

Revelwood Technical Bulletin

A Primer on SkipCheck and Feeders in TM1 Rules

Overview

The purpose of this technical bulletin is to provide TM1 developers and administrators with some insights into the use of SKIPCHECK and FEEDERS when writing rules. As you will see when you read through this bulletin, the use of SKIPCHECK and FEEDERS can be as much an art as it is a science. In other words, there isn't always one answer; however, the goal is to understand how to find the best answer for the problem at hand.

TM1 comes with its own custom coding language for calculations within and between cubes called 'Rules'. Rules provide a means for creating business logic calculations when you need to derive a value (intra-cube rule) or when you need to send data from one cube to another (inter-cube rule). By default, each rule will apply to every cross-section (data point) of the cube as defined in the Area portion of the rule. If the cross section of the rule is very sparse, this can consume too much processing time and memory on the server unnecessarily.

To eliminate the issue above, the rules language has two commands that will allow TM1 to calculate the rules only when it is required, which makes the rules calculate much more efficiently. The two commands are SKIPCHECK and FEEDERS.

The SKIPCHECK command will tell TM1 to 'skip checking' to see if a cross section with a value of zero has a rule associated with it. SKIPCHECK makes the rule more efficient by eliminating unnecessary calculations of the rule for sparse cubes. The issue with the SKIPCHECK command is that cross section values that are only the result of a rule will not calculate because the value is zero until the calculation is initiated.

To overcome this issue with the SKIPCHECK command, you must use the FEEDERS command every time you use SKIPCHECK. FEEDERS act like a flag for cross sections of cubes that tell TM1 to check the cross section for a rule calculation, regardless of the initial zero value, and calculate the rule that provides the value for the cross section.

This Technical Bulletin provides you with concepts to consider when writing your FEEDER statements.

When using the SKIPCHECK command, you must "feed" all rules in order for them to calculate properly. The trick is to be able to optimize the use of feeders for any given rule or set of rules. "Underfeeding" your rules may result in zero value results where you are expecting non-zero results. The cost of underfeeding is that data integrity will be compromised. On the other hand, you can also "overfeed" rules, which may result in time delays and memory explosion. So, it is important to understand how to properly design rules to avoid either under or over feeding.

Let's explore proper use of feeders through some basic examples.

Revelwood Technical Bulletin

PAGE 2 OF 5

Multiplication and Division Rules

For example, consider a simple multiplication rule such as: $['A'] = N: ['B'] * ['C'];$

When do you want this rule to calculate? The answer is whenever both B and C are non-zero. This is because if either B or C is zero the result is zero (any value multiplied by zero is zero). If the result should be zero then it does not matter if the rule is fed. A truth table for multiplication or division would look like this:

Variable	Scenario 1	Scenario 2	Scenario 3	Scenario 4
B	ZERO	VALUE	ZERO	VALUE
C	ZERO	ZERO	VALUE	VALUE
RESULT	ZERO	ZERO	ZERO	VALUE

The only time the result is a VALUE is when both *B* and *C* are non-zero (Scenario 4). So which variable should be the feeder to this rule? The answer is B or C.

If B were the feeder, the rule would get fed in Scenarios 2 and 4. If C were the feeder, then the rule would get fed in Scenarios 3 and 4. As long as Scenario 4 is fed the rule will work.

The decision to use B or C as the feeder is based on which variable has the fewest values.

For example, if B were the number of units sold and C were the price, more than likely B would be used since units sold tends to be more sparse than the price of a product. This is where your knowledge of the application and the data is crucial to creating an efficient model.

Addition and Subtraction Rules

Addition and subtraction are different. Let's analyze the rule: $['A'] = N: ['B'] + ['C'];$

When do you want this rule to calculate? The answer is whenever B or C are non-zero. This is because only when B and C are zero will the result be zero. If the result is zero then it does not matter if the rule is fed. So a truth table for addition and subtraction would look like this:

Variable	Scenario 1	Scenario 2	Scenario 3	Scenario 4
B	ZERO	VALUE	ZERO	VALUE
C	ZERO	ZERO	VALUE	VALUE
RESULT	ZERO	VALUE	VALUE	VALUE

Revelwood Technical Bulletin

PAGE 3 OF 5

The only time the result is ZERO is when **B and C** are zero (**Scenario 1**). So which variable(s) should be the feeder to this rule? The answer is both **B and C**.

If only **B** were the feeder the rule would only get fed in **Scenarios 2 and 4**. This would miss **Scenario 3**. If only **C** were the feeder then the rule would get fed in **Scenarios 3 and 4**. This would miss **Scenario 2**. Only by feeding both **B and C** will all required scenarios (**2,3 and 4**) be fed.

More Complex Rules

With simple mathematics problems, **B*C and B+C**, it is easy to break a rule down with a truth table to determine the feeders but calculations can be more complex.

Let's analyze the rule: **['A'] = N: IF(['B'] > 5, ['B'] + ['C'], ['D'] * ['E']) * ['F'];**

What you need to do in this case is break the problem down and determine when this calculation should result in a value. In essence, the resulting calculations are either **(B + C) * F** or **(D * E) * F** depending on the value of **B**. Since either calculation could occur, your feeders are the combination of the two calculations.

The first calculation feeders could be: **B and C**, or **F**. The second calculation feeders could be **D**, or **E**, or **F**. So the resulting feeders could be: **BCD, BCE, BCF, FD, FE, or F**.

