SAP Solutions for Performance Management

BPC 5 Logic

Part 3 of 4

SAP Solutions for Performance Management

Logic Overview – Part 1

Script Formulas Overview – Part 2

Script Formulas Advanced – Part 3

Agenda

SQL Based Script Formulas

- Performance control
- Managing your destination
- Cross application logic

Performance Control – SQL Based Logic

The engine by default will lump all the selected members of all dimensions into a single query *(could be a performance issue depending on design of logic)*.

Limit the maximum number of members in a query

***XDIM_MAXMEMBERS {dimension}= {max number of members}**

***XDIM_MAXMEMBERS Entity = 50**

Above instruction, if entities to process exceed the limit of 50 members, logic module will break the query into multiple queries of no more than 50 entities each.

Important: the maximum number of members can be specified for up to TWO dimensions:

***XDIM_MAXMEMBERS Entity = 50**

***XDIM_MAXMEMBERS Time = 1**

Or..

***XDIM_MAXMEMBERS Entity = 50, PRODUCT=100**

Performance Control – *Commit

Commit – an instruction to write back results when a logic file may have dependent formulas *(rely on result of calculations performed by the cube or some other formulas in the same logic)

There are a variety of commit instructions that can be used:

***COMMIT**

***COMMIT_EACH_MEMBER={dimname}**

***COMMIT_EACH_LEVEL={dimname}**

The behavior of the XDIM_MAXMEMBERS option for SQL logic is that the process will commit the generated result as they are generated by each single query.

When XDIM_MAXMEMBERS is used in MDX type logics, the logic query is broken in as many queries as required. However all resulting records are committed to the database in one lump at the end of the loop of queries.

- **It may be preferable to perform a commit to the database after each individual query. This can be accomplished inserting the instruction:**

***COMMIT_MAXMEMBERS**

Performance Control - *Go

Unlike a COMMIT, *Go does not write back records after the WHEN / ENDWHEN structure. All generated results are merged with the original set of source records and the logic re-starts from the beginning of the record set, to process the following WHEN / ENDWHEN structures.

There is no limit to the number of GO instructions that can be inserted within a COMMIT section. However, since each GO generates a little bit of overhead, and requires an additional scan of the record set, their number should be kept to the minimum.

Important: even if at a first glance it may seem that the GO instruction can be used in place of a COMMIT instruction, this is not quite true. All instructions that are COMMIT-specific (like for example XDIM_MEMBERSET) are still COMMIT-specific and not GO-specific. In other words you cannot redefine the data region to process for each GO instruction, but only for each COMMIT instruction. The GO instruction only sets a stop-and-go point between WHEN / ENDWHEN structures of the same COMMIT section, i.e. of the same data region.

Any GO instruction defines the end of a logic section, more or less like a COMMIT instruction, in the sense that the logic is executed normally up to that point.

Clarifications on the use of *Destination

The destination region default is equal to the source region. The engine extends it to include the destination members.

- **If for example the logic says *REC(ENTITY="XYZ"), XYZ will be added to the destination region.**
- **If the logic says something unpredictable, like *REC(ENTITY=ENTITY.SOMEPROPERTY), then the destination region is expanded automatically to include ALL members of that dimension (all entities).**

Only the generated records are used to calculate their difference from what is stored ***(If, in the example, the *REC statement generated a records for entity ABC, all records in the destination region pointing to other entities will simply be ignored).***

The destination region is used when the logic is required to calculate the difference and send only the difference to the posting engine.

Clarifications on the use of *Destination

Misunderstandings ***(seen in previous training presentations)*****:**

- **Incrementing results: only happens if the logic is not written correctly, for example if you add a value to itself.**
- **Only if you activate the calculate_difference option, the logic will issue two queries.**
- **To our knowledge the destination region is never wrong. Sometimes it is just un-necessarily big, and you may want to control its size with a *DESTINATION statement.**
- **The only problem with the default destination region is if somebody changes the values in the destination region before the logic is completed, the difference may calculate against an older value.** ***Making the logic check the lck tables might help prevent this from happening (currently we do not do this).***

Understanding the Destination

The instruction *DESTINATION could be used to optimize the size of the destination region to query, with some benefit to the performance and memory footprint of the logic execution.

***DESTINATION {DimensionName}={MemberSet}**

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

***DESTINATION<>{MemberSet}**
This operator can be handy to pass to the SQL query smaller lists of valid members that will be more efficiently parsed by the Microsoft SQL engine.

You can also use this instruction to re-direct the destination. For example update the Budget Category with DECFCST.

