IBM Cognos Proven Practices: IBM Cognos TM1 FEEDERS

Product(s): IBM Cognos TM1; Area of Interest: Financial Management

Guy Jones
Client Technical Manager
IBM

John Leahy
Proven Practices Advisor
IBM

Skill Level: Advanced

Date: 08 Jun 2012

One of the more advanced concepts in the development of IBM Cognos TM1 cubes is the proper implementation of FEEDERS within TM1 rules. This document describes FEEDERS and how to use them effectively for improved performance when building IBM Cognos TM1 cubes.

View more content in this series

Introduction

Purpose

One of the more advanced concepts in the development of IBM Cognos TM1 cubes is the proper implementation of FEEDERS within TM1 rules. This document describes FEEDERS and how to use them effectively for improved performance when building IBM Cognos TM1 cubes.

Pre-requisite

This document covers an advanced IBM Cognos TM1 concept and uses TM1-specific terminology. The reader should have an understanding of IBM Cognos TM1 cubes, dimensions, rules, and terminology before proceeding.

Applicability

IBM Cognos TM1 9.5.1 through IBM Cognos TM1 10.1

Exclusions and Exceptions

No exclusions or exceptions have been identified.

Definition

What are FEEDERS?

FEEDERS are used by the IBM Cognos TM1 calculation engine to efficiently handle sparsity in a cube with rule-based calculations. FEEDERS identify the fields in a cube that are utilized in a rule-based calculation and flag them as exceptions to the sparse data compression algorithm.

Why use FEEDERS?

OLAP cubes can be very sparsely populated due to the different levels of granularity in the available data or simply from the nature of the cube. For example, in a Bill of Materials cube which includes many end products, many different parts, and many different suppliers there will be significantly more empty cells than populated cells, since each end product doesn't require every different part from every different supplier. In our experience, the ratio of non-populated to populated cells in large sparse OLAP cubes can often be greater than 100,000,000:1.

Using standard dense matrix algorithms for calculations in a sparse cube can consume large amounts of computing resources and take a significant amount of time to complete. In order to improve calculation performance in sparse cubes, IBM Cognos TM1 by default uses a sparse data compression algorithm. This allows IBM Cognos TM1 to perform dimensional aggregations or consolidations very quickly and efficiently. Keep in mind that these dimensional aggregations or consolidations are calculated based on the hierarchical design of the dimensions in the cube. They are different than rule-based calculations which are used to calculate various fields within or across dimensions and cubes. An example of a dimensional aggregation or consolidation would be a dimension that contains the items Product A, Product B, Product C, and Total Products. Based on the hierarchical design of the dimension, any cells with values for Product A, Product B, and Product C would aggregate to Total Products. An example of a rule-based calculation would be: Price * Volume = Revenue. In IBM Cognos TM1, rules are defined outside of the dimension in the TM1 Rules Editor and are then augmented to a TM1 cube.

However, once IBM Cognos TM1 detects that a rule has been added to a cube, the sparse data compression algorithm will automatically be disabled by TM1. This is to ensure that the rule will calculate correctly, since if the sparse data compression algorithm were left on then the calculated field (the dependent variable) and some or all of the source fields (the independent variables) would be compressed and not available for the calculation to work properly. If the rule is then implemented, the TM1 cube performance will decrease significantly since it cannot take advantage of the sparse data compression algorithm.

The two options above are 1) Design a cube that has only dimensional aggregations or consolidations and then take full advantage of the sparse data compression algorithm for excellent performance; or 2) Design a cube that utilizes TM1 rules for complex calculations but suffers from poor performance.

Fortunately, there is a third option when designing IBM Cognos TM1 cubes. That option is to design a cube that utilizes TM1 rules for complex calculations but that has the sparse data compression algorithm enabled in the cube with an exception for the fields that are involved in any rule-based calculations. This third option is where SKIPCHECK and FEEDERS come into play in TM1 rules.

What is SKIPCHECK?

SKIPCHECK is used in TM1 rules to restore the TM1 sparse data compression algorithm which is disabled by default when a TM1 rule is created for a TM1 cube. It essentially overrides TM1 default behaviour for cubes with rules.

What is the default setting?

By default IBM Cognos TM1 does not automatically insert SKIPCHECK or FEEDERS into TM1 rules. These parameters must be written into the rule by the TM1 Developer. If SKIPCHECK and FEEDERS are not written into a TM1 rule then all cells in the cube will be evaluated by the TM1 calculation engine and zero suppression algorithms. A TM1 rule without SKIPCHECK or FEEDERS will perform all calculations correctly but performance will be slower than if those parameters had been included. The IBM Cognos TM1 Developer can optionally choose to define whether or not SKIPCHECK and FEEDERS are used on a cube-by-cube basis.

How do FEEDERS work?

FEEDERS identify the fields in a cube that are utilized in a rule-based calculation and flag them as exceptions to the sparse data compression algorithm. This allows the sparse data compression algorithm to continue to work in a cube that has a rule associated with it. The FEEDERS flag identifies those fields that will use the standard dense data structures algorithm, so all cells within the FEEDERS area will be evaluated during calculations. Generally, the more FEEDERS that a rule has, the slower the performance will be.

FEEDERS can also be used for zero suppression. The zero suppression algorithms also use FEEDERS to exclude zero items from the view. Note that the system uses the FEEDER to determine if the cell is included in the view, not whether the cell is actually zero or not. Cells will be correctly calculated and displayed in an unsuppressed view, but when zero suppression is enabled they disappear. This is because the cells are not fed.

FEEDERS are defined by the IBM Cognos TM1 Developer in the TM1 Rules Editor. In almost all cases, the Developer defines a FEEDER for each rule. When the user chooses to start-up the TM1 server or when the user saves the rule, the system will

evaluate the FEEDER statements in the rules and mark the calculated cells in the cube to highlight cells that are non-zero.

One of the more common pitfalls that many IBM Cognos TM1 Developers fall into when implementing FEEDERS is defining FEEDERS in an inefficient manner. The two most common situations that can impact both performance and data integrity are:

- Implementations that result in the IBM Cognos TM1 application being “over-fed”
- Implementations that result in the IBM Cognos TM1 application being “under-fed”

The negative consequence of under-feeding an application is that non-zero cells can be excluded from calculations, rollups, and zero suppression and may therefore result in incorrectly calculated values. The consequence of over-feeding is that more cells will be fed than are required, therefore calculations will be slower and it will often take longer to evaluate the FEEDERS in particular at TM1 server start-up and rule save-time. In addition, the FEEDERS will take-up additional memory. It is therefore better to over-feed rather than under-feed, since over-feeding will only result in slower performance but under-feeding can result in incorrect values. However, there are techniques that can be employed to minimize over-feeding.

To summarize, over-feeding will provide the expected results (calculation integrity etc.), but the application will operate in a less efficient manner taking more computing resources and/or longer times to complete. This point is highlighted by the fact that the IBM Cognos TM1 calculation engine will actually perform 2 passes when it comes to zero suppression. First the engine will evaluate the rules and dimensional rollups and produce a view. In the first pass, it will use FEEDERS to decide whether to include or exclude the cell. If you have zero suppression switched on, the system will perform a second pass on the view produced in the first pass – it will remove zero rows and columns based upon whether the cell in the view is actually zero or not.

