Best Practice in Essbase Business Rule Writing

Evgeniy Galanzovsky
PROMATIS software GmbH
Pforzheimer Straße 160, 76275 Ettlingen

Key Words
Essbase, Bussines Rules, FIX Statements, Aggregation, Block creation, Dense calculation, Sparse calculations,

Introduction

There are two quite important things in any BI system, performance and possibility to extend the functionality. BI systems help to answer questions like "What is going on?" and "Why does it happen?". These questions are not specified and in real life business analysts actually don't know what they are looking for. So this means that they definitely cannot define the requirements of a report. They don't know how the report should look like and what kind of data it should contain. But we can be sure that they want to have it right now. Previously the process of analyzing data looked like this: At first the business user asked an IT consultant to provide him a report with the columns A, B, C and D. After a while when he had got that report he changed his mind and required to add columns F and G as well as one calculation column with formula "A+B". After he had received the new report he changed his mind again. After a few iterations the IT consultant just gave him a big export in excel with all columns that he could add and after that the business user started to play with these columns in excel to get the desired analysis. This did not happen because the business users were not able to understand what they had to do but rather because he did not know what he was looking for. And after he had found what he wanted, he preferred to have another report with new columns and new data. The process of analyzing data represents a process of checking assumptions. After one assumption has been checked the user checks another assumption and so on. A very important premise of that process is continuity. The business users should work with the system by themselves while they are investigating the data based on their ideas about possible dependencies and interesting aspects. If this process is interrupted because the user needs to wait for an IT consultant to create the report or because the system needs to recalculate something the user may lose his ideas.

Thanks to BI vendors like Cognos, Business Objects, Oracle BI and so on users are able to create reports with drag and drop possibilities, they can easily create calculation columns, graphs and whatever they want. So we are able to provide the continuity of the process but there are still some business requirements which cannot be solved on end-user side. In that case IT consultants have to provide all necessary data or calculations. They should be able to extend the functionality of the system.

So there are two aspects which are quite important for business analysis:

Performance and Extensibility.

Let's look at an example (Figure 1).

