryText は KPI クエリのテキスト、pCacheKey はキャッシュされた結果に関連付けられる一意のキーです。通常、これはクエリ・テキストのハッシュ化した値です。
- %OnComputeKPITimestamp() メソッドを実装します。

  ```
  Method %OnComputeKPITimestamp(ByRef pTimestamp As %String,
                                pSourceType As %String,
                                pQueryText As %String = "") As %Status
  ```

  pSourceType はクエリ・タイプを示す文字列 (`"mdx"`、`"sql"`、または `"manual"`)、pQueryText は KPI クエリのテキスト、pTimestamp は KPI のタイムスタンプです。

  %OnComputeKPITimestamp() が、指定された KPI に対して、指定されたキーの KPI キャッシュに格納されている同じタイムスタンプを返す場合、システムはキャッシュされた値を使用します。それ以外の場合、システムは KPI を返します。

  既定では、%OnComputeKPITimestamp() は分単位の精度のタイムスタンプを返します。これは、既定では、キャッシュは (最大) 1 分間保持されることを意味します。

指定された KPI のキャッシュをクリアするには、その %ClearKPICache() メソッドを呼び出します。

FORCECOMPUTE パラメータを指定すると、CACHEABLE が設定されている場合でも、KPI がキャッシュされないことに注意してください。

10.3 非同期 KPI の定義

プラグインを除けば、KPI は同期的に実行されます。

KPI を変更して非同期に実行されるようにするには、以下の変更を KPI クラスに適用します。

- ASYNC クラス・パラメータに 1 を指定します。
- さらに、KPI を変更して、その結果がキャッシュされるようにします。前のセクションを参照してください。

  これは、システムに結果を格納する場所を用意するために必要です。
- %OnCompute() 内で、必要に応じて %SetPercentComplete() を呼び出して、処理状態を通知します。詳細は、"完了状態の通知" を参照してください。

11
プラグインの定義

ここでは、プラグインを定義する方法を説明します。プラグインは、特殊な形式の Business Intelligence KPI です。

注釈　　このページを読む前に、必ず KPI に関する他のすべての項目を読んでください。

プラグインとその他の種類のモデル要素との比較については、"モデル・オプションの概要" を参照してください。

"BI サンプルのアクセス方法" も参照してください。

11.1 はじめに

プラグインは、アナライザおよびクエリで使用する 1 つ以上の計算を定義するクラスです。プラグインには、以下の特徴があります。

- プラグイン・インスタンスは、指定されたコンテキストで最下位レベルのデータにアクセスできます。
- パラメータを使用できます。
- 実行は非同期に行われます。プラグインをピボット・テーブルで使用している場合は、保留中のセルでプラグインの現在の状態を (文字列 n% `complete` として) 表示できます。

  ピボット・テーブルは、結果が生成されると、自動的に更新されます。
- プラグインが返す値はキャッシュされます。

プラグインは、複雑な計算または時間がかかる計算に特に適しています。例えば、ソース・レコードの複数の異なる部分や外部情報を使用する計算をする場合に、プラグインは適しています。

11.1.1 プラグインの使用方法

プラグイン・クラスに応じて、以下の方法のいずれかまたはすべてでプラグインを使用できます。

- MDX %KPI 関数で使用します (この場合は、パラメータに値を指定することもできます)。これは、どのような場合でも可能です。

  つまり、どのような場合でも、プラグインを使用する計算メンバを定義できます (計算メンバの定義の詳細は、"計算メンバの定義" を参照してください)。
- アナライザおよびウィジェットで直接使用する。これは、PLUGINTYPE クラス・パラメータが `"Pivot"` で、PUBLIC クラス・パラメータが 1 (既定値) である場合に可能です。

アナライザまたはウィジェットで直接使用できないプラグインを作成するには、PLUGINTYPE を "Aggregate" と指定します。または、PUBLIC を 0 と指定します。

11.1.2 使用可能なプラグイン・クラス

%DeepSee.PlugIn パッケージには、計算メジャーで使用できるいくつかのプラグイン・クラスが用意されています。これらのクラスは以下のとおりです。

- %DeepSee.PlugIn.Distinct ― 指定されたセルの指定されたレベルの個別値の数を取得します。

  %DeepSee.PlugIn.Median ― セルで使用される最下位レベルのすべてのレコード間で、指定されたメジャーの中央値を取得します。

- %DeepSee.PlugIn.Percentile ― セルで使用される最下位レベルのすべてのレコード間で、指定されたメジャーのパーセント値を取得します。

  これらのプラグイン・クラスは、PLUGINTYPE が "Aggregate" と定義されているため、アナライザまたはウィジェットで直接使用することはできません。これらの詳細は、リファレンス "MDX 関数" に記載の "%KPI" を参照してください。

- その他のクラス ― Text Analytics で使用するためのより高度なプラグイン。これらのプラグインは、アナライザの [ピボット分析] 画面で使用されます。

もう 1 つのサンプル・プラグイン・クラスは BI.Model.KPIs.PluginDemo です。このプラグイン・クラスは、PLUGINTYPE が "Pivot" と定義されているため、直接使用できます。

11.1.3 プラグインを例示するサンプル

BI サンプルで、KPIs & Plug-ins フォルダ内のダッシュボードを参照してください。

- HoleFoods Plug-ins ダッシュボードでは、HoleFoods キューブに定義されている計算メジャー Median Revenue および 90th Percentile Revenue を使用します。これらのメジャーは、%KPI 関数を使用してサンプル・プラグイン・クラス %DeepSee.PlugIn.Median および %DeepSee.PlugIn.Percentile から値を取得します。

- Patients Plug-ins ダッシュボードには、計算メジャー Median Test Score および 90th Percentile Test Score を使用するピボット・テーブルがあります。これらの計算メンバは、前の箇条書きのメンバと同様に Patients キューブに定義されています。

  このダッシュボードには、もう 1 つのピボット・テーブルが含まれています。このピボット・テーブルは、サンプル・クラス BI.Model.KPIs.PluginDemo で定義されているプラグインを直接使用します。

11.2 単純なプラグインの要件

単純なプラグインを定義するには、以下のようにクラスを作成します。

- %DeepSee.KPIPlugIn をスーパークラスとして使用します。

- 少なくとも 1 つのプロパティを指定する KPI という名前の XData ブロックを定義します。以下に例を示します。

Class Member

```
XData KPI [ XMLNamespace = "http://www.intersystems.com/deepsee/kpi" ]
{
<kpi name="PluginDemo" displayName="PluginDemo" caption="PluginDemo" >

<property name="PatientCount" displayName="PatientCount" />
<property name="HighScoreCount" displayName="HighScoreCount" />

</kpi>
}
```

他の KPI と同様に、フィルタも追加できます。

詳細は、"KPI クラスおよびプラグイン・クラスのリファレンス情報" を参照してください。

- BASECUBE クラス・パラメータを指定します。単純なプラグインの場合、1 つのキューブまたはサブジェクト領域の論理名を指定します (ただし、このページで後述する "複数のキューブで使用するプラグインの作成" も参照してください)。
- 使用するベース MDX クエリを指定します。`<kpi>` の `mdx` 属性を指定するか、または %OnGetMDX() メソッドを以下に示す一般的な方法で実装します。

```
Method %OnGetMDX(ByRef pMDX As %String) As %Status
{
    Set pMDX = "SELECT FROM "_..#BASECUBE
    Quit $$$OK
}
```

コンテキスト情報 (行、列、およびフィルタ) が自動的にこのベース・クエリに適用されます。

- %OnCompute メソッドで使用できるようにする必要があるフィールドを指定します。キューブのソース・テーブルに含まれるフィールドか、ファクト・テーブルに含まれるフィールドを指定できます。ハードコードされたリストを指定することも、フィールドのリストを定義するコールバックを使用することもできます。