Five Important Considerations Specific To FEEDERS

The five most important considerations specific to FEEDERS when building a TM1 application are:

1. FEEDERs only apply to rule-calculated cells or a rollup thereof.
2. N: level calculated cells will always need a FEEDER for both calculation integrity and zero suppression. Note that in a hierarchical dimension, N: level items are leaf-level or child-level items. C: level items are consolidated or parent-level items that aggregate N: level items. For example, if you have the items January, February, March, and Q1 defined in a dimension, then the N: level items are the three months and the C: level item is the Q1 total.
3. C: level cells are automatically fed if their N: level children have FEEDERS defined. The system will look at the FEEDERS of the children of a summary cell to determine if the summary cell is fed or not. This principle applies to

dimensional rollups of N: level calculated cells or C: level calculated cells. To illustrate this point suppose you have a measure that represents a “Closing Balance”. You would write a rule so that all summary items in the time dimension grab the value from the last leaf level child of that parent. So for this example, the Q1 closing balance would be taken from March rather than adding up January, February and March. Since March is a child of Q1 you do not need to feed the calculation for Q1. FEEDERS for C: level cells are determined by evaluating the FEEDERS of the N: level children of the parent. C: level FEEDERS are, however, evaluated for zero suppression.

4. Once a cell is fed it will continue to refer to the original FEEDER even after the FEEDER statement is changed. There are two methods to force a reprocessing of the FEEDERS after a change has been made: 1) Issue a CubeProcessFeeders command in a TM1 TurboIntegrator (TI) process; or 2) Restart the IBM Cognos TM1 Server. This is critical to be aware of during the application development phase. For example, if an original FEEDER was over-feeding a cube but was then corrected by writing a more restrictive FEEDER that may appear to be accurate. However, until the TM1 Server is restarted it will not be possible to verify that the FEEDER is working correctly.
5. FEEDERS from numeric cells only fire once. FEEDERS from String cells fire whenever their value changes. A parameter is often used in a rule to determine the location for the calculation result. If a numeric member is being used to feed the calculation, then that FEEDER will only fire once – so any update to the parameter will not create the FEEDER for the new location.

Where Are FEEDERS Defined?

FEEDERS are defined in the TM1 Rule Editor. In general, there should be at least one accompanying FEEDER for each rule. The rules should be structured as follows:

```
SKIPCHECK;
#
# Write your Rules here
#
FEEDERS;
#
# Write Your FEEDERS here
#
# End of Rule
```

For example,

```
SKIPCHECK;
['A*B']=n:['A']*['B'];

FEEDERS;
['A']=>['A*B'];
```

Note that in this example only `['A']` is being used to FEED the rule. The reason why `['B']` is not also included will be explained more fully in the section titled **Multiplication**.

Determining If A Cell Has Been Fed

Use these methods to determine if a cell is fed or not. Note that if you have been performing a lot of updates, it may be worth restarting the IBM Cognos TM1 server before you perform these tests.

Method 1 – Visually Inspect the Rule

This document will provide guidance on defining FEEDERS for many possible applications. By applying the guidelines provided here, it should be possible to visually inspect rules, FEEDERS, and cube data and then identify where FEEDERS can be used to improve performance or where an existing FEEDER can be improved to provide even greater performance.

Method 2 – Visually Check Calculation Results

Check that the calculation is being performed correctly and that the rollup of that item is showing the correct results. In the simple example shown below, notice that the rollup of “A*B” is not being done correctly. This is because “A*B” is not being fed for “Jan”, “Feb” and “Mar” and therefore the IBM Cognos TM1 calculation engine is ignoring those cells when it performs the rollup.

Below is a view of an IBM Cognos TM1 cube as viewed through IBM Cognos TM1 Cube Viewer. The view is showing two attributes “A” and “B” with the result “A*B” being derived via a rule. In this example, the rule is defined as follows:

```
['A*B'] = n: ['A']*['B'];
```

Notice the intersection of “A*B” and “Q1-10” is showing zero, which is an incorrect result.

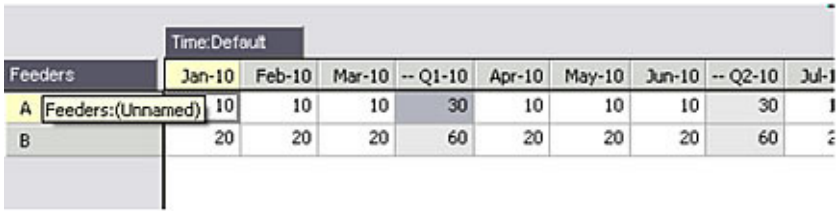
Figure 1 – Incorrect calculation result due to the fact that [A*B] is not being fed correctly or at all

	Time:Default								
Feeders	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10	Jun-10	-- Q2-10	Jul
A	10	10	10	30	10	10	10	30	
B	20	20	20	60	20	20	20	60	
A*B	200	200	200	0	200	200	200	0	

Method 3 – Check for Zero Suppression

Click on the “Suppress Zeros” icon, and then recalculate the view. The view below will occur after you have selected the zero suppression icon, which means that no zeros will be shown in the IBM Cognos TM1 cube. The view also shows no cells for “A*B” due to lack of FEEDERS. Notice the hover message “Feeders:(Unnamed).”

Figure 2 – Shows the dimension items “A” and “B” but not the calculated value “A*B” due to the lack of FEEDERS when zero-suppression is enabled

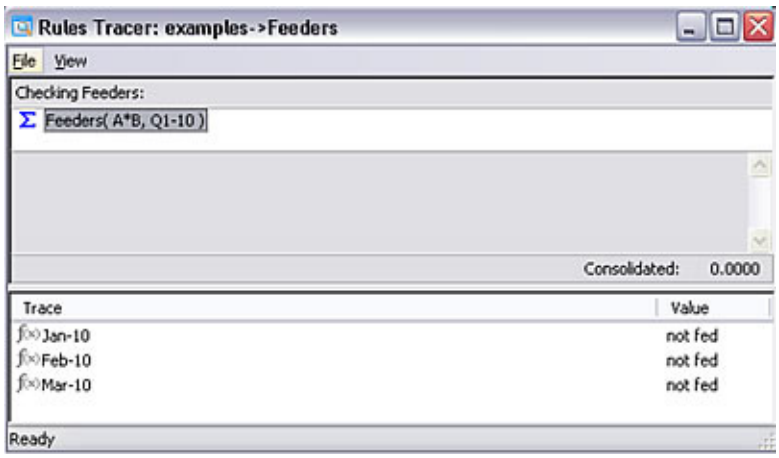


	Time:Default								
Feeders	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10	Jun-10	-- Q2-10	Jul-1
A Feeders:(Unnamed)	10	10	10	30	10	10	10	30	
B	20	20	20	60	20	20	20	60	

Method 4 – Use Check FEEDERS Option

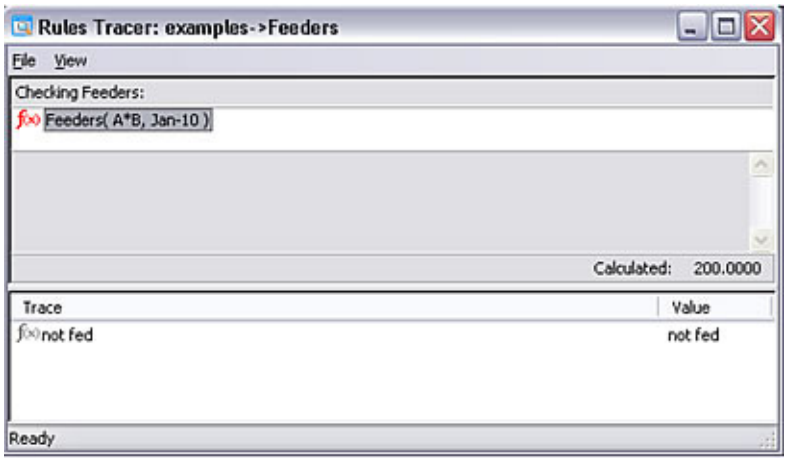
With this method, users can right-click on a calculation cell and select “Check Feeders…”. Figure 3 shows the IBM Cognos TM1 Rules Tracer dialog, which launches when the “Check Feeders…” option is selected, shows that checking “Feeders(A*B, Q1-Q10)” indicates that the values for Jan-10, Feb-10 and Mar-10 are not fed. In this example, the consolidation of these three items will not be shown correctly as the items are not being fed correctly.