***DESTINATION Category=Budget**

***WHEN Category**

***IS “DECFCST”**

***REC(FACTOR=1,Category=“Budget”)**

***ENDWHEN**

Understanding the *Clear_Destination

Clearing a destination – SQL executes only for existing FACT table records. If the FACT table results in a zero, the logic is still executed. However, if the admin performs an optimize, the record may no longer exist but may be required for the calculation (eliminations, allocations or consolidation logic)

- **The logic can force the clearing of all data in the destination region**
- **WARNING: This instruction MAY LEAD TO A DELETION OF THE INPUT DATA, if used incorrectly. A good understanding of their behavior is required, in order to avoid the risk of serious losses of data in the database.**
- **DESTINATION instruction MUST be used for ALL the dimensions for which the user wants to be SURE that the correct region is cleared.**

***CLEAR_DESTINATION**

***DESTINATION {DimensionName}={MemberSet}**

Example

Putting it all Together

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

*XDIM_MEMBERSET INTCO = %IC_MBRS%

*MEMBERSET(%DS_MBRS%,"Descendants([DataSrc].[PreAdj],999,LEAVES)")

*XDIM_MEMBERSET DATASRC = %DS_MBRS%

*WHEN CATEGORY.BUDLOGIC

*IS "Y"

    *WHEN Account

    *IS "SalesUnits"

    *REC(EXPRESSION=%VALUE%*GET(ACCOUNT="SalesPrice")*-1,Account="3rdPartyRev")

    *ENDWHEN

*ENDWHEN

*COMMIT

Cross application – *Destination_App

Destination of data required for the results is not restricted to a single application

- ***DESTINATION_APP - it is possible to redirect the results of a logic execution to be written into an application different from the one against which it was originally executed**

The destination application may only shares some dimensions of the original application. In this case the missing dimensions can be dropped, added or renamed with the instruction:

- ***SKIP_DIM= {dimension name}[,{dimension name},…]**
- ***ADD_DIM {dimension name}={value}[,{dimension name}={value},…]**
- ***RENAME_DIM {dimension name}={value}[,{dimension name}={value},…]**

Example:

***RENAME_DIM ACCOUNT_FLASH= ACCOUNT_MAIN**

Here is a more complete example:

***DESTINATION_APP = CentralApplication**

***SKIP_DIM= PRODUCT,MARKET**

***ADD_DIM DATASRC=INPUT**

***ADD_DIM CURRENCY=LC**

***RENAME_DIM ACCOUNTPM=ACCOUNTMAIN**

Cross Application - *LOOKUP

Source data required for a calculation is not restricted to a single application

- ***Lookup/*EndLookup - This set of instructions can be used in conjunction with a WHEN/ENDWHEN structure to retrieve (“lookup”) some other values that may be needed either in the calculation of the new value or to define some criteria to be evaluated.**
- **A relationship is defined between the current record being processed and the record to lookup**
- **Can be used in Factor\Expression or *WHEN**
- **Can lookup values in current application or different application in the same application set**

LOOKUP Syntax

***LOOKUP {App}**

***DIM [{LookupID}:] {DimensionName}="Value" | {CallingDimensionName}[.{Property}]**

[*DIM …]

***ENDLOOKUP**

Where:

- **{App} is the name of the application from which the amounts are searched**
- **{DimensionName} is a dimension in the lookup app**
- **{CallingDimensionName} is a dimension in the current application**
- **{LookupID} is an optional identifier of the "looked-up" amount. This is only required when multiple values must be retrieved.**

Consider the currency calculation Value * Rate, where value is in the primary application and rate is in the supporting rate application

***LOOKUP RATE**

***DIM RATESRC="RATECALC"**

***DIM RATE=ACCOUNT.RATETYPE**

***DIM SOURCECURR:INPUTCURRENCY=ENTITY.CURRENCY**

***DIM USD:INPUTCURRENCY="USD"**

***DIM EURO:INPUTCURRENCY="EURO"**

***ENDLOOKUP**

Real-world example

The LGF version

```
//CASH FLOW MOVEMENT LOGIC
*SELECT(%FROM_ACCOUNT%, "[ID]", "ACCOUNT", "[CF_TO_ACCT] <> """)
*CALCULATE_DIFFERENCE = 0
*XDIM_ADDMEMBERSET TIME = PRIOR
*XDIM_MEMBERSET ACCOUNT = %FROM_ACCOUNT%
*WHEN DATASRC
                *IS <> OPEN
                *REC(FACTOR=-1,ACCOUNT=ACCOUNT.CF_TO_ACCT,TIME=NEXT)
                *WHEN TIME
                                *IS <> PRIOR
                                *REC(ACCOUNT=ACCOUNT.CF_TO_ACCT)
                *ENDWHEN
*ENDWHEN
*COMMIT
```

The LGX version

```
*CALCULATE_DIFFERENCE = 0
*XDIM_ADDMEMBERSET TIME = PRIOR
*XDIM_MEMBERSET ACCOUNT =
ACCREC,IICACCREC,INVENTORY,OFFSHARES,INVESTSUB,EQUITYMETHOD,ACCPAY,IICACCPAY,LTDEBT,COMMONSTOCK,PREFERREDSTOCK,APIC,FXEQUITYG
L,FXCYNIGL,OUTOFBAL
*WHEN DATASRC
                *IS <> OPEN
                *REC(FACTOR=-1,ACCOUNT=ACCOUNT.CF_TO_ACCT,TIME=NEXT)
                *WHEN TIME
                                *IS <> PRIOR
                                *REC(ACCOUNT=ACCOUNT.CF_TO_ACCT)
                *ENDWHEN
*ENDWHEN
```

Copyright 2006 SAP AG. All Rights Reserved

Copyright 2006 SAP AG. Alle Rechte vorbehalten