  これらのフィールドを指定するには、以下の操作を実行します。

  - 使用するフィールドがファクト・テーブルにある場合は、LISTINGSOURCE クラス・パラメータに `"FactTable"` を指定します (ファクト・テーブルの詳細は、"ファクト・テーブルおよびディメンジョン・テーブルの詳細" を参照してください)。

    このパラメータを省略した場合や `"SourceTable"` を指定した場合、プラグインは指定されたキューブのソース・テーブルを照会します。

  - フィールド名のハードコードされたリストを使用する場合は、LISTINGFIELDS クラス・パラメータを指定します。使用するフィールドのコンマ区切りリストを指定します。

    以下に例を示します。

    Class Member

    ```
    Parameter LISTINGFIELDS = "Field1, Field2, Field3";
    ```

    フィールドにはエイリアスを指定できます。以下に例を示します。

    Class Member

    ```
    Parameter LISTINGFIELDS = "Field1, Field2 as FieldAlias, Field3";
    ```

    他のリストと同様に、矢印構文および SQL 関数も使用できます。

    矢印構文を使用する場合は、フィールドのエイリアスを必ず指定してください。

  - フィールドのリストをプログラムで定義する場合は、%OnGetListingFields() メソッドを実装します。例えば、以下のメソッドにより、プラグインは 1 つのフィールドを取得します。

Class Member

```
Method %OnGetListingFields() As %String
{
    //could use an API to get the field name, but in this case factName is set
    //so the field name is known
    Set tListingFields = "MxTestScore"
    Quit tListingFields
}
```

詳細は、"KPI のリストの定義" を参照してください。

注釈　プラグインの場合、LISTINGFIELDS パラメータと %OnGetListingFields() では、詳細リストまたはリスト・フィールドが定義されません。これらのオプションは、%OnCompute() メソッドで使用できるフィールドのみを定義します。

- %OnCompute() メソッドを実装します。このタスクの詳細は、この後のセクションを参照してください。
- 必要に応じて、PLUGINTYPE と PUBLIC の各クラス・パラメータを指定します。"プラグインの使用方法" を参照してください。

11.3 %OnCompute() の実装

プラグインで使用するピボット・テーブルのセルごとに、プラグインは LISTINGSOURCE の値に応じて DRILLTHROUGH クエリまたは DRILLFACTS クエリのどちらかを実行して、LISTINGFIELDS または %OnGetListingFields() (該当する場合) で指定されているフィールドを返します。その後で、このフィールド値を %OnCompute() メソッドに渡します。このメソッドには、以下のシグニチャがあります。

```
Method %OnCompute(pSQLRS As %SQL.StatementResult, pFactCount As %Integer) As %Status
```

以下は、この指定の説明です。

- pSQLRS は、LISTINGFIELDS または %OnGetListingFields() で指定したフィールドを格納している **%SQL.StatementResult** のインスタンスです。

  このクラスの使用法の詳細は、"ダイナミック SQL の使用" を参照してください。
- pFactCount は、指定されたコンテキストでのファクトの合計数です。

このメソッドの実装では、以下の手順を実行します。

1. 文の結果を繰り返し処理します。この操作には、このインスタンスの %Next() メソッドを使用します。
2. 必要に応じて、各行の値を取得します。文の結果のインスタンス (pSQLRS) には、リスト・クエリの 1 つのフィールドにつき 1 つのプロパティがあり、プロパティの名前はフィールド名と同じです。

   例えば、前のセクションで %OnGetListingFields() は 1 つのフィールド `MxTextScore` を取得しています。この場合、pSQLRS には `MxTextScore` という名前のプロパティが存在します。
3. 必要な計算を実行します。
4. "高度な KPI の定義" の説明に従って、プラグイン・インスタンスのプロパティを設定します。少なくとも以下のプロパティを設定します。
   - **%seriesCount** ― このプラグインの系列 (行) 数を指定します。

プラグインの系列は 1 つにすることをお勧めします (PLUGINTYPE が "Pivot" に設定されているプラグインの場合、ユーザがプラグイン・プロパティをドラッグ・アンド・ドロップすると、アナライザは最初の系列のみを使用します)。

- %seriesNames(n) ー 系列 n の名前を指定します。n は整数値です。
- %data(n,propname) ー 系列 n に対して、特定のプロパティ (propname) の値を指定します。

  プロパティ名は、XData ブロックの <property> の名前と正確に一致する必要があります。

以下はその例です。

ObjectScript

```
// place answer in KPI output
set ..%seriesCount = 1
set ..%seriesNames(1) = "PluginDemo"
//set Count property of KPI -- just use received pFactCount
set ..%data(1,"PatientCount") = pFactCount

// iterate through result set to get HighScoreCount
set n = 0
set highcount = 0
while (pSQLRS.%Next()) {
  set n = n + 1

  set testscore = pSQLRS.MxTestScore
  if (testscore>95) {
    Set highcount = highcount + 1
 }
 if (pSQLRS.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", pSQLRS.%SQLCODE, ": ", pSQLRS.%Message
quit}

}
set ..%data(1,"HighScoreCount") = highcount
```

これはサンプル・クラス BI.Model.KPIs.PluginDemo からの抜粋であり、アナライザで Patients キューブと共に使用できます。

11.4 完了状態の通知

プラグインは非同期的に実行されます。プラグインを含むクエリが実行された場合、プラグインの実行が完了する前にクエリが完了する可能性があります。この場合、結果が保留中になるセルが存在します。これらのセルに、プラグインの現在の状態を (文字列 n% complete として) 表示できます。そのためには、%OnCompute() 内で定期的に %SetPercentComplete() インスタンス・メソッドを起動します。引数は 0 ～ 100 の整数です。例えば、文の結果を繰り返し処理する間に以下の処理を実行できます。

ObjectScript

```
    // update pct complete
    If (n#100 = 0) {
        Do ..%SetPercentComplete(100*(n/pFactCount))
    }
```

適切な手法は、%OnCompute() 内のロジックによって異なります。場合によっては、計算時間の大部分がこの繰り返しの外部で発生する可能性があります。

ピボット・テーブルは、結果が生成されると、自動的に更新されます。

11.5 複数のキューブで使用するプラグインの作成

これまでのセクションでは、1 つのキューブまたはサブジェクト領域で使用可能なプラグインを作成する方法について説明しました。一方、複数のキューブで使用可能なプラグインを作成することもできます。通常はプログラムによってクエリするフィールドを決定する必要があるので、実際には作成は困難です。

複数のキューブで使用可能なプラグインを作成するには、以下の追加手順を使用します。

- BASECUBE クラス・パラメータとして以下のどちらかを指定します。
  - キューブまたはサブジェクト領域の論理名のコンマ区切りリスト
  - "*" — このネームスペースのすべてのキューブおよびサブジェクト領域を示します。

  このオプションにより、プラグインを使用できるキューブおよびサブジェクト領域が決まります。
- XData ブロック内に以下のフィルタ定義を追加します。

  ```
  <filter name="%cube" displayName="Subject Area" />
  ```

  名前は %cube にする必要がありますが、表示名には任意の値を使用できます。

  アナライザ内でこのプラグインを使用する場合（妥当な場合）、現在のキューブまたはサブジェクト領域の名前がこのフィルタに渡されます。同様に、MDX クエリ内でこのプラグインを使用する場合、クエリの FROM 節によってこのフィルタの値が決定します。
- %cube フィルタの値を使用するように %OnGetMDX() メソッドを実装します。以下はその例です。

  Class Member

  ```
  Method %OnGetMDX(ByRef pMDX As %String) As %Status
  {
      Set tBaseCube = ""

      // Use %cube filter to find the base cube
      If $IsObject(..%filterValues) {
          If (..%filterValues.%cube'="") {
              Set tBaseCube = ..%filterValues.%cube
          }
      }

      If (tBaseCube'="") {
          Set pMDX = "SELECT FROM "_tBaseCube
      }
      Quit $$$OK
  }
  ```