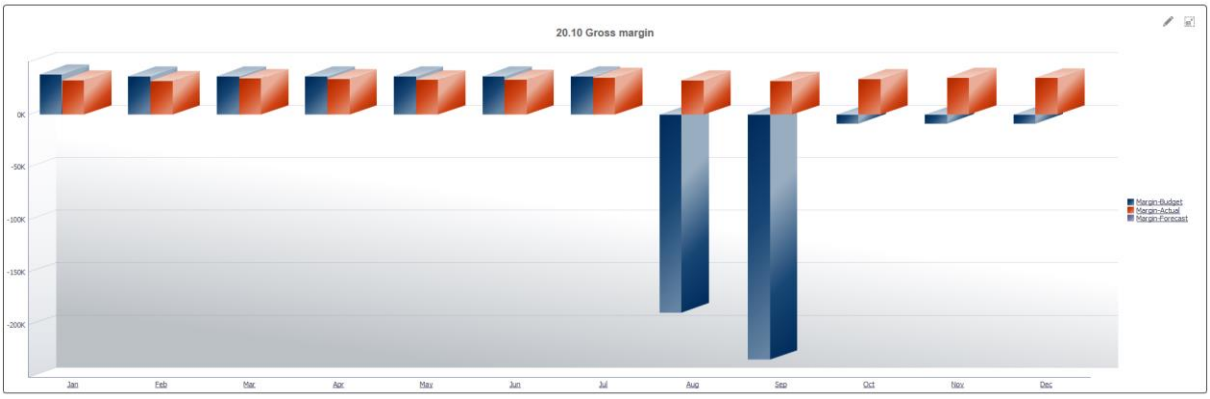

Figure 1: Gross margin dashboard

In this example we want to find out why gross margin is negative in August and September. If we drill down to the lowest level of the data and compare Actual with Budget we will see that Budget COGS is much higher than in Actual. And if we check the calculation of COGS we will see that we include all our cost accounts into the calculation what is wrong. We need to exclude at least research and development expenses from that calculation.

```
FIX (
    {Year},{Scenario},{Version}                                  /* Calculation Parameters  */
    )
    FIX (
        "Materials_NA",                                          /* Materials not defined */
        "Production",                                            /* Production department */
        "Regions_NA",                                            /* Region not defined */
        "3d"                                                     /* Third party customers */
        )
/* COGM calculation*/
            FIX (
                @Relative("YearTotal"),0                         /* All periods*/
                @Relative("ProductsAll",0)                       /* All products*/
                "PL_403000000"                                   /* Account, Change of stock - production*/
                )

                    "INV_05_V" =                                                    /* Inventory movements, Production  */
                                "Expenses_to_Production"->"Products_NA"->"INV_07_V"  /* Total Production costs */
                                *   (
                                    "INV_05_Q"                                      /* Production quantity */
                                        /  ("INV_05_Q"->"ProductsAll")              /* Total Production quantity */
                                    )
                                + ("PL_404000000"->"MaterialsAll"->"INV_07_V");     /* Materials consumption */

                    "INV_05_P" =                                /* Production costs of one unit */
                                "INV_05_V"                      /* Total production costs of one product */
                                /"INV_05_Q";                    /* Production quantity of one product */
            endfix
    endfix
ENDFIX
```

Figure 2: Business rule. Cost of goods manufactured calculation.

We can see that the code is pretty easy to read. We can quickly find the row where we have to change the code. Let's say we want to take into account losses in production. It means, that on row "Total production quantity" we have to add new line within "- "INV_06_Q"->"ProductionAll" " where INV_06_Q is "Production losses"

Now let's have a look at these aspects in terms of Essbase business rules.
When we develop Essbase business it is better to follow some rules which help us to write fast calculations and code which can easily be read.

In this presentation I want to focus on a few of these rules:

- FIX statements and aggregation
- Calculations
- Block creation

FIX Statement
We define 3 levels of FIX statements:

Level 1: Parameters. That level usually is the same over all business rules in the application.
Level 2: Rules. That level defines members which are being used in the current business rule.
Level 3: Calculation. Here we define members for the current calculation.

Every new FIX statement should begin with a new line and a tab indentation. That helps to identify different fix statements. Every new member set of new dimensions should begin with a new line. Every member set should have aliases and a description. It is allowed to locate member sets of the same dimension in the same line but it is required to define all aliases and the name of the dimension.
It is necessary to avoid direct member sets and use member set functions to define list of members. It is allowed to use UDA and attributive dimensions. That approach will help to modify calculations without code modification.
Actually in terms of performance it is better to follow the rule "FIX – Sparse, IF – Dense".

Within levels we can easily find the appropriate row where we need to change a member set, to add a new list of members or to remove some members from the list. Having each member set on a new line helps us to control the quantity of member sets. The main rule here is to define all dimensions before calculation. It is not allowed to skip some dimensions. If we skip some of them the calculation will be applied to all members of skipped dimensions and will increase the time of calculation.

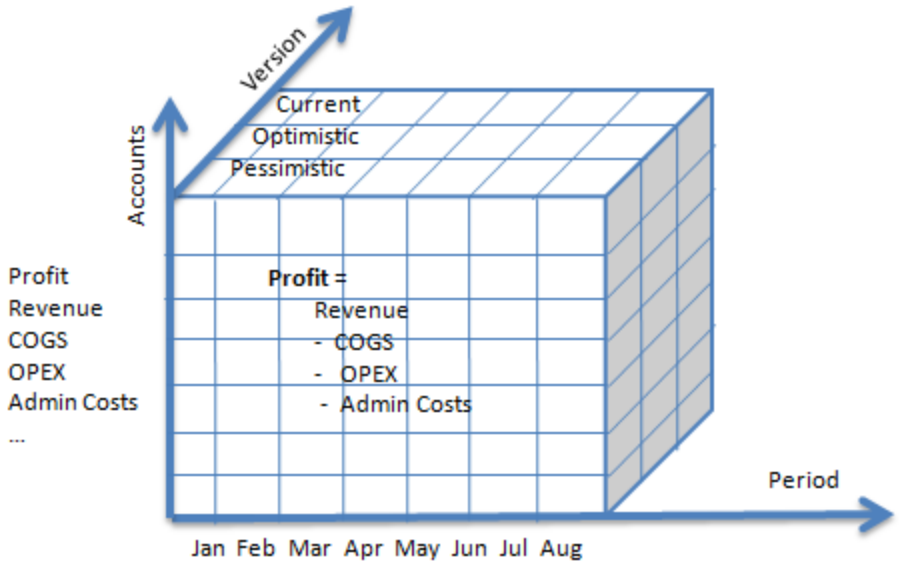

Figure 3: OLAP cube

Calculation
Dense calculations are fast. That is an axiom which we should be remembered when we design an application. A good design will provide a much better performance then all tricks with business rules on later stages. When we create an application it is better to define dense and spares in terms of calculations and data store. It is better to define calculations inside a block. Calculations between blocks will significantly decry the performance of the calculations. That's why 50% of success is defined before business rules are designed and written.
So first of all focus on calculations inside blocks. After that all calculations should be executed within level 0 members. If we need some aggregated data for some calculations we should aggregate only required dimensions. All other aggregations should be done at the end of the business rules.
Then if we use IF statement or sparse calculation, we should not forget to clear the target range of calculations. If we don't do that we will have meaningless data in some ranges.

