IBM Planning Analytics
Version 2 Release 0

TM1 Rules

Product Information

This document applies to IBM Planning Analytics Version 2.0 and might also apply to subsequent releases.

Licensed Materials - Property of IBM

Last updated: 2017-09-19

Contents

Introduction

This document is intended for use with IBM® Cognos® TM1® and the IBM Cognos Analytic Server (ICAS).

The *TM1 Rules* guide describes how to use rules to build an application that transforms business data in ways that provide insight not readily accomplished with raw data. This guide follows the transformation of data for a fictional company called Fishcakes International, which purchases different types of fish from markets around the world and uses that fish to make fishcakes. The company uses rules to calculate purchase costs, exchange rates, inventory levels, inventory depletion, and final production costs.

IBM Cognos TM1 integrates business planning, performance measurement and operational data to enable companies to optimize business effectiveness and customer interaction regardless of geography or structure. TM1 provides immediate visibility into data, accountability within a collaborative process and a consistent view of information, allowing managers to quickly stabilize operational fluctuations and take advantage of new opportunities.

Finding information

To find documentation on the web, including all translated documentation, access IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter).

Samples disclaimer

The Sample Outdoors Company, Great Outdoors Company, GO Sales, any variation of the Sample Outdoors or Great Outdoors names, and Planning Sample depict fictitious business operations with sample data used to develop sample applications for IBM and IBM customers. These fictitious records include sample data for sales transactions, product distribution, finance, and human resources. Any resemblance to actual names, addresses, contact numbers, or transaction values is coincidental. Other sample files may contain fictional data manually or machine generated, factual data compiled from academic or public sources, or data used with permission of the copyright holder, for use as sample data to develop sample applications. Product names referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products. IBM Cognos TM1 has some components that support accessibility features. IBM Cognos TM1 Performance Modeler, IBM Cognos Insight, and Cognos TM1 Operations Console have accessibility features.

Forward-looking statements

This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Chapter 1. Introduction to TM1 Rules

This section provides an introduction to IBM Cognos TM1 cube rules, which enable you to derive cell values through calculations.

The section includes basic information required to use rules.

Requirements for Running the Rules Editor

The Rules Editor is a .NET component that requires the installation of the Microsoft .NET 3.5 SP1 Framework on the client machine on which the Rules Editor will run.

An error message will appear if you try to run the Rules Editor on a system that does not have an installation of the Microsoft .NET 3.5 SP1 Framework.

To acquire the Microsoft .NET 3.5 SP1 Framework, see the Microsoft website.

Overview of Cube Rules

The most common calculation in OLAP applications involves aggregating data along a dimension. In TM1, you create these calculations using consolidation hierarchies. For example, in a Month dimension, you can define a quarterly consolidation that sums January, February, and March values.

In many applications, you also need to perform calculations that do not involve aggregating, such as generating cost allocations and calculating exchange rates. You can use rules to perform such calculations.

Using rules, you can:

- Multiply prices by units to yield sales amounts.
- Override consolidations when needed. For example, you can prevent a quarterly average price from displaying a tally of individual monthly prices.
- Use data in one cube to perform calculations in another cube, or share data between cubes. For example, you can pull sales data into a cube that contains profit and loss data.
- Assign the same values to multiple cells.

Each rule is associated with an individual cube. For example, if you create a rule to calculate values in the Purchase cube, the associated rule appears as a subordinate object of the Purchase cube in Server Explorer. The Purchase rule calculates values only in the Purchase cube.

Compiled rules are stored in files called *cube_name*.rux. When a cube for which you have defined rules is loaded into memory, TM1 searches for the cube's .rux file in the data directory containing the cube. The .rux file **must** reside in the same directory as the associated cube (.cub) file or else rules will not load properly.

When you create a rule, TM1 also generates a file called *cube_name*.blb, which contains format information for the Rules Editor.

If you want to edit a .rux file in another text editor, you should delete the corresponding .blb file. If you do not delete the file, there will be a discrepancy between the contents of the .rux file and the display in the Rules Editor, as the display of the Rules Editor is determined by the .blb file.

Note: When using rules and spreading, if the rules are such that the resultant value does not match the spread desired value, an error will be generated and the spread operation will not be done. See also the *Planning Analytics Installation and Configuration* guide SpreadingPrecision parameter for more information about spreading and rules.

Using Object Names with Special Characters in Rules Expressions

You should be aware that some special characters in object names may conflict when used in a rules expression. For example, the @ character and exclamation point ! character are both valid characters for object names but are also used in rules expressions.

The @ character is a string comparison operator in rules. If you reference any object containing the @ character in rules, the object name must be enclosed in single quotation marks. For example, a dimension named products@location must be referenced as 'products@location' in rules.

The exclamation point character is used as part of the !dimension argument in the rules DB function and should not be used in object names that will be used in rules.

```
DB('MarketExchange',!market,!date)
```

For more details, see the section, "Understanding TM1 Object Naming Conventions", in *TM1 for Developers*.

Accessing Rules

You create and edit rules in the Rules Editor. The Rules Editor is basically a text editor that helps you create accurate cube references, with menu options and toolbar buttons to insert commonly used rule syntax, characters, and functions.

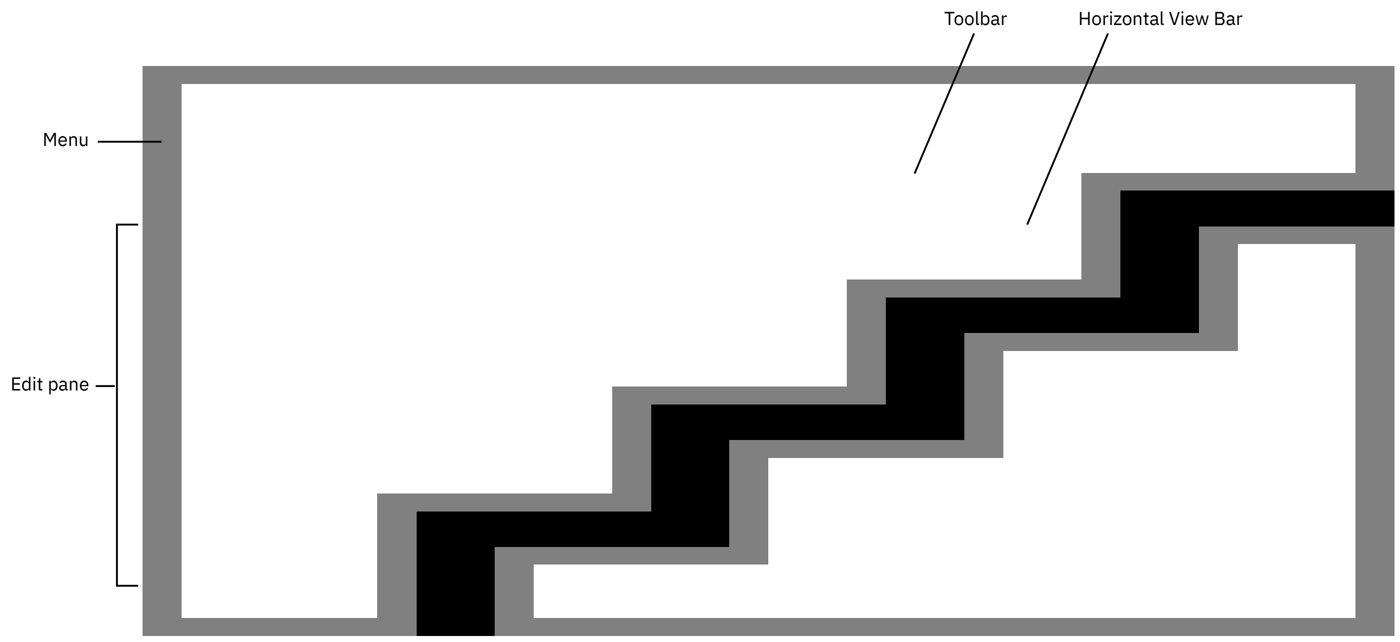
There are several ways to access the Rules Editor.