Figure 3 - Shows the results of selecting “Check Feeders…” for the consolidated item Q1-10 that is missing FEEDERS



The next view shows the IBM Cognos TM1 Rules Tracer dialog for “Feeders(A*B, Jan-10)”. There is no FEEDER as the value for the “Feeders (A*B, Jan-10)” is showing not fed.

Figure 4 - Shows the results of selecting “Check Feeders…” for a detailed item Jan-10 that is not being fed

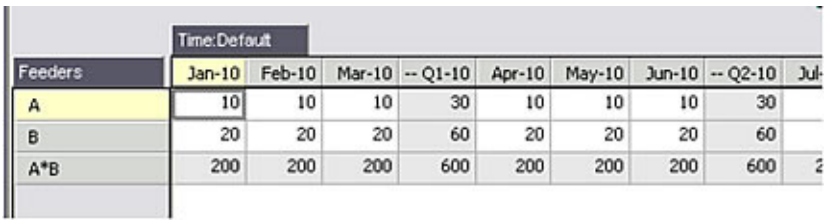


The following figure shows a view of an IBM Cognos TM1 cube as seen through IBM Cognos Cube Viewer after inserting the following FEEDER into the rule:

```
['A']=>['A*B'];
```

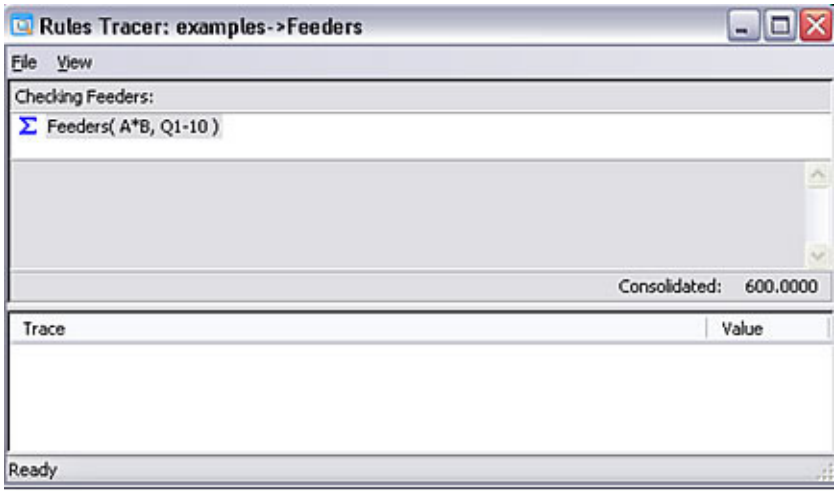
Note that the C: level calculations are now accurate for “A*B”.

Figure 5 Shows the results of “A*B” with FEEDERS defined



In choosing to invoke the “Check Feeders…” function, the IBM Cognos TM1 Rules Tracer dialog does not show any unfed cells for “Feeders(A*B, Q1-10)” as seen below. It also shows the aggregated calculation of 600.0000 for the C: level cell (A*B, Q1-10).

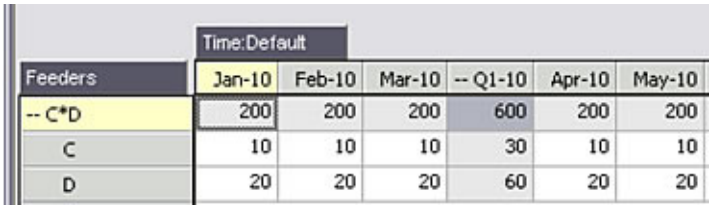
Figure 6 – Rules Tracer showing results of “Check Feeders…” with no unfed cells and with a correct consolidated value



To expand upon the previous example of feeding a calculation, recall that one of the five important considerations for FEEDERS is *Calculated C: level cells are automatically fed if their children are already fed*. The following is a simple example which illustrates this point. Figure 7 is showing two items “C” and “D” with the result “C*D” being derived via a rule. Notice that “C” and “D” rollup to “C*D” in the hierarchy. The rules to calculate “C*D” are as follows;

```
['C*D'] = c:if(ELLEV('Time', !Time) >= 1,Consolidatechildren('Time'),CONTINUE);
['C*D'] = c:['C']*['D'];
```

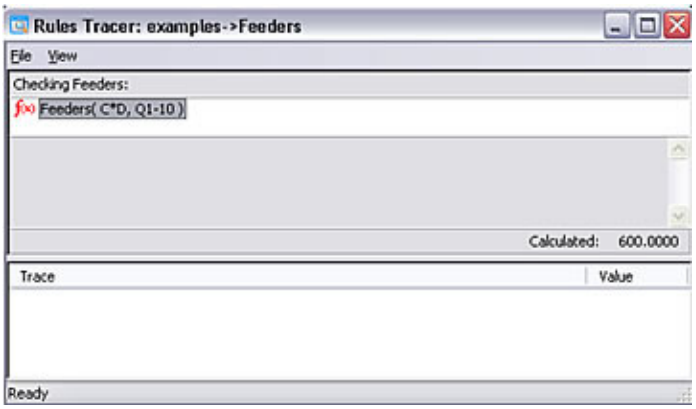
Figure 7 – Shows correctly calculated results for “C*D” without the use of FEEDERS



Time:Default						
Feeders	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10
-- C*D	200	200	200	600	200	200
C	10	10	10	30	10	10
D	20	20	20	60	20	20

The IBM Cognos TM1 Rules Tracer dialog in Figure 8 shows there are no FEEDERS being used to calculate (C*D, Q1-10). In this example, there is no need to define FEEDERS for “C*D” because both “C” and “D” both roll up into “C*D” in the hierarchy. As a result, “C*D” is automatically fed. The key point in this example is that in some situations the user can simplify their development by using the natural hierarchy within a dimension to drive C: level calculated values without needing to use a FEEDER.

Figure 8 – Shows no unfed items for “C*D” as it is a C: level item and is automatically fed from its N: level children



Finally, users should exercise caution when using the “Check FEEDERS” feature. This function examines all of the children of a C: level calculated cell to determine if it is fed or not. If you choose to use the feature on a top level summary, it may take a long time to return. This is because the system has to evaluate all of the children of all dimensions to determine whether the cell is fed or not.

Method 5 – Use an “OverFeeds” Cube

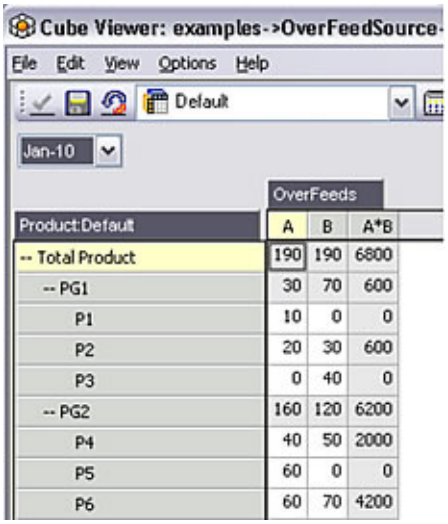
Use this method to determine if over-feeding of an IBM Cognos TM1 cube is occurring. Overfeeding can lead to overall poor IBM Cognos TM1 cube query performance.