- リスト・クエリが、必要なすべてのキューブおよびサブジェクト領域で使用できることを保証します。以下のどちらかの方法を使用します。
  - ハードコードされたリストの場合、すべてのクラスに適合するフィールドのみを使用します。
  - プログラムによって使用するフィールドを決定します。

例は、%DeepSee.PlugIn.Median および %DeepSee.PlugIn.Percentile を参照してください。

11.5.1 プログラムによるリスト・フィールドの決定

プラグインで使用するクエリで LISTINGSOURCE が "FactTable" として指定されている場合、%OnGetListingSQL() で使用するフィールドをプログラムにより決定できるようにする追加のツールがあります。以下を実行できます。

- XData ブロック内に以下のフィルタ定義を追加します。

  ```
  <filter name="%measure" displayName="Measure" />
  ```

  名前は %measure にする必要がありますが、表示名には任意の値を使用できます。このフィルタから、適用可能なキューブまたはサブジェクト領域で定義されているすべてのメジャーのリストが出力されます。

- 以下の手順に従って、%OnGetListingSQL() メソッドを実装します。

  1. %measure フィルタの値を調べます。
  2. %DeepSee.Utils クラスの %GetDimensionInfo() メソッドを使用して、選択したメジャーに関する情報を参照によって取得します。

     この情報は、次の手順の入力として使用します。
  3. %DeepSee.Utils クラスの %GetDimensionFact() メソッドを使用して、選択したメジャーが格納されているフィールドの名前を取得します。

- 必要に応じて、%OnGetListingOrderBy() コールバックと %OnGetListingMaxRows() コールバックを実装します。詳細は、**%DeepSee.KPIPlugIn** のクラスリファレンスを参照してください。

例は、**%DeepSee.PlugIn.Median** および **%DeepSee.PlugIn.Percentile** を参照してください。%DeepSee.Utils クラスのクラスリファレンスも参照してください。

11.6 リストのフィルタ処理

プラグインは、他のシナリオでは使用できない機能を提供します。すなわち、詳細リストが表示されるときに使用するレコードを指定する機能です。既定では、ユーザが結果内の 1 つまたは複数の指定したセルの詳細リストを要求すると、それらのセルに関連付けられたすべてのレコードを示すリストが表示されます。ただし場合によっては、それらのサブセットを表示することが望ましいことがあります。例えば、サンプル・クラス **BI.Model.KPIs.PluginDemo** には HighScoreCount というプラグイン・プロパティがあります。以下では、このプラグイン・プロパティをメジャーとして使用する MDX クエリの例を示しています。

```
SELECT NON EMPTY {[Measures].[%COUNT],%KPI("PluginDemo","HighScoreCount",,"%CONTEXT")} ON 0,
NON EMPTY [AllerSevD].[H1].[Allergy Severities].Members ON 1 FROM [PATIENTS]
```

	Patient Count	HighScoreCount
1 Nil known allergi	158	12
2 Minor	113	7
3 Moderate	103	5
4 Life-threatening	133	9
5 Inactive	122	8
6 Unable to determi	119	6
7 No Data Available	385	29

Nil known allergies の行について考えてみてください。どちらかのセルのリストを表示すると、既定では、158 件のレコードからなるリストが表示されます。既知のアレルギーがない 158 人の患者が存在するからです。しかし、HighScoreCount メジャーの目的は指定されたしきい値を上回るスコアの患者をカウントすることであるため、この行の HighScoreCount セルの詳細リストを表示する際は、このしきい値を上回るスコアの患者のみを表示することが望ましい場合があります。

この種類のフィルタ処理をプラグインに適用するには、リストに表示する必要のある任意のソース・クラス ID について、%OnCompute() の実装に以下のロジックを組み込んでください。

```
set ..%data("IDLIST",pluginProperty,sourceClassID) = ""
```

pluginProperty はこのフィルタ処理を使用する必要のあるプラグイン・プロパティの名前、sourceClassID はソース・クラス内の ID です (プラグインでファクト・クラスが別の形で使用される場合でも、この ID はソース・クラス ID である必要があります。このソース・クラス ID をプラグインで使用できるようにするには、%sourceId をフィールド・リストに追加します)。

指定されたプラグイン・プロパティについて、%data("IDLIST",pluginProperty) が定義されていない場合は、リストには、指定された 1 つまたは複数のセルに関連付けられたすべてのレコードが表示されます。

11.6.1 例

例を確認するには、以下に示すように、サンプル・クラス BI.Model.KPIs.PluginDemo を編集します。

1. 以下のように、LISTINGFIELDS を変更します。

```
Parameter LISTINGFIELDS As STRING = "%sourceId,MxTestScore";
```

2. %OnCompute() の highcount 変数を設定している部分を見つけて、以下のように変更します。

```
if (testscore>95) {
    Set highcount = highcount + 1
    Set tHighScoreId = pSQLRS.sourceId
    Set ..%data("IDLIST","HighScoreCount",tHighScoreId)=""
}
```

3. クラスを保存し、リコンパイルします。

その後、アナライザで、このプラグインの両方のプロパティを使用するピボット・テーブルを作成します (比較するため)。HighScoreCount プロパティを表示するセルを選択して、リストを表示します。スコアの高い患者のみが表示されている点に注目してください。比較のために、PatientCount プロパティを表示するセルを選択して、そのリストを表示します。この場合は、すべてのスコアの患者が表示されます。

11.7 使用可能なエラーのログ作成

プラグインでエラーが発生した場合、マネージャのディレクトリにあるエラー・ログ・ファイルに書き込まれます。このファイルの名前は、DeepSeeTasks_namespace.log です。

11.8 プラグインを使用する計算メンバの定義

任意のプラグイン (および他の任意の KPI) について、そこから値を取得する計算メンバを作成できます。このメンバは、アナライザ内でドラッグ・アンド・ドロップできます。このような計算メンバを作成する手順は以下のとおりです。

- ″計算メジャーの定義″ の説明に従って、計算メジャーを定義します。
- [式] で、以下の形式の MDX 式を指定します。

  ```
  %KPI(pluginname,propertyname,seriesname,"%CONTEXT")
  ```

  pluginname はプラグインの名前、propertyname はプロパティの名前、seriesname は系列の名前です。seriesname は省略できます。省略した場合、この関数はプラグインの最初の系列にアクセスします。

  "%CONTEXT" は、プラグインに行、列、およびフィルタのコンテキストを渡す特別なパラメータです。この情報は、プラグインで使用されるベース MDX クエリに渡されます。

例（系列が 1 つだけのプラグインの場合）：

```
%KPI("PluginDemo2","Count",,"%CONTEXT")
```

PLUGINTYPE が "Pivot" に設定されているプラグインの場合、ユーザがプラグイン・プロパティをドラッグ・アンド・ドロップすると、アナライザは、自身が生成する基になる MDX クエリで上記のような構文を自動的に使用します。

追加のオプションの詳細は、"%KPI" を参照してください。

12
キューブ継承を使用する方法

場合によっては、類似した複数のキューブの定義が必要になります。Business Intelligence には単純な継承メカニズムが用意されていて、これらのキューブをさらに容易に定義できます。ここでは、このメカニズムの使用方法を説明します。説明している手順の一部ではアーキテクトを使用できません。代わりに ObjectScript をサポートする統合開発環境 (IDE) である Visual Studio Code またはスタジオを使用する必要があります。