- When first creating a rule for a cube, right-click the cube in Server Explorer and click **Create Rule**.
- When a rule exists for a cube, right-click the cube in Server Explorer and click **Edit Rule**.
- When a rule exists for a cube, double-click the rule to open it for editing.

Note: Only one Rules Editor window can be opened for the same cube at a time.

Using the Rules Editor Window

The Rules Editor combines advanced code editing features with TM1 specific tools to help you create, manage, and verify your rules. This section provides an overview of the menus, toolbar buttons, and code editing features that you can use when working in the Rules Editor.



Note: You cannot type extended characters directly in the Rules Editor using the ALT key method with an English language keyboard. To enter extended characters in the Rules Editor, use a foreign language keyboard, such as French or German, or copy and paste the characters into the Rules Editor from another application such as MicrosoftWindowsNotepad.

Menus

The Rules Editor has a full set of menus for creating, editing, and managing rules. Keyboard shortcuts are provided for the more commonly used menu options.

File Menu

The following table describes the options in the File menu.

Name	**Description**
Import	Opens a file browse dialog so you can select a text file to import. Imported text will overwrite the current rule if one exists.
Save	Saves the current rule to the TM1 server .
Save As...	Saves the current rule to an external rule .rux file.
Check Syntax	Checks the current rule for syntax errors.
Print...	Opens the Print dialog box so you can print the current rule.
Print Preview	Opens the Print Preview window where you can view a sample printed version of the rule before sending it to a printer.
Exit	Closes the Rules Editor.

Edit Menu

The following table describes the options in the Edit menu.

Name	**Description**
Undo	Undoes the last edit. Multiple levels of undo are supported.
Redo	Reverses the last undo command.
Cut	Removes the selected text and places it in the clipboard.
Copy	Copies the selected text to the clipboard.
Paste	Pastes the contents of the clipboard into the Rules Editor.
Select All	Selects the entire contents of the Rules Editor.
Find	Opens the Find dialog box so you can search for text in the rule.
Find / Replace...	Opens the Find/Replace dialog box to search for and replace text.
Find Next	Locates the next occurrence of the text for which you are searching.
Toggle Bookmark	Turns a bookmark on or off for the current line of code.

Name	Description
Next Bookmark	Moves the cursor to the next available bookmark.
Previous Bookmark	Moves the cursor to the previous available bookmark.
Clear All Bookmarks	Removes all bookmarks.
Comment Selection	Adds a comment symbol # in front of all lines in the currently selected text to exclude the lines from the compiled rule. **Note:** Comment length is limited to 255 bytes. For Western character sets, such as English, a single character is represented by a single byte, allowing you to enter comments with 255 characters. However, large character sets, such as Chinese, Japanese, and Korean, use multiple bytes to represent one character. In this case, the 255 byte limit may be exceeded sooner and not actually allow the entry of 255 characters.
Uncomment Selection	Removes the comment symbol # from in front of all lines in the currently selected text to include the lines in the rule.
Indent	Indents the currently selected lines.
Unindent	Removes the indent from the currently selected lines.
Goto Line...	Displays the Go To Line dialog box so you can enter and jump to a specific line number in the Rules Editor.

View Menu

The following table describes the options in the View menu.

Note: Any changes you make to the settings on the View menu are saved when you exit the Rules Editor and are automatically re-applied the next time you open the Rules Editor.

Name	Description
Word Wrap	Turns on/off the word wrap feature so lines of text either extend to the right or wrap to display within the Edit pane.
Line Numbers	Turns on/off line numbers.
Function Tooltips	Turns on/off the display of function tooltips.
Auto-Complete	Turns on/off the auto-complete feature when typing in the Edit pane.
Toolbar	Turns on/off the display of the main toolbar.
Status Bar	Turns on/off the display of the status bar at the bottom of the Rules Editor.
Control Objects	Turns on/off the display of TM1 control objects when selecting cubes.
Expand All Regions	Expands all of the user-defined regions in the current rule to show all lines.

Name	Description
Collapse All Regions	Collapses all of the user-defined regions in the current rule to hide all lines that are included in a region.

Insert Menu

The following table describes the options in the Insert menu.

Name	Description
Function	Displays the Insert a Function dialog box to enter a new function into the current rule.
Cube Reference	Displays the Insert Cube Reference dialog so you can insert a DB function.

Tools Menu

The following table describes the options in the Tools menu.

Name	Description
Preferences...	Displays the Preferences dialog where you can set the font attributes such as font type, size, and color to be used in the Edit pane.
Options...	Displays the Control Options dialog where you can adjust the global settings for the Rules Editor.

Toolbars

Many of the functions in the Rules Editor have a toolbar equivalent.

The following table describes the Rules Editor toolbar icons.

Icon	Name	Description
📂	Import	Opens a file browse dialog so you can select a text file to import. Imported text overwrites the current rule if one exists.
💾	Save	Compiles and saves the current rule to the TM1 server .
✅	Check Syntax	Checks the current rule for syntax errors.
⬢	Insert Cube Reference	Displays the Insert Cube Reference dialog so you can insert a DB function.
fx	Insert Function	Displays the Insert a Function dialog box so you can enter a new function into the current rule.
[ ]	Brackets	Inserts a set of brackets [ ] into the rule so can create an area definition that references the current cube.

Icon	Name	Description
✂	Cut	Removes the selected text and places it in the clipboard.
⎘	Copy	Copies the selected text to the clipboard.
📋	Paste	Pastes the contents of the clipboard into the Rules Editor.
⊗	Delete	Deletes the selected text from the rule.
↶	Undo	Undoes the last edit. Multiple levels of undo are supported.
↷	Redo	Reverses the last undo command.
⇤	Unindent	Removes the indent from the currently selected lines.
⇥	Indent	Indents the currently selected lines.
🔍	Find	Opens the Find dialog box to search for text.
🖨	Print	Opens the Print dialog box so you can print the current rule.
?	Help	Displays help for the Rules Editor.

Status Bar

The status bar provides information when the Rules Editor is checking the syntax of a rule and when it is saving a rule.

When checking the syntax of a rule, the status bar displays the following message:

Checking Rule

If the rule syntax is correct, the status bar displays the following message:

Rule OK

If the rule syntax is *incorrect*, a warning message is displayed:

Rule Invalid

When saving a rule, the status bar displays the following message:

Saving Rule

After a rule is saved, the status bar displays the Rule Saved message until new edits are made. Once you start making new edits, the status bar no longer displays the Rule Saved message.

Rule Saved

Auto-save Feature

The Rules Editor includes an auto-save feature to prevent loss of changes if the Rules Editor exits unexpectedly.

By default, auto-save is configured to save a backup file of your rule every five minutes while you are working in the Rules Editor. If the Rules Editor exits unexpectedly, TM1 displays the following message the next time you open the same rule.

```
Recovery file exists.
Do you want to overwrite the
current rule with the contents of this file?
```

- If you click **Yes**, the contents of the auto-save backup file is loaded into the Rules Editor. You can then either save this version of the rule as the current version, or close the Rules Editor without saving and reopen it to use the original rule.

  Note: After recovering a backup file with the Rules Editor, the backup file remains on disk until you exit the Rules Editor window. On exit, the backup file is deleted and no longer available.
- If you click **No**, the original version of the rule is loaded into the Rules Editor and the backup file is deleted.

Adjusting the Auto-save Interval
You can change the auto-save feature by editing the file:

```
TM1RuleEditUserPrefs.xml
```

This file exists for the current Windows user and is located in:

```
C:\Documents and Settings\current_user\ApplicationData\Cognos\TM1\
```

where *current_user* is your Windows user name.

Open the TM1RuleEditUserPrefs.xml file and edit the `<autoBackupMinutes>` parameter to adjust the auto-save feature as follows:

```
<autoBackupMinutes>value</autoBackupMinutes>
```

where value is the interval, in number of minutes, when TM1 will save a backup file of a rule.

The default setting is 5 minutes, and a setting of 0 disables the auto-save feature.

For example: `<autoBackupMinutes>5</autoBackupMinutes>`

Location of the Auto-save Backup File
The auto-save feature saves your rule to a backup file with a name based on the name of the TM1 server and cube with which you are working.