Consider the example for an IBM Cognos TM1 cube called “OverFeedSource” where a user wants to determine if any of the cells in this cube are being over-fed. Here is the example rule being used in this cube:

```
SKIPCHECK;
['A*B']= n:['A']*['B'];
FEEDERS;
['A']=>['A*B'];
```

The view of the “OverFeedSource” cube in Figure 9 is showing two items “A” and “B” with the result “A*B” being derived via a rule.

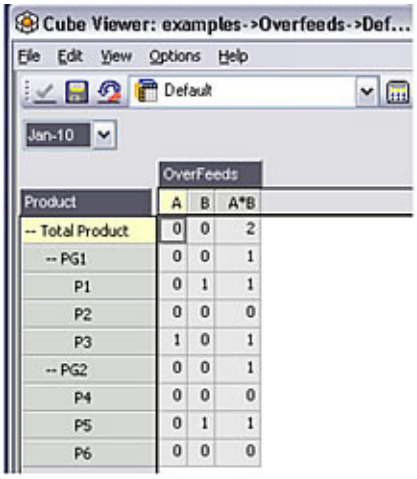
Figure 9 – Showing the results of “A*B” using “A” as the FEED source



The procedure used to verify if over-feeding is taking place is as follows:

1. Create a new cube called “Overfeeds” with the same dimensions as “OverFeedSource”.
2. Add a rule to “Overfeeds” as follows:
```
SKIPCHECK;
[ ] = n: IF(DB('OverFeedSource', !Time, !Product, !OverFeeds) = 0,1,0); FEEDERS;
```
3. Add a FEEDER to “OverFeedSource” as follows:
```
[]=>DB('Overfeeds', !Time, !Product, !OverFeeds);
```
4. Open the “Overfeeds” cube in the IBM Cognos TM1 Cube Viewer. The view is showing two items A and B with the result A*B being derived via a rule. The “Overfeeds” cube will show a value of “1” in every detailed cell that had a “0” in the source cube and a value of “1” in every consolidated cell that is being over-fed.

Figure 10 – Shows a value of “1” in every detailed cell that had a “0” in the source cube and a value of “1” in every consolidated cell that is being over-fed



The best way to understand how over-feeding in the “Overfeeds” cube is working is to consider the following table:

Table 1 – Analysis on how the IBM Cognos TM1 cube reacts to the use of FEEDERS

OverFeedSource Cube		OverFeeds Cube	
Value	**Fed?**	**Value at Leaf**	**Value at C**
0	No	1	0
0	Yes	1	1 (over fed)

In looking at summary levels for calculated fields any non-zero value will indicate an over-feed.

Figure 11 shows a view of the IBM Cognos TM1 cube Overfeeds where “A*B” for “P3” is overfed as it rolls up to “Q1-10” even though it is zero in the source cube. P3 for Jan-10 has been identified with 1, which indicates the cell has been overfed.

Figure 11 – Shows the over-feeding of P3 for Jan-10

	OverFeeds	Time
	A*B	
Product	Jan-10	-- Q1-10
-- Total Product	2	2
-- PG1	1	1
P1	1	1
P2	0	0
P3	1	0
-- PG2	1	1
P4	0	0
P5	1	1
P6	0	0

The explanation for the over-feeding is the construction of the FEEDER:

```
['A']=>['A*B'];
```

The system is using the value of “A” to determine whether it should feed “A*B”. It is ignoring the value of “B”. As a result, the system will feed “A*B” when “A”<>0 AND “B”=0, resulting in a zero “A*B”.

As will be shown later in another example, this is the normal way of feeding an IBM Cognos TM1 cube by using multiplication factors. It will lead to over-feeding, but typically not to such an extent where it severely impacts performance. Unless Conditional Feeders are used, over-feeding may not ever be completely eliminated from a Cube where multiplication, division, exponentiation or other operations take place, but it can be mitigated by feeding the variable that is most likely to be zero.

The normal solution to overfeeding is to use Conditional FEEDERS. Conditional FEEDERS set conditions on FEEDERS. This example can also be used to illustrate

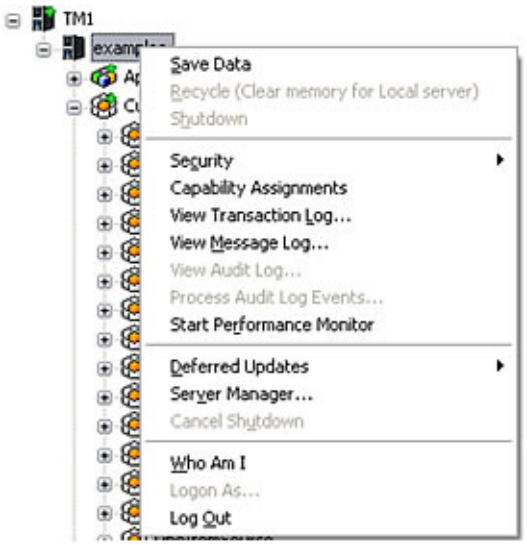
another FEEDER principle. Changing “A” to zero will still show “A” as over-fed in the “Overfeeds” cube. This is because once a cell is fed it is always fed until the TM1 Server is recycled or the TM1 TurboIntegrator function `CubeProcessFeeders()` is executed.

Method 6 – Use the Performance Monitor

This method will not specifically show which FEEDERS are over feeding, but it will give an idea of where to start looking. Often TM1 Developers are presented with a finished or partially finished model which is running slow and is using up a lot of memory.

Start the performance monitor in TM1 Architect by right-clicking on the TM1 Server name and selecting “Start Performance Monitor” from the menu as shown in Figure 12. Confirm that “Display Control Objects” as found under the View menu option has been enabled.

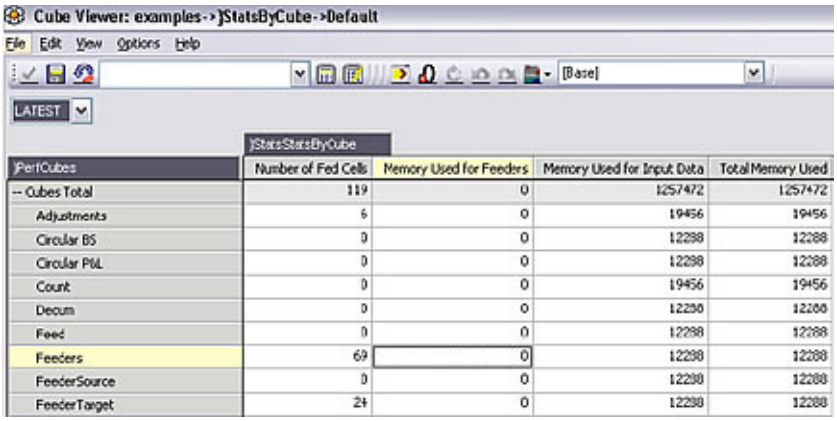
Figure 12 – The context menu displayed after right-clicking on a TM1 Server instance



Confirm that “Display Control Objects” as found under the View menu option has been enabled.

Open the “}StatsByCube” system cube in IBM Cognos TM1 Cube Viewer and notice it contains the following information on FEEDERS as shown in Figure 13. The “Feeders” line shows the number of fed cells and memory used by the FEEDERS under the columns titled “Number of Fed Cells” and “Memory Used for Feeders” respectively. Look for cubes with particularly large values under these columns. Use an “overfeeds” cube as described earlier in Method 5 to determine whether or not any calculations are being over-fed.