12.1 キューブ継承の概要

キューブ継承メカニズムを使用するには、以下を実行します。

1. ″キューブの定義″ で説明されている手順に従って、キューブ・クラスを定義します。このキューブには、類似するすべてのキューブで想定される主要な項目が含まれ、親キューブとして機能します。
2. 必要に応じて、このキューブを抽象としてマークし、直接使用できないようにすることができます。
   a. BI インスタンスでスタジオ・セッションを開き、アナリティクス・ネームスペースにいることを確認します。
   b. XData 定義で <cube> 要素を見つけて、属性のリストで abstract="true" を設定します。これにより、コンパイラによるキューブの検証、アナライザによるキューブの表示、およびユーザによるキューブに対するクエリの実行が行われなくなります。
   c. 変更内容を保存します。
3. 子キューブをそれぞれのキューブ・クラスで定義します。この手順は、親キューブの定義に使用する手順からは外れています。
   a. BI インスタンスで、管理ポータルに管理者権限を持つユーザとしてログインします。
   b. [ ]→[Analytics]→[ ] に移動します。
   c. **[新規作成]** ボタンをクリックします。
   d. **[定義タイプ]** で **[キューブ]** を選択します。
   e. **[キューブ名]** を入力し、オプションで、選択した **[表示名]** を入力します。
   f. **[キューブのソース]** で **[キューブ]** を選択します。
   g. **[ベースキューブ]** フィールドに、先ほど作成した親キューブの論理名を入力します。
   h. **[キューブのクラス名]** を入力し、オプションで **[クラスの説明]** を入力します。

i. [OK] をクリックして、子キューブを作成します。既定では、この方法で作成されたキューブは親キューブからすべての定義を継承します。各キューブは 1 つのベース・キューブのみを持つことができますが、ベース・キューブ自体は他のキューブから継承できます。

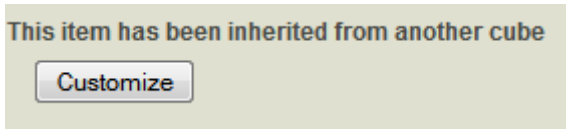
4. 親キューブが子キューブの前にコンパイルされていることを確認してください。モデル・ビューワの右側にある **[詳細]** ペインで、**[依存]** フィールドに親キューブの論理名を入力します。
5. 必要に応じて、親キューブで指定されたディメンジョン、メジャー、またはその他の最上位レベル要素を再定義します。そのためには、子キューブで定義を指定し、親キューブと同じ論理名を使用します。新しい定義で、親キューブからの対応する定義が完全に置換されます。"項目の非表示または削除" も参照してください。
6. 必要に応じて、子キューブで追加の定義 (ディメンジョン、メジャー、リストなど) を指定します。

キューブ継承メカニズムにクラス継承とのリレーションシップはありません。各サブキューブは独自のファクト・テーブルとインデックスを保持し、(実行時に) これらは親キューブに依存しません。キューブ継承メカニズムは、構築時にのみ使用され、そのキューブの定義にのみ影響します。

12.2 キューブ継承とアーキテクト

アーキテクトでサブキューブを表示する際は、継承要素を表示またはオーバーライドしたり、新しい要素を定義したりできます。左側の領域にはキューブのソース・クラスが表示されるため、このクラスから項目をドラッグ・アンド・ドロップして、サブキューブ内の要素として使用できます。アーキテクトの中央の領域には、継承項目 (斜体) とローカル定義項目 (通常書体) の両方が表示されます。

アーキテクトの中央の領域で継承項目を選択すると、**[詳細]** ペインの上部に以下のように表示されます。



[詳細] ペインの残り部分は表示専用です。

以下のサブセクションでは、継承項目を再定義する方法、オーバーライド項目を削除する方法、およびローカル要素を追加する方法を説明します。キューブのコンパイルと構築については、"キューブのコンパイルとビルド" を参照してください。サブキューブは、その他の種類のキューブと同じように扱われます。

12.2.1 継承要素の再定義

継承要素を再定義するには、その要素を中央の領域でクリックしてから、右側の **[詳細]** ペインで **[カスタマイズ]** ボタンをクリックします。カスタマイズする要素が最上位レベル要素でない場合は、その要素が含まれた最上位レベル要素をクリックしてから、右側の **[詳細]** ペインで **[カスタマイズ]** ボタンをクリックします。

[カスタマイズ] をクリックすると、親キューブからサブキューブに定義がコピーされて、継承された定義をオーバーライドするローカル定義が作成されます。この時点で、このローカル定義を編集できます。

12.2.2 オーバーライド項目の削除

継承された定義をオーバーライドするローカル定義を削除する手順は以下のとおりです。

1. アーキテクトの中央領域で、その項目の行の [X] をクリックします。
2. [OK] をクリックしてこの操作を確定します。

12.2.3 ローカル要素の追加

サブキューブにローカル要素を追加するには、**[要素を追加]** ボタンを使用するか、通常のドラッグ・アンド・ドロップ操作を使用します ("キューブへの項目の追加" を参照)。

12.3 %cube ショートカット

親キューブに %cube 変数を使用するソース式が含まれている場合は、そのキューブをシステムで構築できるようにするために、以下のいずれかの操作を実行する必要があります。

- 親キューブ・クラスを拡張するように、子キューブ・クラスを変更します (つまり、クラス継承を使用します)。
- %cube を使用する要素をオーバーライドします。ローカル定義内で、%cube を通常の完全構文 (`##class(package.class)`) に置換します。

12.4 項目の非表示または削除

メジャー、ディメンジョン、計算メンバ、または計算メジャーを非表示にするには、その項目のオーバーライドを追加して、そのオーバーライドで **[隠し]** オプションを選択します。この場合でもその項目は作成されますが、アナライザには表示されません。

レベルまたは階層を非表示にするには、それらが含まれているディメンジョンを再定義します。そのためには、そのディメンジョンのオーバーライドを追加します。そのオーバーライドで、このディメンジョンに含まれる必要のある階層とレベルをすべて定義します。継承されたディメンジョンは、この新しいディメンジョンに完全に置き換えられます。例えば、新しいディメンジョンでは、親キューブ内の対応するディメンジョンよりも少ない、または多い数のレベルを定義できます。

12.5 継承とリレーションシップ

親キューブにリレーションシップがあると、これらのリレーションシップは継承されますが、必ずしも便利な方法で継承されるとは限りません。リレーションシップは必ず元のキューブを指すためです。

例えば、2 つのキューブ (Patient と Encounter) が相互に関連付けられていて、それぞれにサブキューブ (CustomPatient と CustomEncounter) を作成するとします。既定では、CustomPatient には元の Encounter キューブを指すリレーションシップがあります。同様に、CustomEncounter キューブには Patient キューブを指すリレーションシップがあります。CustomPatient と CustomEncounter との間にリレーションシップが必要な場合、そのリレーションシップを明示的にサブキューブで定義する必要があります。

13
中間式の定義

状況によっては、Business Intelligence に、類似のロジックを使用し、同じサブルーチンを実行したり、SQL を使用して別のテーブルを参照したりする、複数のメジャーまたはディメンジョンが存在することがあります。キューブ構築のパフォーマンスを向上させるため、中間値 (各ファクトに値 1 つ) が含まれる式を定義して、その式を他のキューブ要素の定義内で使用することができます。

"BI サンプルのアクセス方法" も参照してください。

13.1 スタジオでの中間式の定義と使用

スタジオで式を定義するには、以下の操作を実行します。

1. スタジオでキューブ・クラスを開きます。
2. <cube> 開始要素を見つけます。

```
<cube name="Patients" displayName="Patients"
   owner="_SYSTEM"
    sourceClass="BI.Study.Patient"
    nullReplacement="None"
    defaultListing="Patient details">
```

3. 大なり記号 (>) の後ろに、1 つ以上の <expression> 要素を追加します。以下はその例です。

```
<expression name="patDetails" sourceExpression='%cube.GetPatientDetails(%source.PatientID)'  />
```