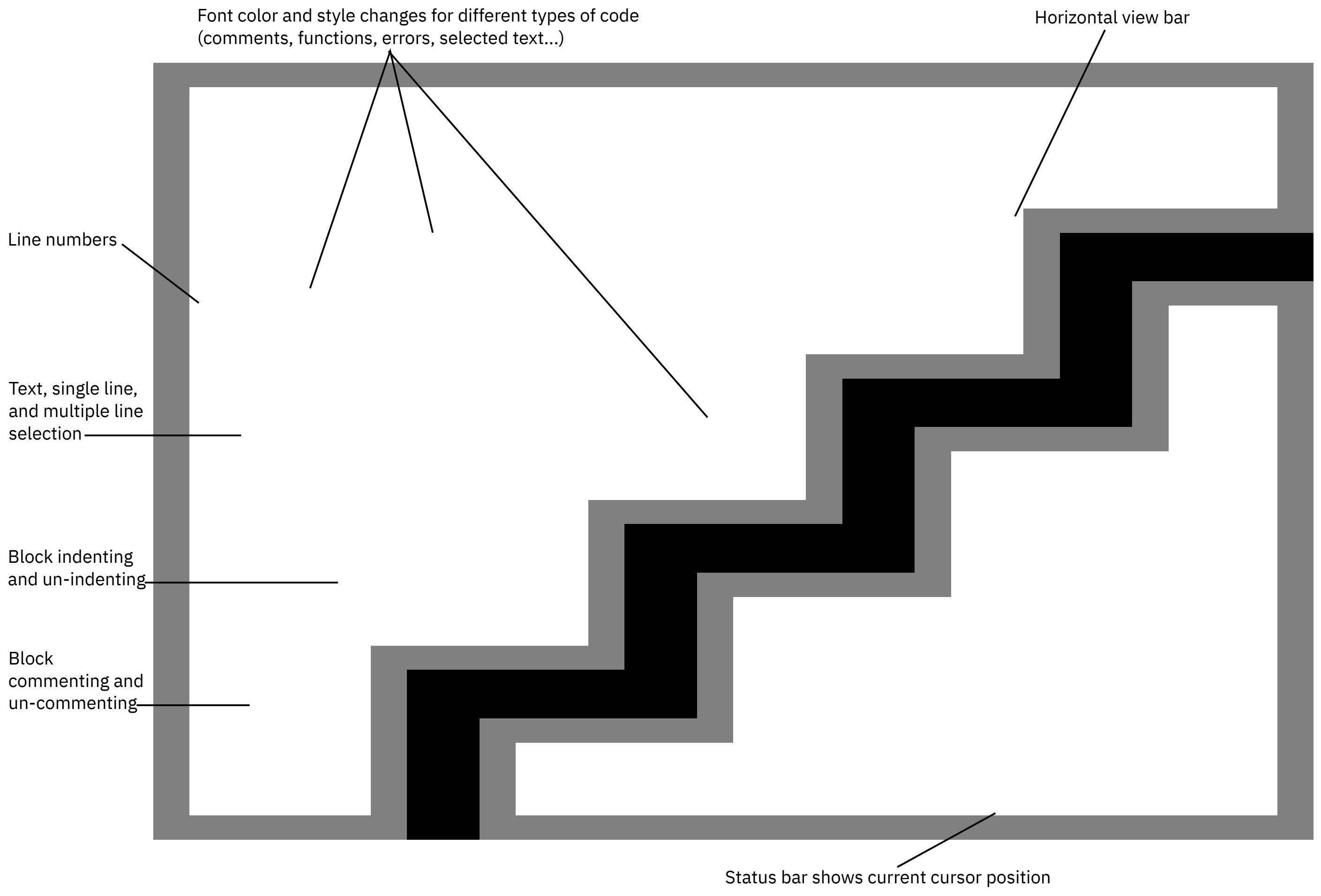
For example: _$planning sample$plan_Control.RUX

The backup file is located in the same location as the TM1RuleEditUserPrefs.xml file:

```
C:\Documents and Settings\current_user\Application
Data\Cognos\TM1\
```

Basic Code Editing Features

The Rules Editor provides basic text and code editing features, including line numbers, block indenting/un-indenting, and font color and style changes for different types of code.


Font color and style changes for different types of code (comments, functions, errors, selected text...)
Horizontal view bar
Line numbers
Text, single line, and multiple line selection
Block indenting and un-indenting
Block commenting and un-commenting
Status bar shows current cursor position

Other basic code editing features include:

- Word-wrapping of long code lines.
- White space indicators to show spaces, tabs, and paragraph return symbols.
- Printing that supports colors and formatting.
- Multiple levels of undo.

Using the Horizontal View Bar

You can divide the Edit pane into two views using the horizontal view bar. This can be useful when working in a long rule that requires a lot of scrolling, or when you want to copy code from one section of a long rule to another.

To split the Edit pane into two separate views, click and drag the horizontal view bar down.

Drag the horizontal view bar up and down to adjust the size of the two views.

To close the second view and return to a single view, drag the horizontal view bar back up to the Rules Editor toolbar.

Using the Context Menu in the Edit Pane

The Rules Editor displays either a standard or dynamic context menu when you right-click in the Edit pane. The specific menu that displays depends on the current location of the text-entry cursor.

- The standard context menu applies to most of the text in the Edit pane. This context menu provides shortcuts to frequently used commands in the main menu of the Rules Editor. Some of the menu options may or may not be available, depending on what is currently selected.
- The dynamic context menus that appear when you right-click in the edit pane provide quick access to the names of cubes, dimensions, elements, and functions. These context menus are described in the section "Advanced Code Editing Features" on page 9.

Advanced Code Editing Features

The Rules Editor includes a number of auto-complete typing, tooltip, and automatic highlighting features that help you enter and verify rules syntax.

Using Auto-complete to Insert a Function Name
As you type in the Edit pane, a list of possible rules keywords and functions displays, as determined by the letter(s) you type.

Note: You can re-display this list at anytime by pressing CTRL-Space when the cursor is positioned next to a letter.

You select a function from the drop-down list in one of the following ways:

- Select the function name from the list and then press the **Enter** key, or
- Double-click on the function name.

When you select a function from the list, the function and arguments are automatically inserted.

For example, selecting the `ATTRN` function inserts the text `ATTRN(dimension, element, attribute)` into your rule.

Using Auto-complete to Insert a Qualifier
When you right-click after a set of area definition brackets [], a context menu that contains a list of qualifiers displays.

To enter a qualifier from the drop-down list, you can either select it from the list and then press the **Enter** key, or double-click on the qualifier.

For more details on using a qualifier with an area definition, see "Inserting Brackets to Create an Area Definition" on page 12.

Using Dynamic Context Menus with the Cube, Dimension, Element, and Attribute Keywords
A dynamic context menu displays when you right-click on the following keywords in the Edit pane.

- cube
- dimension
- element
- attribute

If you have the Auto-Complete option selected in the **View** menu, then these keywords appear as argument place-holders when you select the functions they are used in from the Auto-Complete list.

For example, if you type `ATTRN`, and then press the Enter key, the ATTRN function is inserted into your rule with the dimension, element, and attribute keywords used as placeholders for the arguments.

If you right-click on the dimension keyword, a list of available dimensions displays.

Note: You can also type the cube and dimension keywords anywhere in your rule and then right-click on them to access their dynamic context menu.

Example of Using the Cube Keyword
This example illustrates that the DB function includes the cube keyword as an argument.

Procedure

1. Type DB and then press the **Enter** key.

   The DB function and its arguments appear.

   ```
   DB(cube, e1,e2,[...e256])
   ```
2. Right-click on the **cube** keyword.

   A list of cubes displays for the TM1 server to which you are currently connected.
3. Select a cube from the list and then press the **Enter** key to insert the cube name into the function. You can also double click a cube name in the list to insert it into the function.

The cube name displays in the function, replacing the cube keyword.

```
DB('salescube',e1, e2,[...e256])
```

Example of Using the Dimension, Element, and Attribute Keywords
The ATTRN and ATTRS functions use the dimension, element, and attribute keywords.