Figure 13 – FEEDER information from the TM1 Control Object }StatsByCube



Method 7 – Examine the TM1Server.log

As was the case in the previous method, this method will not directly list which FEEDERS are inefficient, but it will provide a good place to start looking. In the tm1server.log file, the system logs the loading of each of the cubes including the evaluation of the FEEDERS for each cube. The tm1server.log file is located by default in the TM1 Data Directory for the specific TM1 Server instance you are working with. The locations of TM1 Data Directories are user defined. Using the sample PlanSamp TM1 Server provided with the standard IBM Cognos TM1 install package, the tm1server.log file would be located by default in the following location: C:\Program Files\IBM\cognos\tm1\samples\tm1\PlanSamp\tm1server.log.

The following example shows selected lines from the tm1server.log file for the cube name “BW COST CALCULATION”:

```
3536 INFO 2012-05-16 23:41:22.580  TM1.Cube  Loading cube BW COST CALCULATION
3536 INFO 2012-05-16 23:41:22.611 TM1.Server TM1CubeImpl::ProcessFEEDERS: Computing
 FEEDERS for base cube 'BW COST CALCULATION'.
3536 INFO 2012-05-16 23:41:22.611 TM1.Server TM1CubeImpl::ProcessFEEDERS: Done computing
 FEEDERS for base cube 'BW COST CALCULATION'.
3536 INFO 2012-05-16 23:41:22.611 TM1.Cube Done loading cube BW COST CALCULATION
```

This information can be used to determine the time that it takes to evaluate the FEEDERS for each cube. If it takes a long time to evaluate the FEEDERS for a particular cube then that could be an indication that the FEEDERS contained in the cube are inefficient.

Defining FEEDERS

This section details most of the different types of calculations and shows how to construct the accompanying FEEDER. As a general rule, when picking an element to feed a calculation, pick the element that when zero the calculation’s results are also zero.

It is important to note that the way a FEEDER is defined depends on the type of calculation. In this section, a FEEDER strategy will be described for each of the following calculation types:

1. Multiplication
2. Division
3. Addition
4. Subtraction
5. Conditional
6. Cube-to-cube

If the calculation is a combination of the above, then a combination of the appropriate FEEDER strategies will be needed for each component of the calculation. As with rules, there are two ways of defining the FEEDER: 1) by enclosing the member name in square brackets; and 2) using the DB() function. There are examples of both in several of the following sub-sections.

Multiplication

```
['A*B'] = n: ['A']*['B'];
['A']=>['A*B'] ;
```

Multiplication is the easiest calculation to feed. In the above example, we are using “A” to feed the calculation. We could also have chosen “B”. We could have chosen either one as either a zero “A” or a zero “B” will force the calculation to be zero.

To further optimize the FEEDER, one should choose the element which is most likely to be zero. To further clarify this concept, we are going to use a typical Revenue calculation;

```
[‘Revenue’] = [‘units’]*[‘price’];
```

In this example, we would choose to feed “units” because “units” are most likely to be zero. For example, not all customers will purchase all products, so a lot of the combinations will be zero. “price” will most likely be non-zero and mainly fixed for all combinations of product and customer.

You can also define the FEEDER using the DB format.

```
['A']=> DB('FEEDERS', 'A*B', !Time);
```

It would not normally be necessary to use the DB format unless you were,

1. Defining cube-to-cube rules
2. Defining a conditional FEEDER
3. Manipulating the FEEDERS so that FEEDER elements match target elements

The DB method gives you more flexibility as you are able to embed conditional statements and IBM Cognos TM1 rule functions. Please refer to subsequent sections for further details and examples.

Division

```
['A/B']=n:['A']/['B'];
['A']=>['A/B'];
```

The same principles discussed in the Multiplication section apply to Division. Again, the choice of ‘A or ‘B’ doesn’t really matter when choosing which item to feed. If either item is zero then the calculation result will be zero or undefined.

Addition

```
['A+B']=n:['A']+['B'];
['A']=>['A+B'];
['B']=>['A+B'];
```

Here we have to feed both because a zero “A” will not necessarily force the calculation to be zero.

Subtraction

```
['A-B']=n:['A']-['B'];
['A']=>['A-B'] ;
['B']=>['A-B'] ;
```

As with Addition, we have to feed both because a zero “A” will not necessarily force the calculation to be zero.

Conditional Rules

To illustrate this, we have chosen an example that is commonly used in budgeting or forecasting applications. In the example below (Figure 14), we are performing a calculation for only those months that are flagged as forecast months (i.e. from “May-10” onwards, the forecast months have a grey background.

Figure 14 – Shows the results of using a Conditional FEEDER for the forecast months starting with May-10

	Time Default											
Feeders	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10	Jun-10	-- Q2-10	Jul-10	Aug-10	Sep-10	-- Q3-10
E	10	10	10	30	10	10	10	30	10	10	10	30
F	20	20	20	60	20	20	20	60	20	20	20	60
E*F (For Forecast Months)	201	202	203	606	204	200	200	604	200	200	200	600

For those months that are flagged as “Actual” (in this case “Jan-10” to “Apr-10”), we just want to upload or enter the result of the calculation, notice that the actual forecast months (row “E*F (For Forecast Months)” and months “Jan-10” through “Apr-10”) have a white background in their cells which means that these cells can accept manually entered values and are not the result of a calculation. The reason being is that for the actuals, the number is static and we don’t want the system to calculate it differently than the way that it is stored in the system of record.

In this example we are using an attribute on the time dimension to denote actual or forecast months. Figure 15 is a view of the Attributes Editor in which a new attribute

named “Actual Flag” has been added and a few specific months (“Jan-10” through “Apr-10”) have been denoted with an “A” to show which months contain actuals.

Figure 15 – The time dimension attribute named “Actual Flag (Text)” with “A” in the months denoted as actual data

	Actual Flag (Text)
Dec-09	
2010	
Q1-10	
Jan-10	A
Feb-10	A
Mar-10	A
Q2-10	
Apr-10	A
May-10	
Jun-10	
Q3-10	
Jul-10	

The rule is defined as follows:

```
['E*F (For Forecast Months)'] = n: IF( ATTRS('Time', !Time, 'Actual Flag')
 @<>'A',['E']*['F'],STET);
```

The FEEDER is as follows:

```
['E']=> ['E*F (For Forecast Months)'];
```

Cube-to-Cube Rules

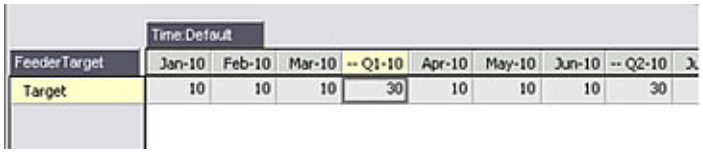
A simple example is used below to illustrate cube-to-cube rule FEEDERS as it is more complex and it introduces more challenges. Consider source and target cubes defined for the source cube “FeederSource”. The “FeederSource” cube has two dimensions “FeederSource” (with a single item named “Source”) and “Value” (with a single item named “Value”) and a single data value of 10 as shown in Figure 16.

Figure 16 – Shows the example FEEDER source cube named “FeederSource”

	Value
FeederSource	Value
Source	10

In Figure 17 below, we see a view of an IBM Cognos TM1 cube named “FeederTarget” showing values being fed from the source cube named “FeederSource”. The “FeederTarget” cube has two dimensions “FeederTarget” (with a single item named “Target”) and “Time” (with N: level items for each month and C: level items for quarterly consolidations “Q1-10”, “Q2-10”, etc.) and the N: level items have all been populated with the value from the “FeederSource” cube of 10.