<expression> 要素には少なくとも以下の属性が必要です。

属性	値
name	式の名前。
sourceExpression	必要に応じて、指定されたソース・レコードに対して単一の値を返す ObjectScript 式を指定します。これを指定した場合は、sourceProperty を指定しないでください。 <expression> 要素では、sourceExpression を使用する可能性が高くなります (<expression> によって指定される中間ステップがなくても、キューブ要素がプロパティを直接使用できるため)。 <expression> 要素は、別の <expression> 要素を参照できます。
sourceProperty	必要に応じて、キューブで使用されるベース・クラスに関連するプロパティ名を指定します。ドット構文を使用してプロパティのプロパティを参照できます。これを指定した場合は、sourceExpression を指定しないでください。

sourceExpression および sourceProperty の追加オプションについては、"ディメンジョンまたはレベルのソース値の定義" および "ソース式の詳細" を参照してください。

4. メジャー、ディメンジョン、レベルまたは別の <expression> の定義内では、以下の構文を使用して式を参照します。

```
%expression.expressionName
```

5. クラスを保存し、リコンパイルします。
6. キューブを再構築するか、式を使用する任意のキューブ要素の選択的構築を実行します。

13.1.1 例

まず、<expression> 要素を使用する場合のシナリオを考えてみましょう。Patients キューブには、PatientDetails テーブルに対する SQL クエリを介してデータにアクセスするソース式によって定義されるレベルが複数存在します。例えば、Favorite Color レベルは以下のソース式で定義されます。

```
%cube.GetFavoriteColor(%source.PatientID)
```

GetFavoriteColor() メソッドには、以下のように埋め込み SQL が含まれます。

```
ClassMethod GetFavoriteColor(patientID As %String) As %String
{
    New SQLCODE
    &sql(SELECT FavoriteColor INTO :ReturnValue
    FROM BI_Study.PatientDetails
    WHERE PatientID=:patientID)
    If (SQLCODE'=0) {
        Set ReturnValue=""
        }
    Quit ReturnValue
}
```

Profession および Industry のレベルの定義は類似しています。このため、Patients キューブの構築時に、PatientDetails テーブルに対する 3 つのクエリがソース・クラスの行ごとに実行されます。

ソース・クラスの行ごとに、PatientDetails テーブルに対してクエリが 1 つだけ実行されるように、このキューブを再定義することができます。そのためには、以下の操作を実行します。

1. エディタで、Patients キューブ・クラス BI.Model.PatientsCube を開きます。

2. このクラスの <cube> 要素内で、以下の要素を追加します。

```
<expression name="patDetails" sourceExpression='%cube.GetPatientDetails(%source.PatientID)'  />
```

この式は、キューブ・クラス内でメソッドを実行します。このメソッドは以下のように定義されます。

Class Member

```
ClassMethod GetPatientDetails(patientID As %String) As %String
{
    New SQLCODE
    &sql(SELECT Profession->Industry,Profession->Profession,FavoriteColor
    INTO :Industry,:Profession,:FavoriteColor
    FROM BI_Study.PatientDetails
    WHERE PatientID=:patientID)

    If (SQLCODE'=0) {
        Set Industry="",Profession="",FavoriteColor=""
    }
    Set ReturnValue=$LISTBUILD(Industry,Profession,FavoriteColor)
    Quit ReturnValue
}
```

このメソッドは、指定の患者の複数のフィールドを取得し、その情報を含むリストを構築して、そのリストを返します。

3. 以下のように PatientDetail テーブルを使用するレベルを再定義します。

   - 以下の sourceExpression を使用するように、Industry レベルを再定義します。

     ```
     sourceExpression='$LI(%expression.patDetails,1)'
     ```

   - 以下の sourceExpression を使用するように、Profession レベルを再定義します。

     ```
     sourceExpression='$LI(%expression.patDetails,2)'
     ```

   - 以下の sourceExpression を使用するように、Favorite Color レベルを再定義します。

     ```
     sourceExpression='$LI(%expression.patDetails,3)'
     ```

4. クラスを保存し、リコンパイルします。
5. キューブを再構築するか、再定義されたレベルの選択的構築を実行します。

13.2 アーキテクトでの中間式の定義

アーキテクトで式を定義するには、以下の操作を実行します。

1. アーキテクトでキューブ定義を開きます。
2. **[要素を追加]** を選択します。

   ダイアログ・ボックスが表示されます。
3. **[新規項目名の入力]** に、名前を入力します。
4. **[式]** を選択します。
5. [OK] を選択します。
6. 右側の **[詳細]** タブで、それぞれのフィールに値を指定します。

14
その他のオプション

ここでは、Business Intelligence のキューブおよびサブジェクト領域のその他のオプションの使用方法について説明します。

"BI サンプルのアクセス方法" も参照してください。

14.1 キューブ定義内の maxFacts の指定

キューブの開発時は、一般にリコンパイルと再構築を頻繁に行います。大量のデータ・セットを使用している場合は、キューブをより迅速に再構築するため、ファクト・テーブル内のファクト数を制限することがあります。そのためには、%BuildCube() の pMaxFacts 引数を指定します。"ターミナルでのキューブの構築" を参照してください。

または、以下のように `maxFacts` 属性を指定できます。

1. スタジオでキューブ・クラスを開きます。
2. `<cube>` 要素を見つけます。

```
<cube name="HoleFoods"
caption="HoleFoods Sales"
defaultListing="Listing"
nullReplacement="Missing Value"
actionClass="HoleFoods.KPIAction"
sourceClass="HoleFoods.Transaction">
```

3. この要素に `maxFacts` 属性を追加します。

```
<cube name="HoleFoods"
caption="HoleFoods Sales"
defaultListing="Listing"
nullReplacement="Missing Value"
actionClass="HoleFoods.KPIAction"
sourceClass="HoleFoods.Transaction"
maxFacts="10000">
```

   指定した値によって、ファクト・テーブルの最大サイズが決まります。
4. クラスを保存し、リコンパイルします。
5. キューブを再構築します。

重要　　`maxFacts` 属性は、必ずキューブの配置前に削除してください。

14.2 キューブで使用するレコードの制限

既定では、システムはソース・クラスのすべてのレコードを使用します。キューブ定義を変更して、レコードの一部を無視することができます。このためには、以下のいずれかまたは両方を実行します。

- キューブ定義に [**ビルドの制限**] オプションを指定します。詳細は、"InterSystems Business Intelligence のモデルの定義" を参照してください。

  キューブがデータ・コネクタに基づいている場合、このオプションは何の効果も持ちません。

- キューブ・クラスの %OnProcessFact() コールバックを定義する。

14.2.1 %OnProcessFact() コールバック

ベース・テーブルのレコードの一部を無視して、それらをキューブに含めないようにするために、キューブ定義クラスで %OnProcessFact() メソッドを定義できます。

```
classmethod %OnProcessFact(pID As %String, ByRef pFacts As %String,         Output pSkip As %Boolean,
 pInsert As %Boolean) as %Status
```

システムは、ファクト・テーブルの構築または更新時に、ベース・テーブル内の各行へのアクセスの直後にこのメソッドを呼び出します。このメソッドには以下の値が渡されます。

- pID : 処理対象のソース・データ内の行の ID。
- pFacts : 行で使用される値を含む多次元配列。この配列の構造は以下のとおりです。

ノード	値
pFacts(factName)。factName は、[ファクト・テーブルのフィールド名] オプションで指定された、ファクト・テーブル内のレベルまたはメジャーの名前です。	そのレベルまたはメジャーの値。例 : Magnolia、47。

- pInsert : 新規の行の場合は 1。

このレコードをスキップする場合はメソッドで pSkip を True に設定し、スキップしない場合は pSkip を False に設定します。

例えば、以下のコールバックでは、キューブは好きな色が青であるすべての患者を無視します。

Class Member

```
ClassMethod %OnProcessFact(pID As %String, ByRef pFacts As %String,
Output pSkip As %Boolean, pInsert As %Boolean) As %Status
{
    if pFacts("DxColor")="Blue"
    {
        set pSkip=1
    } else {
            set pSkip=0
            }
     quit $$$OK
}
```