Procedure

1. Type ATTRN followed by the **Enter** key, and the dimension, element, and attribute keywords appear with the function.

   ```
   ATTRN(dimension, element, attribute)
   ```
2. Right-click on the **dimension** keyword to display a list of dimensions in the current cube.
3. Select a dimension from the list and then press the **Enter** key to insert the dimension name into the function. You can also double click a dimension in the list to insert it into the function.

   The dimension name displays in the function, replacing the dimension keyword.

   ```
   ATTRN('model', element, attribute)
   ```
4. Right-click the **element** keyword to select an element.

   The Subset Editor displays for the previously selected dimension.
5. Select an element from the Subset Editor and click **OK** to return to the Rules Editor.

   The element keyword is replaced with the element selected from the Subset Editor.

   ```
   ATTRN('model', 'L Series 1.8
   l Wagon', attribute)
   ```
6. Right-click on the **attribute** keyword to display a list of attribute names for the current dimension and element combination.

   Note: The specific attribute names that appear in this list depend on whether the function uses a string or numeric attribute. For example:

   - When using the ATTRN function, the names of only *numeric* attributes appear in the list.
   - When using the ATTRS function, the names of only *string* attributes are shown.
   - If there are no valid attributes for the specific dimension, a blank list displays.
7. Select an attribute from the list.

   The attribute keyword is replaced in the formula with the attribute name that you selected from the list.

   ```
   ATTRN('model', 'L Series 1.8
   l Wagon', 'Engine Size')
   ```

Viewing Tooltip Help for Function Syntax, Arguments, and Description
As you type a function name followed by an opening parenthesis, the editor displays a popup tooltip for the function syntax, arguments, and description.

When you hover your mouse over a function name that has already been entered in the Edit pane, a short description for the function displays in a popup tooltip.

Using Automatic Highlighting to Match Braces and Parentheses
To view pairs of opening and closing braces or parenthesis in rules, place the cursor at any brace or parenthesis symbol.

The Rules Editor highlights both the opening brace/parenthesis and the corresponding closing brace/parenthesis.

Using Find and Replace

Similar to other text editors, the Rules Editor includes tools for finding and replacing text.

Finding Text

The following step illustrates how to find text.

Procedure

To open the find dialog box, click **Edit**, **Find**, or press CTRL + F.

Replacing Text

The following step illustrates how to replace text.

Procedure

To open the Replace dialog box, click **Edit**, **Find/Replace**, or press CTRL + H.

Using Regular Expressions When Searching

Both the Find and the Replace dialog boxes include an option for using regular expressions when searching for text.

Procedure

1. In either the Find or the Replace dialog box, click the **Use regular expressions** check box.
2. Click in the Find text box where you want to insert a regular expression symbol.
3. Select the **Use regular expressions** check box and then click > to display the regular expressions menu.
4. Then click on a symbol in the menu to insert it into the **Find** textbox.

Using Bookmarks to Navigate Rules

Bookmarks provide an easy way to mark important lines in your rule and navigate back to them at a later time. Any number of individual lines can be bookmarked and bookmarks can also be removed.

Bookmarks are shown as indicators in the left side of the Edit pane.

Bookmark commands are accessible from both the main menu and standard context menu as follows:

- **Main menu** - Click **Edit** in the main menu of the Rules Editor.
- **Context menu** - Right-click in the Edit pane to display the standard context menu and then click **Bookmarks**.

You can manage and navigate bookmarks using the following menu options:

Menu Option	**Description**
Toggle Bookmark	Turns a bookmark on or off for the current line of code. Keyboard shortcut is CTRL+F2.
Next Bookmark	Moves the cursor to the next available bookmark. Keyboard shortcut is F2.
Previous Bookmark	Moves the cursor to the previous available bookmark. Keyboard shortcut is SHIFT+F2.
Clear All Bookmarks	Removes all bookmarks.

Using Regions to Define Sections of Code

A region defines a section of code, containing one or more lines of code, with a user-defined name. Regions can be expanded or collapsed to help you visually manage complex rules.

Regions are defined with the following syntax:

```
#Region RegionName
```

```
#endregion
```

After creating a region, you can click on the **+** or **-** symbol at the beginning of the region to expand or collapse the region.

Hover the mouse cursor over a collapsed region to show the contents as a tooltip.

Inserting Brackets to Create an Area Definition

Use the Brackets button as a quick way to create a rule statement. This button helps you create an area definition in your rule to reference elements in the cube with which the rule is associated.

Before you begin

For details on the syntax to use for bracket symbols and area definitions, see "Components of a Rule" on page 17

Procedure

1. Click in the Edit pane where you want the brackets inserted, and then click the Brackets button on the Rules Editor toolbar.

   A set of brackets are automatically inserted into the rule at the current text position and a list of dimensions for the current cube displays.
2. Select a dimension from the drop-down list in one of the following ways:
   - Select the dimension from the list and then press the **Enter** key, or
   - Double-click on the dimension.

   The Subset Editor displays so you can select one or more elements from the selected dimension.
3. Select an element in the Subset Editor and then click **OK**.

   The Subset Editor closes and the element name that you selected is automatically inserted into the Rules Editor.

   Note: If want to add another element in the brackets, enter a comma directly after the first element name to re-display the list of dimensions. For more details, see "Entering Multiple Element Names within Brackets" on page 12.
4. Right-click directly after the closing bracket to display a context menu containing a list of qualifiers to insert.

   A qualifier indicates whether the area definition applies to leaf/numeric cells, consolidated cells, string cells, or all cells. For more details on using a qualifier with an area definition, see "Components of a Rule" on page 17.
5. Select a qualifier in one of the following ways:
   - Select the qualifier from the list and then press the **Enter** key, or
   - Double-click on the qualifier.

   The qualifier is inserted after the set of brackets and you can now finish building the rest of the rule statement.

Entering Multiple Element Names within Brackets

You can enter multiple element names within a set of brackets using the Subset Editor or by entering a comma.

Using the Subset Editor to Enter Multiple Elements within Brackets
If you select more than one element in the Subset Editor, the names of the selected elements are inserted into the Rules Editor as a subset. The subset is enclosed in curly braces and each element name is separated by a comma.

```
[{'Salmon','Cod','Rock Oyster'}]
```

Using a Comma to Enter Multiple Elements within Brackets
You can add additional elements to the area definition by entering a comma directly after an element name in the brackets. This action re-displays the list of dimensions.

Procedure

1. Type a comma directly after an element name in the brackets.

   The list of dimensions re-displays.
2. Select a dimension from the drop-down list in one of the following ways:

   - Select the dimension from the list and then press the **Enter** key, or
   - Double-click on the dimension.

   The Subset Editor displays.
3. Select an element in the Subset Editor and then click **OK**.

   The Subset Editor closes and the new element name that you selected is automatically inserted into the existing brackets after the first element.
4. Repeat the above steps to enter additional element names into an existing set of brackets.

Inserting a Cube Reference

Use the Insert Cube Reference dialog to build and insert a `DB()` function into your rule. The `DB()` function references values from another cube on the same TM1 server , other than the cube with which the rule is associated.

Click the Insert Cube Reference button in the toolbar to open the Insert Cube Reference dialog.

Procedure

Click the Insert Cube Reference button in the toolbar to open the Insert Cube Reference dialog.

Inserting a Function

The following dialog boxes help you insert functions into your rule statements.

Using the Insert Function Dialog Box
The Insert Function dialog provides an accurate, easy way to find and insert a function into your rule with the correct arguments. This dialog displays a list of the available functions along with a brief description of each one.

Procedure

1. Use one of the following methods to open the Insert Function dialog:

   - Click **Insert**, **Function**, or
   - Click the **Insert Function** button on the Rules Editor toolbar.
2. To see a brief example of how to use a specific function, hover the mouse cursor over the description of the selected function.

   To see a full description of the function, click the **Help on this Function** link.

Using the Function Arguments Dialog Box
The Function Arguments dialog is dynamic and varies depending on the function you select.

Procedure

In the **Insert a Function** dialog box, select a function and click **OK**.