Figure 17 – A view of an IBM Cognos TM1 target cube including values loaded using a FEEDER



	Time:Default								
FeederTarget	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10	Jun-10	-- Q2-10	Jı
Target	10	10	10	30	10	10	10	30	

The reason why the “cube-to-cube” FEEDERS are more complex is that you define the rule in the target cube and the FEEDER in the source cube. In effect the source cube is sending the FEEDERS to the Target cube and the rule in the target cube is receiving them. The situation is complicated by the fact that the dimension structures in the source and target cubes may be different. The following example will walk you through the process step by step:

Step 1 – Define the rule in the target cube

```
['Target'] = n: DB('FeederSource', 'Value', 'Source');
```

An important point here, which isn’t fully highlighted in the example, is that the DB function will have a parameter structure which pertains to the dimensional structure of the source cube (the first parameter will be the cube name, the second the first dimension in the source cube, the third, the second dimension in the source cube, etc.) However, the values that you supply in the parameter should be with respect to the target cube or a hard-coded value.

Since this is such an important point, we will illustrate it with a sub-example.

Suppose you have two cubes, “Cube S” and “Cube T”. Both have identical cube structures with identical dimensions, but the dimensions in each cube are a copy of the other. For example, the dimensions in “Cube S” are “Product S” and “Time S”. The dimensions in “Cube T” are “Product T” and “Time T”. The two product dimensions are identical and the two time dimensions are identical. Then the rule would be:

```
[‘Target’] = n: DB (‘Cube S’, ‘Source’, !Product T, !Time T);
```

Note that the dimensions in the target cube are substituted into the parameters of the DB function representing the structure of the source cube. This is an important point – TM1 Developers have spent many an hour staring at a rule trying to figure out why it wouldn’t save!

Step 2 – Define the FEEDER in the source cube

```
['Source'] => DB('FEEDERTarget', '2010', 'Target');
```

‘2010’ – It is necessary to hard-code a value here because there is no Time dimension in the source cube. By hard-coding a summary element from the time

dimension, it forces the system to feed all of the N: level items of 2010. Note that if the source cube did contain the Time dimension, then the FEEDER would be constructed as follows:

```
['Source'] => DB('FEEDERTarget', !Time, 'Target');
```

'Target' – It is necessary to hard-code the measure because there is a different measure element name in the source and target. We need to hard-code it to an element in the target measures dimension.

Here is the FEEDER for our sub-example:

```
[‘Source’]=> DB (‘Cube T’, ‘Target’, !Product S, !Time S);
```

The FEEDER here is defined as the reverse of the rule. Note that the dimensions in the source cube are substituted into the parameters of the DB function representing the structure of the target cube. This can cause some issues and the following list was assembled to highlight the best practices in designing cube-to-cube FEEDERS;

1. If the same dimension is in both the target and source cubes, simply use “*!DimensionName*” in that parameter.
2. If you have different dimensions, that are copies of each other, then you must use “*!DimensionNameInSource*” for that parameter.
3. If you have a dimension in the target that does not exist in the source, then you will need to hard-code a summary item in the target dimension. In the above example, we hard-coded ‘2010’. This means that the system will automatically feed all children of ‘2010’. This feature should be used with caution as it can lead to over-feeding scenarios and long server start up times.
4. If you have a dimension in the source that does not exist in the target, then you will need to hard-code an item in the source dimension.
5. If you want to target a particular element, then just hard-code it in the parameter. For example, ‘Revenue’ will just target the revenue measure. You can minimize the need to do this by using the same measure names in the source and target. That way you can just use “!TargetMeasureDimName”.
6. The FEEDERS that you send from the source cube must match the elements in the target cube. This applies to all dimensions. To illustrate this point, consider the following example. In feeding a weekly cube to a monthly cube, the weeks did not exist in the monthly cube, so nothing got fed. You can get around this by changing the FEEDER so that the parents of week are passed in the FEEDER. Since the parents of weeks are months and months are present in the target cube, the will FEEDERS will work as expected.

Cube-to-cube FEEDER avoidance

There are occasions where you can avoid defining the cube-to-cube FEEDER if the target measure is used in a subsequent calculation. In Figure 18, “Target” is the result of a cube-to-cube calculation, but is also used further in the calculation “Target*C”.

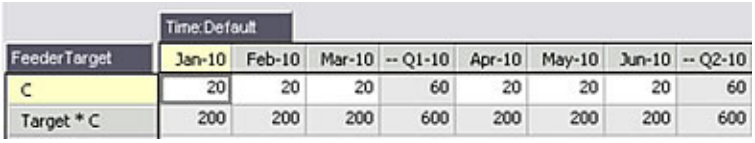
The value for “C” and “Jan-10” is 20. The calculated value of “Target*C” for “Jan-10” is 200 (“Target” has a value of 10 and “C” has a value of 20.) The IBM Cognos TM1 cube is loading values based on the following FEEDER formula:

```
['Target * C'] = n: DB('FeederSource', 'Value', 'Source') * ['C'];
```

You can remove the need for the cube-to-cube FEEDER by simply feeding “C” as follows:

```
['C'] => ['Target * C'];
```

Figure 18 – Shows target cube circumventing the need for a source cube FEEDER definition by simply using the source cube values in a calculation and feeding that calculation within the target cube



	Time:Default							
FeederTarget	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10	Jun-10	-- Q2-10
C	20	20	20	60	20	20	20	60
Target * C	200	200	200	600	200	200	200	600

Conditional FEEDERS

You can use conditional FEEDERS to reduce or eliminate over-feeding. Typically you would use a conditional FEEDER to accompany a conditional rule. To illustrate the point, consider the over-feeding example that we discussed earlier:

```
['A']=>['A+B' ];
```

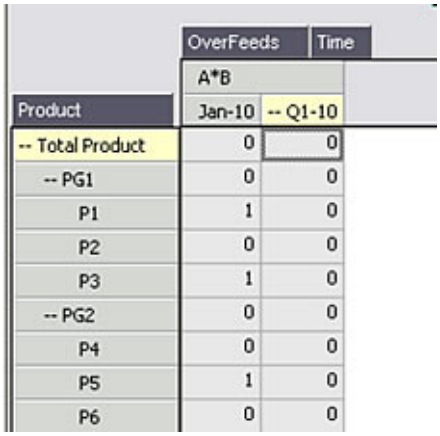
As we discussed before, the system is ignoring the value of “B” when it is constructing the FEEDER. You can construct conditional FEEDERS as follows to solve the problem:

```
['A']=>DB(IF(['B']=0,'','OverFeedSource'), !Time, !Product, 'A*B');
['B']=>DB(IF(['A']=0,'','OverFeedSource'), !Time, !Product, 'A*B');
```

Note that in order to conditionally feed, you need to use a DB function. In this case, you put an IF statement in the parameter of the DB function that represents the cube name. Return the cube name for cells that you want to feed and ‘’ (null) for ones that you don’t.

In looking at the cube, we now see that we are not over-feeding this calculation. Figure 19 is a view of the “OverFeeds” cube showing the results of Jan-10 and Q1-10 with the over-feeding corrected as illustrated by the values for Q1-10 not having any values equal to 2.

Figure 19 – Shows no values equal 2 in the Q1-10 consolidation field, which means there is no overfeeding of the IBM Cognos TM1 cube

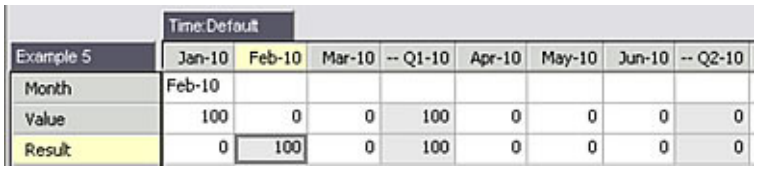


	OverFeeds	Time
	A*B	
Product	Jan-10	-- Q1-10
-- Total Product	0	0
-- PG1	0	0
P1	1	0
P2	0	0
P3	1	0
-- PG2	0	0
P4	0	0
P5	1	0
P6	0	0

How Often Do FEEDERS Fire?