If we cannot avoid the sparse calculations it is better to create calculations in block mode. In that case the calculation will take less time. We should avoid all unnecessary calculations and check whether there is any data or not before calculating.
There is another tip regarding performance. Fewer members – faster calculation. That means that for some calculations it is better to use some other dense dimension to fix the statement. If the quantity has calculated more than 12 member dimensions it is better to use period dimensions (if the period dimension is a dense dimension).

Creating blocks
One of the general pain of business rules is creating blocks. There are a lot of cases when we need to create blocks before calculating and block creation is a quite resource intensive task. So if we miss something in design stage block creation task will be headache. If you have to create blocks in your calculation very often please review your application design.
Anyway block creation is a problem which we can solve without reduction of performance.
First of all let's look at the list of commands that create blocks:

```
Data copy
Data load
@CALCMODE (TOPDOWN)
Aggregation of sparse dimension
Sparse calculation without cross-dimm
@CreateBlock
```

The best in terms of performance is data copy. We can use that command when there is a block in source. Let's say we know that the users enter data in version Current. That means that we can use the version Current as source for the blocks. We also can use that form only when the quantity of blocks which we are going to create is the same as in source. In that case data copy is the best choice. Otherwise we have to choose less performance commands. Do not forget to clear data in the target source.
We can use data load if we need to create a quite big quantity of blocks. We can use JExport and JImport commands. Technically it is just an export of data into a flat file and an import. We can define that we want to import data. Essbase automatically creates block before importing data.
Another method is creating blocks in calcmode. We have to pay attention on the comment set @CALCMODE (TOPDOWN) before calculation. If we calculate data for blocks which does not exist and if the right side of the calculation returns data blocks will be created. Please be very careful with that command and use it only for small ranges of members. Nevertheless it is a convenient command which is easy to use. That command is resource intensive.
Two other methods are quite similar. We can run aggregations by sparse dimensions or calculate sparse dimension members. It helps us to create blocks precisely. This approach is also working fast and in comparison with data copy it does not create nonsense data.
But in general I would propose to pay attention on application architecture. If you see that you need to create blocks very often in your calculation it is time to think about redesigning the application. Do not create empty blocks in the database. A good approach also checks the data in database in regard to the value "0" appearing in any block. It is better to avoid storing "0"so if there is any "0" in the database it is better to replace it within "#missing". That will decrease index file and increase performance in terms of retrieving data.

Restrictions
There are also some commands which are very resource intensive.
CREATENONMISSINGBLK as @ALLOCATE will create blocks in all potential blocks. It means that every calculation will read that block and try to calculate something but the block is empty. The only way how you can delete empty blocks is to restructure the database. Other commands like @XREF and @XWRITE are very convenient and allow to simplify the developer's job. Apart from that they are very resource intensive. If you cannot avoid using that commands try to apply them to a very small range of the data only.

Summary

I showed you only two aspects which have an impact on the performance and extensibility of the system. There are a lot of other things which are also important, such as order calculation, parallel calculation, cache memory settings, data storage settings, data base compressions, quantity of levels in hierarchies and so on. The structure of our code, where we add comments and different kinds of comments - it is only about possibility to extend or improve the calculation. If we add a fix in one row, that does not mean that the calculation will work slower. But it means that we will spend less time to find the place where we have to change something, and moreover the good structured code will help us to avoid mistakes.

Now you know how to organize your Essbase Bussines rules and you've heard about some commands which will help you to improve the performance of your Essbase Bussines rules.

Contact address:
Evgeniy Galanzovsky
Senior Consultant
PROMATIS software GmbH
Pforzheimer Straße 160
D-76275 Ettlingen

Telefon:	+49 (0) 7243-2179-0
Fax:	+49 (0) 7243-2179-99
E-Mail	evgveniy.galanzovsky@promatis.de
Internet:	http://www.promatis.de
	http://www.horus.biz