SAP Solutions for Performance Management

SAP Business Planning and Consolidation
5.x Logic Explained
Part 2 of 3
Agenda

SQL Based Script Formulas

- Syntax example
- Managing scope
Script Logic Approach

Script Formulas

- Use when calculation can be performed before the aggregations
- Data is generated at the base levels in all dimensions and the cube performs natural aggregations
  - Triggered via a data update or DTS package
- Currency conversion, Unit X Rates, Allocations etc.
- Syntax MDX or SQL based logic?

*Note: SQL Logic is NOT Microsoft but our proprietary language processed on the application tier*

- Answer is ALWAYS go with SQL logic
- MDX will perform poorly with concurrent users on the application
- SQL logic has the same capabilities with new functions like “Calc_Dummy_Org” for parent values and “Calc_Each_Period” for carry forward logic
Calculate 3rdPartyRev as Sales Units * Sales Price

The “When” question
- Before or after aggregations?

The “WHAT” and the “HOW”.
- Data creation
- Logic files get created in the Admin Console
  - Default Logic – Executed on Data Send
  - Package – Executed in Batch Mode
- Syntax
  - MDX – AVOID
  - SQL
  - Table (parameter) driven – Stored Procedures (script program generated)
### Before or after aggregations?

<table>
<thead>
<tr>
<th></th>
<th>Entity1</th>
<th>Entity2</th>
<th>Entity3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales Units</td>
<td>100,000</td>
<td>115,000</td>
<td>127,000</td>
<td>342,000</td>
</tr>
<tr>
<td>Sales Price</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Sales</td>
<td>1,000,000</td>
<td>1,150,000</td>
<td>1,270,000</td>
<td>10,260,000</td>
</tr>
</tbody>
</table>

### After Aggregations

<table>
<thead>
<tr>
<th></th>
<th>Entity1</th>
<th>Entity2</th>
<th>Entity3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales Units</td>
<td>100,000</td>
<td>115,000</td>
<td>127,000</td>
<td>342,000</td>
</tr>
<tr>
<td>Sales Price</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Sales</td>
<td>1,000,000</td>
<td>1,150,000</td>
<td>1,270,000</td>
<td>3,420,000</td>
</tr>
</tbody>
</table>

### Before Aggregations
Script Logic – MDX Consideration

Advantages
- Straightforward (more intuitive)
- Standard in market (not proprietary)

Disadvantage - Performance, Performance, Performance
- General inefficiency
- Concurrency
- Running on a lot of intersections (Batch Mode)

BPC 5 schema impact
- Hierarchy specific = high maintenance
- SQL 2000 complex tuple syntax required
- SQL 2005 performance killer just to have the property

Bottom line – DON”T USE IT unless you have no SQL alternative
SQL Based Logic

Developed for performance and scalability

Completely separate syntax (proprietary)

Bases calculations on the existence of data (record) in FACT tables *(always keep this in mind as you get more advanced in your calculations this can be critical in your design)*

Calculations performed at the SQL level not the OLAP (Cube) level.

Major paradigm shift
SQL Based Logic – Behind the Scenes

Logic module loads a selection of data into memory (scope of the query).

The user defined formulas are applied to the selected data and a result set of records is generated.

The result set is written back to the database.
Understanding the selection region (Scope of the Query)

- When invoked via Excel the Scope is
  - Account Dimension – All non-calculated accounts
  - All Other Dimensions – Only specific members sent via worksheet/workbook

- When invoked via a DMM Package
  - Selected Dimension – Only members selected for prompted dimensions (package and logic dependent)
  - Non-specified Dimension – All base members
  - Currency – LC if not specified

- Scope can also be controlled by written logic instructions in the script
SQL Based Logic Basic Syntax

The WHEN statement triggers SQL Logic

*WHEN {criteria}

*IS {valid condition1}[,{valid condition2},…]

*REC[((FACTOR|EXPRESSION={Expression}[,{dim1}= {member},{dim2}=…])

[*ELSE]

...