According to the five important considerations specific to FEEDERS, FEEDERS from numeric cells fire only once and FEEDERS from String cells fire whenever their value changes. This is important when a parameter is used to determine the location of the calculated result. To illustrate this point Figure 20 shows an IBM Cognos TM1 cube with the value of 100 calculated at the intersection of “Result” and “Feb-10” highlighted:

Figure 20 – An IBM Cognos TM1 cube highlighting the value 100 at the intersection of “Result” and “Feb-10”



	Time:Default							
Example 5	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10	Jun-10	-- Q2-10
Month	Feb-10							
Value	100	0	0	100	0	0	0	0
Result	0	100	0	100	0	0	0	0

The rule is defined as:

```
['Result'] = IF(DB('FEEDERS', 'Jan-10', 'Month') @= DIMNM('Time', DIMIX('Time', !Time)),
 DB('FEEDERS', 'Jan-10', 'Value'),STET);
```

The FEEDER is:

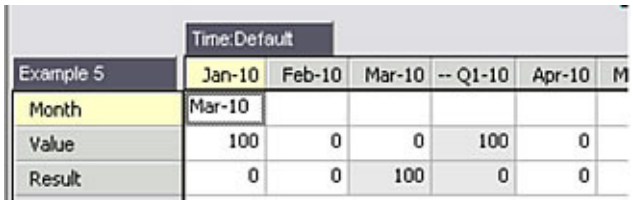
```
['Value','Jan-10']=> DB('FEEDERS', DB('FEEDERS', 'Jan-10', 'Month'), 'Result');
```

As you can see, we are using “Value” to feed “Result” in a rule where the parameter “Month” is being used to populate the correct month in the time dimension.

If we change the Month parameter by changing the field at the “Month”/”Jan-10” intersection from “Feb-10” to “Mar-10”, then the cell is no longer fed. This is illustrated in Figure 21 where “Q1-10” is zero instead of 100. This is because the FEEDERS for numeric cells fire only once. The cell “Result” for “Feb- 10” was fed initially, so “Feb-10” is the only cell that can be fed. The solution to this is to change the FEEDER to the following:

```
['Month','Jan-10']=> DB('FEEDERS', DB('FEEDERS', 'Jan-10', 'Month'), 'Result');
```

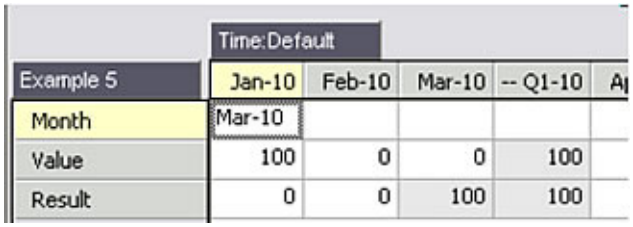
Figure 21 – Shows that feeding the cube with a numeric value can result in changes not being properly reflected



Example 5	Time:Default					
	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	M
Month	Mar-10					
Value	100	0	0	100	0	
Result	0	0	100	0	0	

Note that we are now using “Month” instead of “Value” to feed the calculation and because “Month” is a string, this will cause the FEEDER to fire each time that it changes. The following view of an IBM Cognos TM1 cube shows the correct results for “Q1-10” of 100.

Figure 22 – Shows that feeding the cube with a string value will display updated results as soon as a data field is changed



Example 5	Time:Default				
	Jan-10	Feb-10	Mar-10	-- Q1-10	A
Month	Mar-10				
Value	100	0	0	100	
Result	0	0	100	100	

Persistent FEEDERS

Persistent FEEDERS were introduced into IBM Cognos TM1 in version 9.5.1. The default value for this parameter is off but you may enable it by using the `PersistingOfFEEDERS` parameter in the TM1s.cfg file. To enable persistent FEEDERs and improve reload time of cubes with FEEDERS at TM1 Server startup, set the `PersistingOfFEEDERS` parameter to a value of “T” (true) to store the calculated FEEDERS to a FEEDERS file.

```
PersistingOfFEEDERS=T
```

When persistent FEEDERs are enabled and the TM1 Server encounters a persistent FEEDER file, it loads the saved FEEDERS which reduce the time normally taken to recalculate those FEEDERS. FEEDERS are saved when the data is saved or rules are edited. You do not explicitly save the FEEDERS.

For installations with many complex FEEDER calculations, persisting FEEDERS and then reloading them at server startup will improve performance. For simple FEEDERS, the time taken to read FEEDERS from disk may exceed the time to re-calculate the FEEDERS but most installations will benefit.

It is important to be aware that using persistent FEEDERS will increase your system size on disk only. Memory size is not affected by the use of persistent FEEDERS.

You need to be careful when developing applications using persistent FEEDERS. As mentioned earlier, the normal method for re-evaluating the FEEDERS is to restart the TM1 Server. If however you have persistent FEEDERS enabled, you will first need to run a TM1 TurboIntegrator process containing the following function in the prolog:

```
DeleteAllPersistentFEEDERS()
```

This will force FEEDER evaluation at TM1 Server startup rather than just reading from the persistent FEEDER cache.

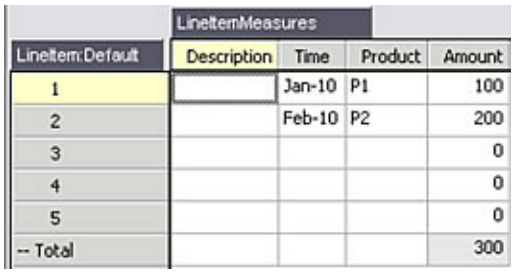
Complex FEEDER Examples

The complex FEEDER section details some more complex examples for real world situations.

Line Items Detail-to-Summary Cube

A common modelling problem, especially with budgeting and planning applications, is to link a line item detail cube to a summary cube where dimensions are pick-lists (a specific list of items such as products) in the source cube and real dimensions in the target cube. Consider the following input cube called “LineItemSource”, which includes Line Item, Description, Time, Product, and Amount as shown in Figure 23.

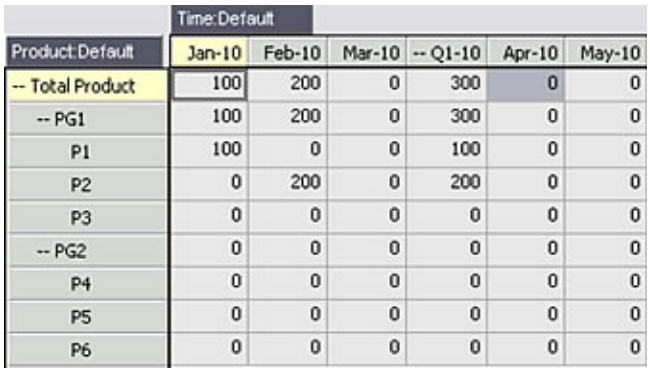
Figure 23 – Shows the input cube “LineItemSource”



	LineItemMeasures			
LineItem:Default	Description	Time	Product	Amount
1		Jan-10	P1	100
2		Feb-10	P2	200
3				0
4				0
5				0
-- Total				300

Figure 24 is a view of the summary cube “LineItemTarget”, which consolidates the values where the pick-lists become dimensions (i.e. Time is now columns and Product is now rows). This cube shows the detail line items as well as summary values at the Total Product level and the Time Quarterly level.