この例では、キューブが [**ファクト・テーブルのフィールド名**] オプションで好きな色のレベルとして DxColor を定義していると想定しています。

注釈　%OnProcessFact() によるレコードの評価が、pFacts 配列を使用して指定するレベルまたはメジャーに基づいている場合、キューブに対して選択的構築を実行する際には常に、対応するキューブ要素を含める必要があります。前述の例では、キューブに対するあらゆる選択的構築に DxColor を含める必要があります。そうしないと、pFacts("DxColor") の値が未定義になるため、すべてのレコードでエラーが生じます。

別の例として、以下のコールバックでは、キューブは ID が 1000000 未満であるすべてのレコードを無視します。

Class Member

```
ClassMethod %OnProcessFact(pID As %String, ByRef pFacts As %String,
Output pSkip As %Boolean, pInsert As %Boolean) As %Status
{
    if pID<1000000
    {
        set pSkip=1
    } else {
            set pSkip=0
            }
     quit $$$OK
}
```

14.3 レベルのメンバの手動指定

アーキテクトでレベルを定義した後に、メンバおよびその順序を手動で指定できます。そのためには、以下のように、スタジオでキューブ・クラスを変更します。

1. スタジオでキューブ・クラスを開きます。
2. レベルを定義するセクションを見つけます。

```
<level name="MyLevel" displayName="MyLevel" ... >
</level>
```

3. </level> 行の直前で、以下のように一連の <member> 要素を追加します。

```
<level name="MyLevel" displayName="MyLevel" ... >
<member name="first" displayName="first" spec="1" />
<member name="second" displayName="second" spec="2" />
<member name="third" displayName="third" spec="3" />
<member name="fourth" displayName="fourth" spec="4" />
<member name="fifth" displayName="fifth" spec="5" />
</level>
```

各 <member> 要素によって 1 つのメンバが識別されます。<member> 内では、name はそのメンバの名前を指定し、displayName はそのメンバのオプションのローカライズ表示名を指定し、spec はそのメンバのオプションのキー仕様を指定します。

name と spec のルールは以下のとおりです。

シナリオ	必要な権限	結果
name のみを指定する場合	name の値は、大文字と小文字の区別も含めて、ソース・プロパティまたはソース式の値と完全に一致している必要があります。	name の値がキーになります。
name と spec の両方を指定する場合	spec の値は、大文字と小文字の区別も含めて、ソース・プロパティまたはソース式の値と完全に一致している必要があります。	spec の値がキーになります。

その他の注意事項：

- これらの属性で XML 予約文字を使用することはできません。この後のサブセクションを参照してください。
- <member> 要素を必要な数だけ組み込みます。
- これらの要素の順序によって、対応するメンバの既定の並べ替え順序が決まります。
- レベルのすべてのメンバを組み込みます (ワイルドカード・オプションはありません)。
- <member> に、同じレベル内で <property> と同じ name を指定することはできません。

4. クラスを保存し、リコンパイルします。
5. キューブを再構築するか、レベルの選択的構築を実行します。

注釈　このオプションは、メンバのセットに変更が発生しないと予測される場合にのみ有益です。新規レコードを受信し、そのレコードに <member> 要素のいずれかで指定される値が含まれない場合、そのレコードはこのレベルに表示されません。

14.3.1 XML 予約文字

スタジオでキューブ・クラスを編集する際は、name や displayName などの属性の値に XML 予約文字を使用することはできません。代わりに、以下のように置き換えます。

予約文字	代わりに使用する文字
<	<
&	&
"	"
'	'

> を使用してもエラーにはなりませんが、この文字の代わりに > を使用できます。

これらの文字をアーキテクトの入力フィールドに入力すると、上記の置換が自動的に行われます。スタジオでキューブ・クラスを検証すると、アーキテクトで他の置き換えも行われる場合があります。前述のテーブルは、必須の置換のみを示しています。

14.4 ファクト・テーブルへのカスタム・インデックスの追加

必要なすべてのインデックスが自動的に定義されます。ただし、ファクト・テーブルは独自の用途に直接的に使用でき、このように使用する際は追加のインデックスが必要となる場合があります。このようなインデックスを追加するには、スタジオでキューブ・クラスを編集し、必要に応じて <index> 要素を追加します。

"キューブ・クラスのリファレンス情報" を参照してください。

14.5 その他のキューブ・コールバックのカスタマイズ

クラス %DeepSee.CubeDefinition には、オーバーライドしてキューブの動作をさらにカスタマイズできるコールバック・メソッドがあります。このセクションでは、一般的にオーバーライドされるいくつかのメソッドについて説明します。

また、このページの "%OnProcessFact() コールバック" および "キューブまたはサブジェクト領域の動的フィルタ処理" も参照してください。

その他のオプションの詳細は、%DeepSee.CubeDefinition のクラスリファレンスを参照してください。

14.5.1 %OnAfterProcessFact() コールバック

%OnAfterProcessFact() コールバックを使用すると、任意の指定されたレコードがキューブのファクト・テーブルで追加または更新された後に実行されるカスタム・コードを追加できます。

```
ClassMethod %OnAfterProcessFact(pID As %String, ByRef pFactArray As %String, pUpdateStatus As %Status)
 as %Status
```

システムは、このコールバックに以下の情報を渡します。

- pID : 処理対象のソース・データにおける行の ID。
- pFacts : ファクト名を添え字とした、行で使用される値を含む配列。
- pUpdateStatus : %ProcessFact() によって返されようとしている状態。エラーが渡されると、このエラーは Business Intelligence ログおよび **^DeepSee.BuildErrors** グローバルにすぐに記録されます。

%ProcessFact() メソッドは、このメソッドによって返される値を無視します。

14.5.2 %OnGetDefaultListing() コールバック

%OnGetDefaultListing() コールバックを使用すると、システムで既定のリストが使用されている場合に、使用するリストの名前をプログラムで指定できます。このメソッドのシグニチャは、以下のとおりです。

```
ClassMethod %OnGetDefaultListing() as %String
```

このコールバックは、ユーザが既定のリストを要求したときに呼び出され、特定のリストが要求されたときには効果を示しません。以下に例を示します。

```
ClassMethod %OnGetDefaultListing() As %String {     Quit "Listing By Product" }
```

例えば、これを使用すると、ユーザが属するロールをチェックし、そのロールで適切なリストを表示できます。

14.5.3 %OnExecuteListing() コールバック

リスト・クエリを実行するために追加の設定作業が必要になる場合もあります。

そのためには、キューブ定義クラスに %OnExecuteListing() メソッドを実装します。

```
ClassMethod %OnExecuteListing(pSQL As %String) as %Status
```

システムは、リスト・クエリを実行する直前にこのメソッドを呼び出します。システムはこのメソッドを呼び出すときに、実行対象のリスト・クエリである値 pSQL を渡します。

14.5.4 %OnAfterBuildCube() コールバック

%OnAfterBuildCube() コールバックは、定義されている場合、キューブの構築後に呼び出されます。

```
ClassMethod %OnAfterBuildCube(pBuildStatus As %Status, pBuildErrors As %Boolean = 0) As %Status
```

システムは、**%DeepSee.Utils** の %BuildCube メソッドの最後のステップとしてこれを呼び出します。システムは、このコールバックに以下の情報を渡します。

- buildStatus は、その時点でのキューブ構築状況です。
- factErrorCount は、構築エラーが発生したファクトの数です。

お使いの実装は、**%Status** のインスタンスを返す必要があります。

このメソッドは、キューブ構築のロックが解放される前に呼び出されるため、一度に 1 つのプロセスのみがこのコールバックを実行できます。

14.6 キューブまたはサブジェクト領域の動的フィルタ処理

サブジェクト領域に対してハードコードされたフィルタを指定する代わりに（またはこの指定に加えて）、%OnGetFilterSpec() コールバックを実装できます。これによって、実行時にフィルタのコンテンツを指定できます。このコールバックは、キューブ・クラスでも使用できます。したがって、キューブおよびサブジェクト領域の両方を動的にフィルタ処理できます。