Obviously **F** sounds like the logical choice, since it is a single variable. However, to prevent overfeeding you want to choose the feeders that are most sparse. If **F** is a very dense variable it would not be a good feeder. You may choose **BCD** or **BCE** based on your knowledge of the data model. Or you may actually choose **Z** as the feeder for the calculation above.

You are probably thinking, "but Z is not even part of the rule". A feeder is separate from a rule. The entire purpose of the feeder is to tag rules so that rules calculate properly at each cross section that is necessary. In complex modeling environments you can have 10, 20, 30, even 100 rules or more. Determining and maintaining the feeders for the entire model can be a time consuming challenge. In many cases you can use a controlling variable for your feeders.

For example, in a revenue model UNITS tends to be the driving force of the model. If no units are sold, many of the rule calculations are irrelevant and should not calculate, even if these calculations do not use UNITS directly. Finding a single controlling feeder variable can make the maintenance of the overall model much easier.

Revelwood Technical Bulletin

PAGE 4 OF 5

Special Considerations

C level rules are somewhat special. A C level rule is a rule that ONLY calculates when the cross section being requested for a variable has at least one C element. If the element the rule applies to has been fed, either via a value in one of its N elements or via a feeder to one of the N level elements, the C rule will calculate.

Feeders only feed the N level elements, and by feeding the N level elements the C elements dependent on the N level elements are by definition fed. For example, feeding YEAR will actually feed all 12 months (JAN-DEC) assuming the months are N elements rolling up into YEAR. Let's review the following two examples to illustrate this point:

1. If your feeder is ['Qtr1', 'Actual'] => ['Jan', 'Budget']; that has the same result as:

['Jan', 'Actual'] => ['Jan', 'Budget'];

['Feb', 'Actual'] => ['Jan', 'Budget'];

['Mar', 'Actual'] => ['Jan', 'Budget'];

2. If your feeder is ['Dec', 'Actual'] => ['Qtr4', 'Budget']; that has the same result as:

['Dec', 'Actual'] => ['Oct', 'Budget'];

['Dec', 'Actual'] => ['Nov', 'Budget'];

['Dec', 'Actual'] => ['Dec', 'Budget'];

If the rule is a C level rule then it should always be fed. With a C level rule it can be fed in two ways. If the N level elements below the rule are not rule driven then it may be valid to just let the values in the N elements feed the consolidation naturally and then the rule just overrides the natural consolidation.

An example of this would be a revenue model where the base measures in the model are UNITS, REVENUE and STDCOST. At a C level you still want to know the STDDCOST. So you create another measure EXTSTDCOST and create an N level rule ['EXTSTDCOST']= N: ['UNITS'] * ['STDCOST']. This rule needs to be fed using UNITS as the feeder so that it consolidates. At the C level you now want to create a rule to determine the STDCOST average. So you create the rule ['STDCOST']= C: ['EXTSTDCOST'] \ ['UNITS']. In this case the C level rule does NOT have to be fed since the N level elements associated with STDCOST have values they will naturally feed the consolidated elements.

You only need feeders if the Skipcheck command is active in the rule, otherwise TM1 assumes all cross sections are always calculated. So, it is possible to set up rules to calculate without feeders, not that this is recommended (due to overhead associated with it).

Revelwood Technical Bulletin

PAGE 5 OF 5

Also, you ONLY need to feed a value if you want it to consolidate or show up in a zero-suppressed browse.

In TM1 Server versions 8 and 9, you ask for a VIEW and then TM1 calculates the entire grid that you ask for. This will calculate some cells even though they are NOT fed properly. But when you zero suppress the view the first check TM1 does is to get the result of all non-zero values (real values and fed values). So, when a value is not fed it will not show in a zero-suppressed view. In addition, it will also not consolidate properly into C level calculations that are dependent on it.

If the rule is an N level rule then it only needs to be fed if the value needs to be consolidated into a parent element or will be used in a zero-suppressed view. You may ask yourself, when would either of these cases not be true? You may have a cube with rules in it where the rule results are only used by other N level rules and the cube is never really browsed. An example of this would be a time information cube where you have MONTHS and YEARS. You want to create a series of rules to give you FirstDayofMonth, LastDayofMonth in Julian form. You add a third dimension to the cube for the elements FirstDayofMonth and LastDayofMonth and create rules using the DAYNO() function. The resulting values will be used by other rules in the model and the results at an N level may be pulled into an Excel sheet but they do not need to consolidate and it really is not necessary to zero suppress a view of this cube. In this case the rules actually do not need to be fed.

Feeders ensure that rules properly calculate whenever needed and that results of rules consolidate properly. If you do not feed a rule, it may still calculate in certain situations.

How to Test That a Rule is Fed Properly

1. Create a VIEW at the same level as the rule (i.e. Base level N view etc.) with zero suppression turn off. This will prove that the rule itself works properly.
2. If that works, then zero suppress that view – all values should still appear. If not, the rule is not fed properly and is being suppressed out of the view.
3. If that works then look at a consolidation (if an N level rule) that is dependent on the rule and make sure the rule result is consolidating properly.

If you are concerned with the performance of your server due to rules being used in your analytical models (cubes) or if you have unique calculation requirements, give your Revelwood consultant a call to discuss how best to utilize the power of rules in your environment.

Revelwood Professional Services Group

Dan Bernatchez, VP PSG
Russ Cartisano, Senior Consultant