The Function Arguments dialog box displays. This dialog guides you through entering the arguments for the function, depending on the number and type of arguments required for the selected function.

Note: Some functions do not use arguments, and in this case, the function is inserted directly into the Rules Editor.

Using the Preferences Dialog to Adjust Font Schemes

The Rules Editor provides a default font scheme to help you visually identify the different categories of rules syntax when working in the Edit pane.

You can change the default font scheme using the Preferences dialog to select your own font style, size, color, and background. Examples of the different syntax categories include comments, errors, functions, and selected text.

Any changes you make are saved when you exit the Rules Editor so they can be re-applied the next time you edit a rule.

Note: The font *style* and font *size* apply to all text in the Edit pane and cannot be set for individual categories of rules syntax.

Using the Control Options Dialog

The Control Options dialog provides settings for the appearance and behavior of the Rules Editor. These settings are saved when you exit the Rules Editor and re-applied the next time you open the editor.

Configuring Appearance Options

The Appearance category includes settings for the Areas, Text, and Control options.

The following options are available for modifying the Areas appearance.

Area	Description
Indicator Area	Enables the display of the indicator margin area where bookmarks are displayed. By default, this option is enabled.
Word Wrap Area	Not used.
Selection Area	Enables a thin vertical area along the left margin of the Edit pane where you can click and drag to easily select the entire contents of multiple lines. By default, this option is enabled.
User Area	Not used.
Line Numbers	Displays line numbers on the left side of the Edit pane. By default, this option is enabled.
Changed Line Markings	Displays a yellow line in the left margin area to indicate lines that have been edited. By default, this option is disabled.

Setting the Areas Appearance Options

The following options are available for modifying the Areas appearance.

Indicator Area

Enables the display of the indicator margin area where bookmarks are displayed.

By default, this option is enabled.

Word Wrap Area

Not used.

Selection Area

Enables a thin vertical area along the left margin of the Edit pane where you can click and drag to easily select the entire contents of multiple lines.

By default, this option is enabled.

User Area

Not used.

Line Numbers

Displays line numbers on the left side of the Edit pane.

By default, this option is enabled.

Changed Line Markings

Displays a yellow line in the left margin area to indicate lines that have been edited.

By default, this option is disabled.

Setting the Text Appearance Options

The following options are available to modify Text appearance.

Text Option	**Description**
Lines Wrapping Marks	When the Word Wrap option is enabled, this option displays an indicator at the *end* of a wrapped line. By default, this option is enabled.
Wrapped Line Marks	When the Word Wrap option is enabled, this option displays an indicator at the *beginning* of a wrapped line. By default, this option is enabled.
Indentation Guidelines	When you click in a section of text that is contained within parentheses, this option displays a vertical line between the matching parentheses. By default, this option is enabled.
Indentation Block Borders	When you click in a section of text that is contained within parentheses, this option automatically highlights the text between the matching parentheses.
Column Guides	Not used.
Outlining Collapsers	Not used.
Transparent Selection	Controls whether selected text is highlighted with a translucent or solid color, as shown in the following figures. By default, this option is enabled for translucent text highlighting.

Setting the Control Appearance Options

The following options are available to modify the Control appearance.

Control Option	Description
Horizontal Scrollbar	Enables/disables the display of the horizontal scrollbar. By default, the horizontal scrollbar is enabled.
Vertical Scrollbar	Enables/disables the display of the vertical scrollbar. By default, the vertical scrollbar is enabled.
Status Bar	Enables/disables the display of the status bar at the bottom of the Rules Editor window. By default, this option is disabled.
XP Style	Controls the visual style of the Rules Editor user interface, independent of the operating system. By default, this option is disabled.

Configuring Behavior Options
The Behavior category includes settings for the General and Tabs options.
Setting the General Behavior Options
The following options are available to modify General behavior.

General Option	Description
Virtual Space Mode	Enables you to type anywhere in the Edit pane. When this option is enabled, you can type anywhere after the last existing character in a line, extending the line to the newly entered text. The Rules Editor automatically inserts the appropriate number of white spaces to extend the line. By default, this option is disabled.
Insert Mode	This option is equivalent to pressing the Insert key on the keyboard. When Insert Mode is enabled, the characters you type into the Edit pane are inserted without overwriting the existing text. When Insert Mode is disabled, the text you type into the Edit pane overwrites the existing text in the current position. By default, this option is enabled.
View White Space	Displays indicators to show spaces, tabs, and line feeds (returns). By default, this option is disabled.
Word Wrap	Displays the entire contents of each line within the Edit pane by wrapping text at the right edge of the Edit pane. By default, this option is disabled.
Group Undo	Groups similar consecutive actions, such as typing characters or deleting characters, so you can apply the undo action to the whole group. After typing in a word with this option *disabled*, the undo command will remove one character at a time. After typing in a word with this option *enabled*, the undo command will remove the entire word. By default, this option is enabled.

Setting the Tabs Behavior Options

The available options are available to modify Tabs behavior.

Tab Option	Description
Use Tabs	Controls whether a tab or an equivalent number of spaces is inserted when the Tab key is pressed in the Edit pane. By default, this option is enabled.
Tab Stops	Switches the size of a tab between a standard amount of seven spaces or a user-specified amount of spaces. If enabled, tabs are based on a standard amount of seven spaces. If disabled, tabs are based on the amount of spaces you specify with the Tab Size property.
Tab Size	Lets you specify the size of a tab in terms of spaces. This setting applies only when the Tab Stops property is disabled. When you change this setting, the new value is applied to all existing tabs in the edit pane, as well as new ones you enter.
Auto Indent Mode	Not used.

Components of a Rule

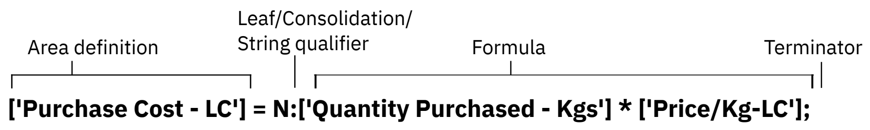
A rule is composed of one or more *calculation statements* that define how values in the associated cube are calculated.

Optionally, a rule can also contain *feeder statements* that optimize the performance of rules. When you use feeder statements in a rule, the rule must also contain a SKIPCHECK declaration and a FEEDERS declaration. The SKIPCHECK declaration must immediately precede any calculation statements in the rule, while the FEEDERS declaration must precede the feeder statements.

For details on feeder statements, SKIPCHECK declarations, and FEEDERS declarations, see Chapter 5, “Improving Performance with Feeders,” on page 49.

Components of a Calculation Statement

A calculation statement consists of the following components: area definition, leaf/consolidation/string qualifier, formula, and terminator.



```
['Purchase Cost - LC'] = N:['Quantity Purchased - Kgs'] * ['Price/Kg-LC'];
```

Area Definition

The area definition tells TM1 which values to calculate by a rule statement.

Using areas you can specify different calculations for different parts of a cube. For example, in a tax application, formulas for margin may vary from location to location, in which case you would specify different areas (and accompanying formulas) for each location.

The following table provides all valid area definitions for the Purchase cube in the sample database that accompanies this guide.

Area Definition	Scope
[ ]	All cells in the cube.
['Purchase Cost - LC']	All cells identified by the Purchase Cost - LC element.
['Jan-01', 'Price/Kg - LC']	All cells identified by both the Jan - 01and Price/Kg - LC elements. This area definition might reflect special discount pricing available only on New Years Day.
['Salmon', 'Helsinki', 'Jan-13']	All cells identified by the Salmon, Helsinki, and Jan-13 elements.

Regardless of the scope of an area, you must follow these conventions when writing an area definition:

- Enclose each element in single quotes.
- Use commas to separate each element.
- Enclose the entire area definition in square brackets.

Using Subsets in an Area Definition
You can use a subset in place of a single element in an area definition by enclosing all subset members in curly braces.

For example, the following area definition applies a calculation formula to all cube cells identified by the element Trout and any of the elements Karachi, Boston, or Helsinki:

```
['Trout', {'Karachi', 'Boston', 'Helsinki'}] =
```

Using Special Characters and Non-Unique Element Names in an Area Definition
You can use the syntax `'dimensionname':'elementname'` in an area definition to specify elements that are not unique to a single dimension or for dimension names that contain special characters.