このメソッドのシグニチャは、以下のとおりです。

```
classmethod %OnGetFilterSpec(pFilterSpec As %String) as %String
```

pFilterSpec は **<subjectArea>** の **filterSpec** 属性の値です。このメソッドは、有効な MDX セット式を返す必要があります。以下に例を示します。

Class Member

```
ClassMethod %OnGetFilterSpec(pFilterSpec As %String) As %String
{
    Quit "AgeD.H1.[20 to 29]"
}
```

以下に簡単な例をもう 1 つ示します。この例では、メソッドが $ROLES 特殊変数をチェックして、ユーザが **%All** ロールに属する場合は、フィルタ処理を解除します。

Class Member

```
ClassMethod %OnGetFilterSpec(pFilterSpec As %String) As %String
{
    if $ROLES["%All" {
        //remove any filtering
        set pFilterSpec=""
        }
    Quit pFilterSpec

}
```

別の例では、以下のコールバックが元の値と追加のフィルタの交差結合を実行することにより、元のフィルタ値を変更します。

Class Member

```
ClassMethod %OnGetFilterSpec(pFilterSpec As %String) As %String
{
    //test to see if $ROLES special variable includes TestRole
    if $ROLES["%DB_SAMPLE" {
        //a member expression like the following is a simple set expression
        set colorrestrict="colord.h1.[favorite color].red"

        //create a tuple that intersects the old filter with the new filter
        //this syntax assumes original is just a member
        set newfilter="CROSSJOIN("_pFilterSpec_","_colorrestrict_")"
        set pFilterSpec=newfilter
        }
    Quit pFilterSpec

}
```

A

KPI クラスおよびプラグイン・クラスのリファレンス情報

ここには、Business Intelligence KPI クラスおよびプラグイン・クラスのリファレンス情報が記載されています。

変更を加えた後は、KPI クラスをリコンパイルする必要があります。

A.1 基本要件

KPI を定義するには、以下の要件に合致したクラスを作成します。

- %DeepSee.KPI を拡張する必要があります。
- KPI という名前の XData ブロックを格納している必要があります。
- この XData ブロックに対して、以下のように XMLNamespace が指定されている必要があります。

  ```
  XMLNamespace = "http://www.intersystems.com/kpi"
  ```

- XData ブロック内のルート要素が <KPI> であり、この要素がこのページに記載された要件に従っている必要があります。
- クラスには、複数のクラス・パラメータを定義できます。"クラス・パラメータの指定" を参照してください。

プラグインの要件は、以下の例外を除いて、同じです。

- クラスは、%DeepSee.KPI ではなく、%DeepSee.KPIPlugIn を拡張する必要があります。
- クラスには PLUGINTYPE クラス・パラメータを定義できます。"プラグインの定義" を参照してください。

A.2 KPI またはプラグインの共通属性

KPI またはプラグインの要素の多くに、以下の属性が存在します。わかりやすくするため、ここにこれらをリストします。

属性	目的
name	要素の論理名。

属性	目的
displayName	(オプション) ユーザ・インタフェースで使用されるこの要素のローカライズ名。この属性を指定しない場合は、代わりに name 属性で指定した値が使用されます。詳細は、"ローカライズの実行" を参照してください。
description	(オプション) この要素の説明。
disabled	(オプション) コンパイラがこの要素を使用するかどうかを制御します。この属性が "true" の場合、コンパイラはこの属性を無視します。既定では、この属性は "false" です。

A.3 <kpi>

<kpi> 要素は、KPI クラスまたはプラグイン・クラスにおける XData ブロックのルート要素です。この要素には以下の項目があります。

属性または要素	目的
name、displayName、description、disabled	このページで前述の "KPI の共通属性" を参照してください。
caption	使用されていません。
sourceType、mdx、sql	"ハードコードされたクエリを使用した KPI の定義" を参照してください。
rangeLower、rangeUpper、thresholdLower、thresholdUpper	"速度計の範囲としきい値の指定" を参照してください。
actionClass	(オプション) この KPI 内で定義されたアクションに加えてこの KPI で使用可能なアクションを定義する関連 KPI クラスを指定します。別の KPI の完全なパッケージとクラス名を指定します。
<property>	(オプション) ゼロ個以上の <property> 要素を含めることができます。このそれぞれが、KPI で使用するクエリの列に対応します。<property> 要素の指定がないと、クエリ結果を表示できません。
<action>	(オプション) ゼロ個以上の <action> 要素を含めることができます。このそれぞれを、この KPI を基にしたスコアカードの作成時に使用できます。
<filter>	(オプション) ゼロ個以上の <filter> 要素を含めることができます。このそれぞれを、この KPI を基にしたスコアカードの作成時に使用できます。

A.4 <property>

<kpi> 内の <property> 要素には以下の属性があります。

属性	目的
name、displayName、description、disabled	このページで前述の "KPI の共通属性" を参照してください。 KPI プロパティの name または displayName は、このプロパティを表示するメータ・ウィジェットの既定のキャプションとして使用されることに注意してください。その用途に適した名前を指定すると便利です。
columnNo	このプロパティのデータを格納するクエリ内の列の番号。最初のデータ列は 1 です。
format	(オプション) この KPI がピボット・テーブル・ウィジェットで表示される場合のこのプロパティの既定の数値形式。例えば、format="##.###" のようになります。
style	(オプション) この KPI がピボット・テーブル・ウィジェットで表示される際に、このプロパティを表示するセルに適用される CSS スタイル。例えば、style="color:red" のようになります。
headerStyle	(オプション) この KPI がピボット・テーブル・ウィジェットで表示される際に、対応するヘッダ・セルに適用される CSS スタイル。例えば、style="color:red;font-style:italic" のようになります。
defaultValue	使用されていません。

KPI テスト・ページでは、format、style、および headerStyle の各属性は無視されます。これらの属性は、ピボット・テーブル・ウィジェットにのみ作用します。

A.5 〈filter〉

<kpi> 内の <filter> 要素には以下の属性があります。

属性	目的
name、displayName、description、disabled	このページで前述の "KPI の共通属性" を参照してください。
filterProperty	このフィルタで制御される <property> 要素の論理名。このオプションは、プログラムを使用してフィルタ値を取得する際に使用されます。
defaultValue	(オプション) このフィルタの既定値。スコアカードに KPI を組み込む際にもフィルタの既定値を指定できることに注意してください。その既定値は、<filter> 要素の defaultValue 属性より優先されます。
sql、valueList、displayList	"KPI フィルタの定義" を参照してください。
searchType	(オプション) このフィルタをウィジェットに表示する際に使用するコントロールの種類を指定します。既定のコントロールは、ドロップダウン・リストです。代わりにカレンダーを表示するには、この属性を "day" と指定します。
dependsOn	(オプション) このフィルタが依存する別のフィルタの名前を指定します。例えば、KPI に State および City という名前のフィルタがある場合、City フィルタの定義に dependsOn="State" と指定できます。

A.6 <action>

<kpi> 内の <action> 要素には以下の属性があります。

属性	目的
name、displayName、description、disabled	このページで前述の "KPI の共通属性" を参照してください。 name 属性は、applyFilter、setFilter、refresh、showListing、viewDashboard、navigate、newWindow、rowCount、rowSort、colCount、colSort のいずれにもすることができません (大文字と小文字の区別はありません)。

<action> 要素は、アクション自体を定義するものではありません。アクションは、クラスの %OnDashboardAction() コールバック・メソッド内で定義します。"カスタム・アクションの定義" を参照してください。

KPI またはプラグインに定義されているアクションをダッシュボード・デザイナーに通知し、選択して使用できるようにするには、<action> 要素が必要です。

B

Text Analytics で使用するセカンダリ・キューブの生成

重要　InterSystems IRIS Business Intelligence 内での Text Analytics の統合では、InterSystems IRIS 自然言語処理 (NLP) が使用されていますが、インターシステムズではこれは非推奨となっています。このため、これはインターシステムズ製品の今後のバージョンから削除される可能性があります。以下のドキュメントは、リファレンスとして既存ユーザのみに提供されます。代わりのソリューションを見つけるサポートを希望する既存ユーザは、インターシステムズのサポート窓口 にお問い合わせください。