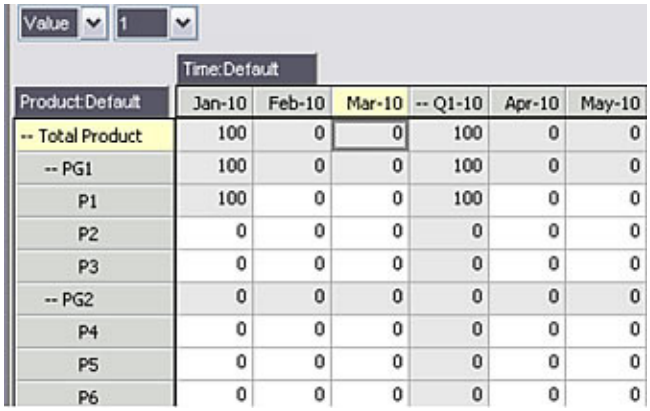
Figure 24 – The "LineItemTarget" cube view with dimensions created from the pick-lists in the "LineItemSource" cube



	Time:Default					
Product:Default	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10
-- Total Product	100	200	0	300	0	0
-- PG1	100	200	0	300	0	0
P1	100	0	0	100	0	0
P2	0	200	0	200	0	0
P3	0	0	0	0	0	0
-- PG2	0	0	0	0	0	0
P4	0	0	0	0	0	0
P5	0	0	0	0	0	0
P6	0	0	0	0	0	0

Since it is not possible to model this type of cube structure for directly transferring data from "LineItemSource" to "LineItemTarget", it is necessary to go through an intermediate cube called "LineItemCalc". The following view is the intermediate cube which contains all of the dimensions of both cubes – Item, Product and Time.

Figure 25 – The intermediate cube "LineItemCalc" which contains all the dimensions from both "LineItemSource" and "LineItemTarget"



Value 1

	Time:Default					
Product:Default	Jan-10	Feb-10	Mar-10	-- Q1-10	Apr-10	May-10
-- Total Product	100	0	0	100	0	0
-- PG1	100	0	0	100	0	0
P1	100	0	0	100	0	0
P2	0	0	0	0	0	0
P3	0	0	0	0	0	0
-- PG2	0	0	0	0	0	0
P4	0	0	0	0	0	0
P5	0	0	0	0	0	0
P6	0	0	0	0	0	0

The almost correct Rules and FEEDERS would be as follows:

For **LineItemSource**,

```
SKIPCHECK;
FEEDERS;
['Amount'] => DB('LineItemCalc',DB('LineItemSource', !LineItem, 'Product') , !LineItem,
 DB('LineItemSource', !LineItem, 'Time'), 'Value');
```

For **LineItemCalc**,

```
SKIPCHECK;
['Value' ] = n: IF(DIMNM('Product',DIMIX('Product', !Product)) @= DB('LineItemSource',
 !LineItem, 'Product') & DIMNM('Time',DIMIX('Time', !Time)) @= DB('LineItemSource',
 !LineItem, 'Time') ,DB('LineItemSource', !LineItem, 'Amount'),STET);

FEEDERS;
['Value'] => DB('LineItemTarget', !Product, !Time, !Value);
```

For **LineItemTarget**,

```
SKIPCHECK;
['Value'] = n: DB('LineItemCalc', !Product, 'Total', !Time, !Value);
FEEDERS;
```

Now let us go through the rules and FEEDERS one by one and explain.

First, let us start with the rule that converts the pick-lists to dimensions in the "LineItemCalc" cube:

```
['Value'] = n: IF(DIMNM('Product',DIMIX('Product', !Product)) @= DB('LineItemSource',
 !LineItem, 'Product') & DIMNM('Time',DIMIX('Time', !Time)) @= DB('LineItemSource',
 !LineItem, 'Time') ,DB('LineItemSource', !LineItem, 'Amount'),STET);
```

The components of the rule are:

1. **Bold** – check to see if the element name of the product dimension in the "LineItemCalc" cube matches the product entered in the "LineItemSource" cube. Note the use of "@=" as they are both strings.
2. *Italicized* – check to see if the element name of the time dimension in the "LineItemCalc" cube matches the month entered in the "LineItemSource" cube. Note the use of "@=" as they are both strings.
3. ***Bold Italicized*** – Return the value of "Amount" from the "LineItemSource" Cube if TRUE.
4. Do nothing if FALSE.

Now, let us look at the associated FEEDER in the "LineItemSource" cube.

```
['Amount'] => DB('LineItemCalc',DB('LineItemSource', !LineItem, 'Product') ,
 !LineItem, DB('LineItemSource', !LineItem, 'Time'), 'Value');
```

In looking at an initial design implementation, the FEEDER was defined as follows:

```
['Amount'] => DB('LineItemCalc', 'Total Product', !LineItem, '2010', 'Amount');
```

In recalling a previous tip in the **Cube-to-Cube Rules** section, if there is a dimension in the target cube that is not in the source, in this example Product and Time, you can hard-code a summary item into the appropriate parameter as shown in boldface above.

While this will work it may not be optimal for all situations as data volumes may negatively impact TM1 Server start-up times. This is because this type of FEEDER results in severe over-feeding as the system has to feed every single product and every single month for every single line item. Consider changing the FEEDER so that it only feeds the Product selected on any particular line item. For example, substitute **'Total Product'** with **DB('LineItemSource', !LineItem, 'Product')** and **'2010'** with **DB('LineItemSource', !LineItem, 'Time')**. In making these changes, the system start-up time will be improved.

An additional problem with the FEEDER in the “LineItemSource” cube is that it is feeding only once so it will not continue to work if we change either the product or month data. The solution is to feed the strings, as shown in the following example:

```
['Product'] => DB('LineItemCalc',DB('LineItemSource', !LineItem, 'Product') ,
 !LineItem, DB('LineItemSource', !LineItem, 'Time'), 'Value');

['Time'] => DB('LineItemCalc',DB('LineItemSource', !LineItem, 'Product') ,
 !LineItem, DB('LineItemSource', !LineItem, 'Time'), 'Value');
```

In this example, the FEEDER is repeated for each dimension (for both Product and Time). This ensures that the numbers correctly flow when both product and time are changed. So the correct rules and FEEDERS are;

For **LineItemSource**,

```
SKIPCHECK;

FEEDERS;

['Product'] => DB('LineItemCalc',DB('LineItemSource', !LineItem, 'Product') ,
 !LineItem, DB('LineItemSource', !LineItem, 'Time'), 'Value');

['Time'] => DB('LineItemCalc',DB('LineItemSource', !LineItem, 'Product') ,
 !LineItem,
DB('LineItemSource', !LineItem, 'Time'), 'Value');
```

For **LineItemCalc**,

```
SKIPCHECK;

['Value'] = n: IF(DIMNM('Product',DIMIX('Product', !Product)) @= DB('LineItemSource',
 !LineItem, 'Product') & DIMNM('Time',DIMIX('Time', !Time)) @= DB('LineItemSource',
 !LineItem, 'Time') ,DB('LineItemSource', !LineItem, 'Amount'),STET);

FEEDERS;
['Value'] => DB('LineItemTarget', !Product, !Time, !Value); LineItemTarget

SKIPCHECK;
['Value'] = n: DB('LineItemCalc', !Product, 'Total', !Time, !Value);
FEEDERS;
```

About the authors

Guy Jones

Guy Jones is a Client Technical Manager for the Cognos Performance Management solutions. He has been using TM1 for 5 years and has been extensively involved in developing complex proof of concept systems for customers.

John Leahy

John Leahy is a Proven Practices Advisor for IBM Business Analytics specializing in Financial Performance Management solutions and is an IBM Cognos TM1 Certified Developer.