For example, the area

```
['Units', 'Mar', 'Region':'North America']
```

lets you write a statement when the element North America is not unique to the Region dimension.

Similarly, the area

```
['}Groups':'ADMIN']
```

allows you to write a statement for the `}Groups` dimension, which contains the curly brace } special character.

Using the CONTINUE Function to Apply Multiple Formulas to the Same Area
When included as part of a calculation statement formula, the `CONTINUE` function allows a subsequent statement with the same area definition to be executed. Normally, TM1 only executes the first calculation statement encountered for a given area.

For example, if a rule contains the following two statements

```
['Jan']= if(!region @= 'Argentina',10,CONTINUE);['Jan']=20;
```

all cells identified by Jan and Argentina are assigned a value of 10. Cells identified by Jan and any other Region element are assigned a value of 20.

Leaf/Consolidation/String Qualifier
In TM1 , there is a distinction between leaf or numeric cells.

The leaf or numeric cell have coordinates which are *all* leaf/numeric (N:) elements in their respective dimensions; consolidated cells, which have at least one consolidated (C:) element among their coordinates; and string cells, use a string (S:) element as the last coordinate.

The leaf/consolidation/string qualifier indicates whether, for a defined area, a statement applies only to leaf cells, only to consolidated cells, only to string cells, or (in the absence of a qualifier) to all cells.

If the qualifier is `N:`, the statement applies to leaf cells only. If the qualifier is `C:`, the statement applies only to consolidated cells. If the qualifier is `S:`, the statement applies to string cells only.

If there is no qualifier, the calculation statement applies to all cells in the defined area.

Important: When you use a qualifier, it must immediately precede the formula component of a statement. A qualifier placed at the start of a statement will cause a syntax error.

Restricting a Statement to N: Level_Cells
Use the following syntax to write a rules statement that applies only to N: level cells in an area.

```
[Area] = N:[Formula];
```

For example:

```
['Sales'] = N:['Price']*['Units']\1000;
```

Restricting a Statement to C: Level Cells
Use the following syntax to write a rules statement that applies only to C: level cells in an area.

Here is an example of the syntax:

```
[Area] = C:[Formula];
```

For example:

```
['Price'] = C:['Sales']\['Units']*1000;
```

Calculating Values Differently at the N: and C: Levels
When a specific area of a cube is calculated differently at the N: and C: levels, you can use the following syntax.

```
[Area] = N:[Formula A]; C:[Formula B];
```

For example:

```
['Price'] =
```

```
     N:DB('PriceCube', !Actvsbud, !Region, !Model, !Month);
```

```
     C:['Sales']\['Units']*1000;
```

Formula
The formula portion of a calculation statement tells TM1 how to calculate a value for the defined area.
Numeric Constants
The simplest components of a calculation formula are numeric constants. A numeric constant consists of numerals, an optional leading sign, and an optional decimal point.

Examples of valid numeric constants are: 5.0, 6, -5. Examples of invalid numeric constants are: 1-, 1A, 3..4. Numeric constants have a maximum length of 20 characters. You can use scientific notation to enter a numeric constant.

The following rules statement uses a numeric constant to assign the value 200 to all cells in a cube:

```
[ ] = 200;
```

Arithmetic Operators
You can combine numeric constants using the following arithmetic operators.

Operator	Meaning
+ (plus sign)	Addition

Operator	Meaning
- (minus sign)	Subtraction
* (asterisk)	Multiplication
/ (forward slash)	Division
\ (back slash)	Same as / (forward slash), but returns zero when you divide by zero, rather than an undefined value.
^ (caret mark)	Exponentiation

TM1 evaluates arithmetic operators in the following order:

1. Exponentiation
2. Multiplication
3. Division
4. Addition
5. Subtraction

You must use parentheses to force a different order of evaluation. The expression 2*3+4 produces the same result as (2*3)+4 because multiplication takes precedence over addition. To perform the addition first, rewrite the formula as 2*(3+4).

This changes the result from 10 to 14.

Using Conditional Logic
Use the IF function to include conditional logic in calculation statements.

The general format is:

```
IF(Test, Value1, Value2)
```

The IF function returns one of two values depending on the result of a logical test. When the expression Test is true, the IF function returns Value1; when false, it returns Value2. The data type returned by an IF function is determined by the data types of Value1 and Value2. Value1 and Value2 must be the same data type, either string or numeric. An IF function where Value1 is a string and Value2 is a number yields an error statement.

You can also nest IF statements:

```
IF(Test1, Value1, IF (Test2, Value2, Value3))
```

The following table shows two IF examples.

Expression	Result
IF (7>6,1,0)	yields 1
IF (7>6, 'True', 'False')	yields 'True'

Using Comparison Operators
You can use the following comparison operators to compare values in the formula portion of a calculation statement.

Operator	Meaning
>	Greater than

Operator	Meaning
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=	Equal to
<>	Not equal to

To compare two string values, insert the @ symbol before the comparison operator, as in the following example:

```
IF ('A' @= 'B',0,1) yields the number 1.
```

Using Logical Operators
You can combine expressions in a calculation statement using logical operators.

Operator	Meaning	Example
& (ampersand)	AND	(Value1 > 5) & (Value1 < 10) Returns TRUE if the value is greater than 5 and less than 10.
% (percentage sign)	OR	(Value1 > 10) % (Value1 < 5) Returns TRUE if the value is greater than 10 or less than 5.
~ (tilde)	NOT	~(Value1 > 5) Equivalent to (Value1 <= 5)

Concatenating Strings
You can concatenate strings using the pipe | character.

For example, the following expressions returns Rheingold.

```
(Rhein | gold)
```

If the string resulting from a concatenation is longer than 254 bytes, TM1 displays an error message.

Using Cube References
The formula portion of a calculation statement can contain cube references, which retrieve values from a cube.

The formula portion can retrieve values from the cube for which you are writing a rule (internal cube references) or from other cubes on the same TM1 server (external cube references). Cube references are reviewed in great detail later in this book.

Internal Cube References
Internal cube references use the same syntax as the area definitions.

Examples include:

```
['January']
```

```
['Sales','January']
```

```
['Germany','Sales','January']
```

In the following example, Gross Margin for Germany is calculated by multiplying Sales for Germany in the same cube by 0.53:

```
['Gross Margin','Germany']=['Sales']*0.53;
```

External Cube References
Use the DB function to retrieve values from external cubes.

```
DB('Cube', DimaArg1, DimArg2,...DimArgn)
```

Cube	Name of the external cube
DimArg	Specify a `DimArg` argument for each dimension of the external cube. The `DimArg` arguments must be ordered to correspond to the order of dimensions in the external cube. One of the following arguments: The name of an element in the appropriate dimension of the external cube, enclosed in single quotes. The name of the appropriate dimension preceded by an exclamation mark (!). This is called variable notation, which indicates that the reference applies to the element in the external cube that corresponds to the value currently being requested. For example, if a DB function is used in a formula that calculates a value for Belgium, the argument `!Region` resolves to Belgium. An expression that resolves to an element in the appropriate dimension.

Terminator

The semicolon indicates the end of a rules statement, whether a calculation statement or a feeder statement, and is mandatory.

You can distribute statements over many lines, as long as each statement ends with a semicolon (;). You will see many long calculation statements formatted this way as you develop complex applications in later sections of this book.

Referencing a String in a Calculation Statement

If a calculation statement references a string element, you must use a DB function to retrieve the string, even if the string resides in the same cube for which you are writing a rule.

Numerical Attributes

Numerical attributes are considered special elements and rules for these elements are not allowed.

To use a rule to populate the cells, create a text attribute. The string values can then be converted to numeric data as needed using the NUMBR function.

Bypassing Rules

By using the STET function, you can bypass the effect of a calculation statement for specific areas of a cube.

For example, you may want to write a statement for Gross Margin that applies to all regions except France. You can write the general rule and the exception in two ways.