ここでは、"キューブでの Text Analytics の使用法" で説明したように、NLP メジャーを Business Intelligence キューブに追加済み (およびそのメジャーを使用する NLP ディメンジョンを作成済み) であることを前提としています。ここでは、エンティティの出現およびディクショナリ・マッチング結果を分析するセカンダリ・キューブを生成する方法について説明します。これらのキューブを構築する前に、必ずメイン・キューブを構築してください。

このページで紹介している方法は、"キューブでの Text Analytics の使用法" の "エンティティ出現を数値化するメジャーの追加" および "一致結果を数値化するメジャーの追加" で説明したプラグインを使用する方法の代替方法です。メイン・キューブを再構築または同期化するときには、必ずセカンダリ・キューブを手動で再構築する必要があるため、プラグインの方が優れた方法です。

"BI サンプルのアクセス方法" も参照してください。

B.1 エンティティ出現キューブ

エンティティの出現を表すキューブを生成するには、ターミナルで以下のコマンドを使用します。

ObjectScript

```
d ##class(%iKnow.DeepSee.CubeUtils).CreateEOCube(cubename,measurename)
```

cubename は NLP メジャーが含まれるキューブの名前、measurename は NLP メジャーの名前です。

このメソッドは、キューブ・クラスがエンティティ出現データにアクセスするための読み取り専用クラスを生成します。このエンティティ出現クラスの名前は BaseCubeClass.measurename.`EntityOccurrence` です。BaseCubeClass はベース・キューブ・クラスのクラス名、measurename は NLP メジャーの名前です。

このメソッドは、キューブ・クラス BaseCubeClass.measurename.`EOCube` も生成します。この新しいキューブの定義は以下のとおりです。

- このキューブの論理名は BaseCubemeasurenameEO で、BaseCube はベース・キューブの論理名、measurename は NLP メジャーの名前です。
- このキューブは、エンティティの出現を表します。すなわち、このキューブのファクト・テーブルには、一意のエンティティの出現ごとに 1 行が存在します。
- キューブには Count メジャーが定義され、エンティティの出現をカウントします。
- キューブには Entity Value ディメンジョンが定義され、エンティティ値別にエンティティの出現をグループ化します。

  これは、カスタム計算ディメンジョンです。計算ディメンジョンの一般情報は、"キューブおよびサブジェクト領域の高度な機能の使用" を参照してください。

  このキューブの場合、ファクトは 1 回のエンティティの出現を表します。このキューブでは、ファクトとエンティティ値の間に一対一のリレーションシップが存在することに注意してください。一方、メイン・キューブでは、ファクトとエンティティ値の間に一対多のリレーションシップが存在します。
- キューブには Roles ディメンジョンが定義され、メンバとして concept と relation が存在します。これらのメンバにより、エンティティの出現は概念および関係にグループ化されます。これらの用語の詳細は、"NLP が識別する論理テキスト・ユニット" を参照してください。
- キューブには、ベース・キューブへのリレーションシップが含まれます。このリレーションシップの名前は Main cube です。

以下に、Aviation デモでエンティティ出現キューブを使用するピボット・テーブルの例を示します。

	Airplane	Balloon	Glider	Gyrocraft	Helicopter	Ultralight
airplane	4,980		15		1	5
pilot	4,322	22	98	9	478	4
engine	1,438		8	1	121	1
accident	1,181	4	22	3	164	
flight	1,178	2	21	2	156	1
time	887	2	13	3	118	
runway	829		20	1	8	
helicopter	47				701	
examination	601		6	2	92	

このピボット・テーブルは、以下のように定義されています。

- メジャーは、Count (エンティティ出現数) です。
- 行には、Entity Value ディメンジョンのメンバが表示されます。このディメンジョンの各メンバは、1 回のエンティティの出現を表します。例えば、pilot メンバは、ソースにエンティティ pilot が 1 回出現することを表します。
- 行には、メイン・キューブの Aircraft ディメンジョンの Category レベルのメンバが表示されます。このディメンジョンの各メンバは、指定された航空機カテゴリに関連付けられているソース・ドキュメントを表します。

このピボット・テーブルは、例えば、Airplane カテゴリのイベントのレポートにエンティティ airplane が 4980 回出現していることを示します。

B.2 マッチング結果キューブ

マッチング結果を表すキューブを生成するには、ターミナルで以下のコマンドを使用します。

ObjectScript

```
d ##class(%iKnow.DeepSee.CubeUtils).CreateMRCube(cubename,measurename)
```

cubename は NLP メジャーが含まれるキューブの名前、measurename は NLP メジャーの名前です。

このメソッドは、キューブ・クラスがマッチング結果データにアクセスするための読み取り専用クラスを生成します。このマッチング結果クラスの名前は BaseCubeClass.measurename.MatchingResults です。BaseCubeClass はベース・キューブ・クラスのクラス名、measurename は NLP メジャーの名前です。

このメソッドは、キューブ・クラス BaseCubeClass.measurename.MRCube も生成します。この新しいキューブの定義は以下のとおりです。

- このキューブの論理名は BaseCubemeasurenameMR で、BaseCube はベース・キューブの論理名、measurename は NLP メジャーの名前です。
- このキューブは、ディクショナリの一致を表します。すなわち、このキューブのファクト・テーブルには、一意のディクショナリの一致ごとに 1 行が存在します。
- キューブには Count メジャーが定義され、ディクショナリの一致をカウントします。
- キューブには Score メジャーが定義され、ディクショナリの一致のスコアを示します。
- キューブには Dictionary ディメンジョンが定義されます。Dictionary レベルは、ディクショナリ別に一致をグループ化し、Dictionary Item レベルはディクショナリ項目別に一致をグループ化します。

  このキューブの場合、ファクトは 1 つのマッチング結果を表します。このキューブでは、ファクトとディクショナリ項目の間に一対一のリレーションシップが存在することに注意してください。一方、メイン・キューブでは、ファクトとディクショナリ項目の間に一対多のリレーションシップが存在します。
- キューブには Type ディメンジョンが定義され、メンバとして entity、CRC、path、および sentence が存在します。これらのメンバにより、一致はエンティティ、CRC、パス、および文にグループ化されます。これらの用語の詳細は、"NLP が識別する論理テキスト・ユニット" を参照してください。

  このデモの場合、Type ディメンジョンのメンバは entity と CRC だけです。
- キューブには、ベース・キューブへのリレーションシップが含まれます。このリレーションシップの名前は Main cube です。

以下に、Aviation デモでマッチング結果キューブを使用するピボット・テーブルの例を示します。

		Accident		Incident	
		Count	Score	Count	Score
Injuries	fatal	106	85.31		
	minor	152	151.50	1	1
	none	713	340.52	24	12.85
	serious	93	91.51		

このピボット・テーブルは、以下のように定義されています。

- メジャーは Count (マッチング結果数) および Score (ディクショナリの一致の累積スコア) です。

- 行には、Injuries ディクショナリのメンバが表示されます。このディメンジョンの各メンバは、このディクショナリの特定の項目に一致するエンティティを表します。例えば、minor メンバは、ディクショナリ項目 minor に一致するソースの各エンティティを表します。
- 行には、メイン・キューブの Type ディメンジョンのメンバが表示されます。このディメンジョンの各メンバは、Accident または Incident のどちらかのレポート・タイプに関連付けられているソース・ドキュメントを表します。

このピボット・テーブルは、例えば、タイプ Incident のソースには、ディクショナリ項目 fatal との一致が存在しないことを示します。一方、タイプ Accident のソースには、このディクショナリ項目との一致が 106 件存在します。