*ENDWHEN

Lets break this down…
WHEN Examples

*WHEN * Operates on the entire selection
*IS *

*WHEN ACCOUNT Operates only on the SalesUnits Account
*IS “SalesUnits”

*WHEN Account.ACCTYPE Operates only on Accounts where the ACCTYPE property is INC or EXP.
*IS “INC”, "EXP"
The *REC statement generates a new record

*REC[[[FACTOR|EXPRESSION={Expression}[,{dim1}= {member},{dim2}=…}]]

FACTOR and EXPRESSION – Derive the new value by applying simple math.

- FACTOR is faster and simpler to use if just multiplying the value of current record by something (an exchange rate or a price or a percent ownership or just a straight value).
- EXPRESSION lets you write more complex formulas like EXPRESSION=(%VALUE%+GET(ACCOUNT="SomeAccount"))/GET(ACCOUNT="SomeOtherAccount") you cannot write this expression using a FACTOR
  - %VALUE% contains the value of current record.
  - The GET lets you retrieve the value of some other record.
  - Hint: the GET only looks for records you already have in memory, it does not go to the DB.

The last parameter(s) allow you to redirect the result to different dimension members.
To calculate a new Statistical Account Named “UpsideRev” that is the total of all revenue accounts marked up 25%..

*WHEN ACCOUNT.GROUP
*IS “REV”
*REC(FACTOR=1.25,ACCOUNT=“UpsideRev”)
*ENDWHEN
*COMMIT
SQL Based Syntax - Expression

To calculate a new Statistical Account Named “UpsideRev” that is the total of all revenue accounts marked up 25%..

*WHEN ACCOUNT.GROUP
*IS “REV”
*REC(EXPRESSION=%VALUE%*1.25,ACCOUNT=“UpsideRev”)
*ENDWHEN
*COMMIT
GET allows you to apply another value from the selected region and use it in the FACTOR or EXPRESSION.

\[
GET(\{dimension\}={member}[, \{dimension\}={member}]…)
\]

Instead of hard coding a percentage, to pull the percentage from another Account named “UpsidePCT”

*WHEN ACCOUNT.GROUP
*IS “REV”
*REC(FACTOR=GET(ACCOUNT=“UpsidePCT”),ACCOUNT=“UpsideRev”)
*ENDWHEN
*COMMIT
More Logic Control

There are more methods of controlling your complex logic execution:

- Expanded scope
- Selecting a range
Query Scope Considerations

Reminder: SQL logic only applies to the region of records in memory (not in the DB)

There are often issues where a value required in the calculation is outside the scope of your selection.

There are a few of ways to handle this
- Expand the member region
- Override the member region
- Use the LOOKUP\ENDLOOKUP structure
Expand Member Selection

It is possible to expand the selection range passed into the Logic Module runtime memory.

*XDIM_MEMBERSET {Dimension name} = {Members Set}

We want our Unit * Price Calculation to run on the “Non_Interco” member of the INTCO Dimension.

*XDIM_MEMBERSET IntCo = Non_Intco

It is possible to merge members to the set of members passed into the region.

*XDIM_ADDMEMBERSET {dimension} = {members set}
Assume you need only this calculation: Account A = Account B + Account C

The logic to achieve this will look as follows:

*WHEN ACCOUNT
*IS B,C (if you find values for these accounts…
*REC(ACCOUNT=A) … add them into account A)
*ENDWHEN

This logic would, by default, load in memory all possible accounts. However, if this is the only calculation, there is no need to load in memory all accounts, and the logic will run faster if it’s written this way:

*XDIM_MEMBERSET ACCOUNT=B, C
*WHEN *
*IS *
*REC(ACCOUNT=A)
*ENDWHEN

The instruction * XDIM_MEMBERSET supports also the “not equal to” operator with the syntax:

* XDIM_MEMBERSET {Dimension}<>{MemberSet}
To avoid MDX to get a parent value, you can generate parent totals in memory

// make sure you have all entities in memory
*XDIM_MEMBERSET ENTITY=<ALL>

// generate all parent values based on the hierarchy number
*CALC_DUMMY_ORG ENTITY=PARENTH1

// use them as appropriate (note the # sign for valid Parent IDs)
*WHEN ENTITY

    *IS #SALESEUROPE

        *REC(FACTOR=1/GET(ENTITY=#WORLDWIDE1), ACCOUNT=”SomeRatio”) 

*ENDWHEN
SQL Based Logic Syntax – Calc_Each_Period

To avoid MDX to get previous balances, you can generate the carry-forward balance

Combination of the following is required:

- Keyword “Prior”
- SQL syntax for Calc_Each_Period
- SQL syntax for Calc_Dummy_Org
- Memory variables
SQL Based Logic Syntax – Calc_Each_Period

*CALC_EACH_PERIOD
*XDIM_MEMBERSET TIME=PRIOR,%TIME_SET%,%YEAR%.DEC
*CALC_DUMMY_ORG ACCOUNT=PARENTH1
*WHEN TIME
*IS PRIOR
  *WHEN ACCOUNT
    *IS CYNI
      *REC(ACCOUNT=#OPE_CYNI,TIME=NEXT)
    *ENDWHEN
  *ELSE
    *WHEN ACCOUNT
      *IS #NETINCOME
        *REC(ACCOUNT=#OPE_CYNI,TIME=NEXT)
        *REC(ACCOUNT=CYNI)
    *IS #OPE_CYNI
      *WHEN TIME.PERIOD
        *IS<>JAN
          *REC(ACCOUNT=#OPE_CYNI,TIME=NEXT)
          *REC(ACCOUNT=CYNI)
        *ENDWHEN
    *ENDWHEN
  *ENDWHEN
*ENDWHEN
Selecting a Range

We want our Unit * Price Calculation to run on the members of the INTCO Dimension where the GROUP Property has a value of “NoInp”.

*SELECT(%IC_MBRS%, "ID", "INTCO", 
"[GROUP] = ‘NoInp’")

*XDIM_MEMBERSET IntCo = %IC_MBRS%

*SELECT is run against the SQL tables

Or using MDX against the cube

*MEMBERSET({variable}, {member set in MDX format})

*MEMBERSET (%IC_MBRS%, "filter{[INTCO].members, 
[INTCO].properties("GROUP")="NoInp"}")

*SELECT and *MEMBERSET statements are executed at the time the logic is validated, and the expanded result is written in the LGX file. This means that if the related dimension is modified, it may be necessary to re-validate the logic.

- Workaround is to call an LGF file instead of an LGX file in the DM package. The file is validated at run-time which captures any recent changes to the dimensions.
- Can also call an LGF for default logic using an include myDefault.lgf
Selecting a Range

We want our Unit * Price Calculation to run on the base members of the DataSrc Dimension under the “PreAdj” Parent..

*MEMBERSET(\%DS_MBR\S\%,"Descendants([DataSrc].[PreAdj],999, LEAVES")

*XDIM_MEMBERSET DATASRC = \%DS_MBR\S\%

*MEMBERSET uses MDX and is run against the Cube.

The MDX query is very light (no access to values only to dimension information), so we still use it.

In allocations there is newer syntax not requiring MDX by which you can say DATASRC=BAS(PREADJ)

*XDIM_MEMBERSET, as follows:

*XDIM_MEMBERSET DATASRC=BAS(PREADJ)  However it's not supported yet.

If you have only one level you can also use this instruction, which is fully SQL:

*SELECT(\%DS_MBR\S\%,ID,DATASRC,PARENTH1='PREADJ')

*XDIM_MEMBERSET DATASRC=\%DS_MBR\S\%