- Write the STET statement first followed by the general rule:

```
['Gross Margin', 'France']
= STET;
```

```
['Gross Margin'] = ['Sales'] * 0.53;
```

- Write one rules statement that includes an IF function:

```
['Gross
Margin'] = IF(!Region @= 'France', STET, ['Sales']* 0.53);
```

General Considerations

Here is additional information about bypassing rules.

- The rules syntax is not case-sensitive. You can use both uppercase and lowercase letters.
- You can use spaces within rules to improve clarity.
- A rules statement can occupy one or more lines in the Rules Editor. It can also contain one or more formulas. End each statement with a semicolon.
- To add comments and to exclude statements from processing, insert the number sign at the beginning of a line or statement. For example:

```
#The
following rule is not active# ['Gross Margin']=['Sales']*0.53;
```

  The length of a comment line is limited to 255 bytes. For comments longer than 255 bytes, you must break up the comment into multiple lines, with each one including the number sign, #, at the beginning.

  Note: For Western character sets, such as English, a single character is represented by a single byte, allowing you to enter comments with 255 characters. However, large character sets, such as Chinese, Japanese, and Korean, use multiple bytes to represent one character. In this case, the 255 byte limit may be exceeded sooner and not actually allow the entry of 255 characters.

How Rules Work

The following sections describe how rules work within TM1 and provide considerations you should keep in mind when developing rules.

Calculate on Demand

TM1 calculates values only when requested, or "on demand."

TM1 uses very efficient data compression to allow large data sets to fit in relatively small amounts of RAM, resulting in improved performance and reduced storage requirements.

When a value is requested from a location in a cube, the TM1 server checks if the location corresponds to the area definition of any calculation statements associated with the cube. If the location does correspond to a statement, TM1 evaluates the formula portion of the calculation statement and returns a value to the relevant area.

Note: TM1 only calculates a value for each cube location once, and the first formula for a given area takes precedence over later formulas for the same area. If you have multiple rules statements that address the same area, you should order them most-specific to least-specific according to area definition. TM1 provides a rules function named CONTINUE to address the rare instances when you want to apply multiple formulas to the same area. For details on the CONTINUE function, see "Using the CONTINUE Function to Apply Multiple Formulas to the Same Area" on page 18 and the Rules Functions chapter of *TM1 Reference*.

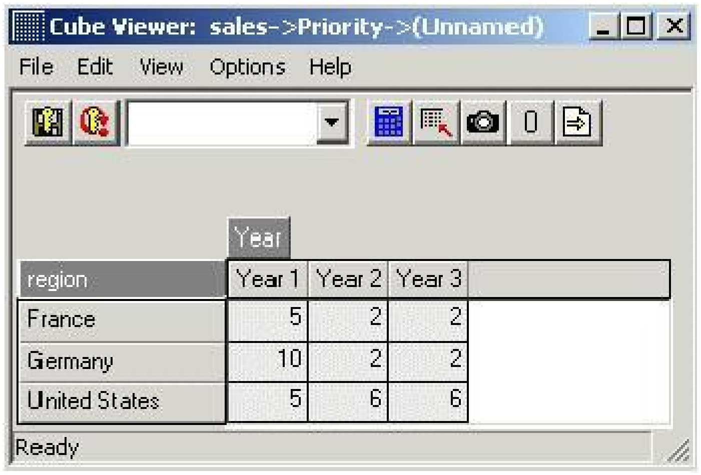
Precedence of Rules Statements

When more than one statement in a set of rules applies to the *same* area, the first statement takes precedence.

Consider this example. A cube named Priority has two dimensions, Region and Year. The rule for the Priority cube contains the following four statements:

```
['Germany', 'Year1'] = 10;['Year1'] = 5;['United States']=6;[
] = 2;
```

Here are sample values for the Priority cube, all of which are derived by the calculation statements.


Cube Viewer: sales->Priority->(Unnamed)
File Edit View Options Help
Year
Ready

region	Year 1	Year 2	Year 3
France	5	2	2
Germany	10	2	2
United States	5	6	6

TM1 processes the statements as follows:

- The first statement assigns the value 10 to the Germany, Year1 cell. It takes precedence over the second statement, which specifies that all Year1 cells contain 5.
- The second statement takes precedence over the third statement. Therefore, the cell for United States, Year 1 contains 5, even though the third statement specifies that all values for United States should be 6.
- The last statement `[ ] = 2` specifies that all values in the cube contain the value 2. This statement applies to all cells that are not affected by the statements, such as France, Year2.

Rules and Dimension Consolidations

Rules work in concert with consolidations defined in dimensions.

Although you can define consolidations using rules, this is not recommended for performance reasons. Consolidations defined in dimensions process much faster than those defined in rules, especially in very large, sparse cubes.

Order of Calculation

Rules take precedence over consolidations within dimensions.

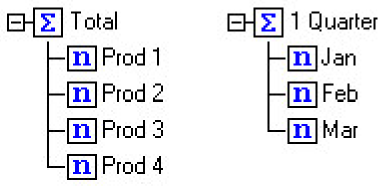
When a cube cell is calculated by a rule and a consolidation, the rules statement is examined first. However, if the rules statement refers to cells that are the result of consolidations, TM1 first performs the required consolidations and then calculates the rules statement using the results.

Conversely, if a cell is defined by consolidation only, TM1 looks at the values needed to perform the consolidation. When some values are the result of rules calculation, TM1 performs the required rules calculation before performing the consolidation.

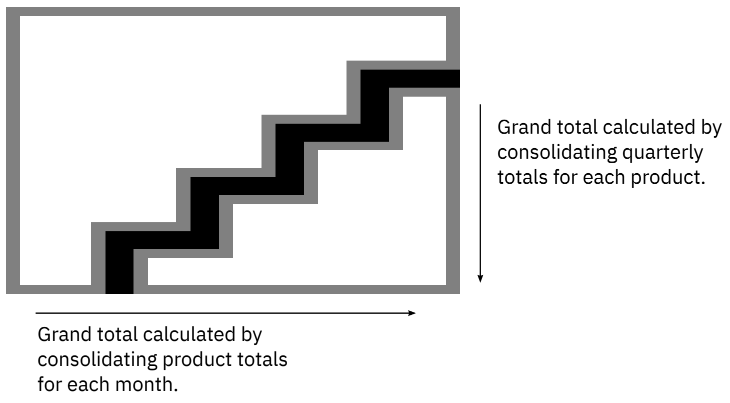
Overriding C: Level Elements with Rules

You should avoid writing a rule that overrides a consolidated value that is a component of another consolidation.

A simple example illustrates this issue. Suppose you have a two-dimensional cube named Sales that is composed of the dimensions Product and Month, with product (Total) and quarterly (1 Quarter) consolidations defined.



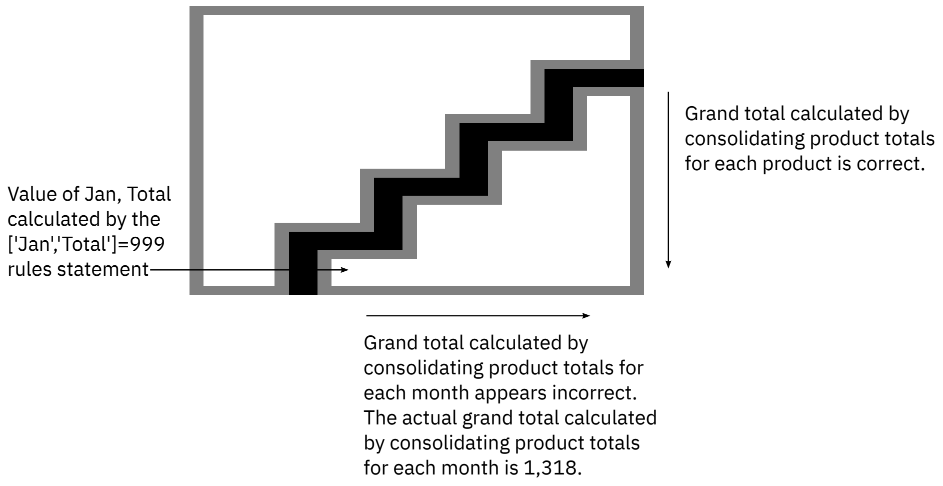
To calculate the grand total (Total, 1 Quarter), TM1 can consolidate the product totals for each month or consolidate quarterly totals for each product.


Grand total calculated by consolidating quarterly totals for each product.
Grand total calculated by consolidating product totals for each month.

Suppose further, you write a rule that calculates a value for Total product sales in Jan, and that the rules-calculated value does not sum the individual product values for Jan. A rule that defines the value of Total products in Jan as 999 serves as an illustration.

```
['Jan','Total']=999;
```

If the grand total is calculated by consolidating the product totals for each month, the value will differ from the consolidation of the quarterly totals for each product. This is because the rules-calculated value for total product sales in Jan overrides the natural consolidation defined in the Product dimension.

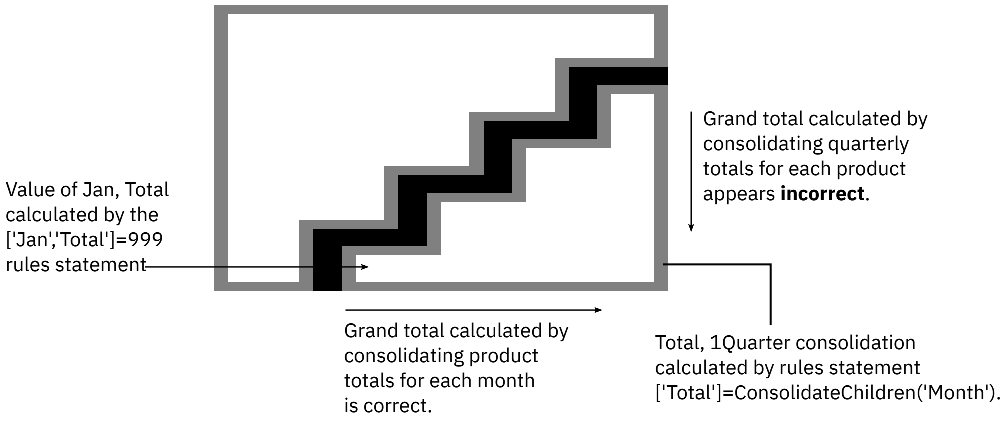

Grand total calculated by consolidating product totals for each product is correct.
Value of Jan, Total calculated by the ['Jan','Total']=999 rules statement
Grand total calculated by consolidating product totals for each month appears incorrect. The actual grand total calculated by consolidating product totals for each month is 1,318.

You have no control over the order in which TM1 performs dimension consolidations. Furthermore, depending on which consolidation path is optimal at any given moment, TM1 may alternate between paths. Consequently, you may request the Total, 1 Quarter value twice in the same session and get different results.

You can remedy this situation by writing a rules statement that calculates the value of the Total, 1 Quarter consolidation as the sum of its immediate children along the Month dimension, thereby overriding the Product dimension consolidation. The statement

```
['Total']=ConsolidateChildren('Month')
```

performs this calculation.


Value of Jan, Total calculated by the ['Jan','Total']=999 rules statement
Grand total calculated by consolidating quarterly totals for each product appears **incorrect**.
Grand total calculated by consolidating product totals for each month is correct.
Total, 1Quarter consolidation calculated by rules statement ['Total']=ConsolidateChildren('Month').

However, there remains an implicit inconsistency when viewing the cube: the sum of the quarterly totals for each product is different from the sum of product total for each month. Thus, overriding `C:` level values that are components of other consolidations is not recommended.

Stacking Rules
A rules statement can refer to a cube cell that is defined by another rules statement, and so on.

TM1 stacks rules statements until it can obtain a final value, and then works back to return a result. The number of levels of stacking that can be accommodated is limited only by available memory.

If a circular reference occurs within a rules stack, or the maximum level of stacking is exceeded, TM1 returns the error message:

```
Error Evaluating Rule: Possible Circular Reference
```

Here is an example of a circular reference:

```
['Sales'] = ['Units'] * ['Price'] ;
```

```
['Price'] = ['Sales'] / ['Units'] ;
```

Comparing Floating Point Numbers

Due to the "floating point" representation of numbers, some numbers such as .35 turn into .34999999999999998 when stored in a binary format.

Typically this accuracy is adequate for normal processing. But, when a model represents percentages as fractions, this loss of precision means the aggregation adds up to .99999999999999998 instead of 1.0.

If a rule is triggered when the aggregate of the percentages is 1 (100%), then most combinations of percentages will trigger the rule. But, in some cases due to the floating point representation, the totals do not completely add up to 1, so the 1.0 total is never reached and the rule is not triggered.

To avoid this problem, you can represent percentages as whole numbers (35 instead of 0.35 for 35%) or rework the rule so that the comparison between the aggregation and 1 doesn't have to be exact.

Memory Utilization and Rules

TM1 does not store rules-derived values on disk or load such values into memory when a cube is initially loaded. Instead, when you request a cube value.

TM1 checks to see if there is any rules statement associated with that value.

TM1 checks this by using a very fast algorithm that examines all the rules statements against the key for the value being retrieved. If a match occurs, TM1 executes the appropriate rules statement and returns the value.

Values for rules-calculated cells are then stored with the cube in memory. This prevents the same calculation from being repeatedly executed for a cell, and greatly increases performance. TM1 flags calculations as invalid when dependent values in the cube are modified. The next time a rules-derived value is requested, a fresh calculation is performed.

Monitoring rules statistics

You can monitor rules statistics that provide insight as to how frequently individual statements within a rule execute and how long it takes to run a rule statement.

About this task

Statistics about rule execution are stored in the }StatsByRule control cube.

Each time a rule is changed or compiled, the data for that rule is cleared and updated in the }StatsByRule control cube. This helps you to immediately see the impact of a rule change. The data in the }StatsByRule control cube does not persist between server sessions; it is cleared every time that you restart your TM1 server.

The }StatsByRule cube contains three dimension:

- }Cubes - Contains elements corresponding to each cube on your TM1 server.
- }LineNumber - Contains elements 1 through 10,000, corresponding to line numbers in a TM1 rules .rux file.

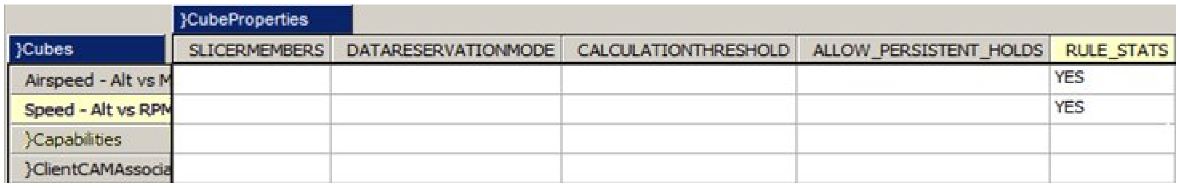
  Tip: The TM1 Rules Editor does not display line numbers. Open the .rux file in a text editor that supports line numbers to view the line numbers for a rule.
- }Rules Stats - Contains elements corresponding to the information and statistics that are collected for rules, including:
  - Rule Text - The first portion of a rule statement, provided to help you identify the statement.
  - Total Run Count - The total number of times the rule statement has run.
  - Min Time - The minimum amount of time taken for the rule statement to run, in milliseconds.
  - Max Time - The maximum amount of time taken for the rule statement to run, in milliseconds.
  - Total Time - The total amount of time taken for the rule statement to run, in milliseconds.
  - Last Run Time - The amount of time, in milliseconds, it took for the most recent execution of the rule statement.

Rule statistics collection is enabled on a per-cube basis by setting the RULE_STATS property to YES in the }CubeProperties control cube.

Note: The collection of rule statistics does incur a slight performance cost that increases with the frequency of rule execution. You should enable statistic collection only while debugging or tuning your rules. During normal operation you should disable statistic collection.

Procedure

1. Open the }CubeProperties control cube.
2. For each rule that you want to collect statistics, enter YES in the cell at the intersection of the cube name and the RULE_STATS property.



}CubeProperties

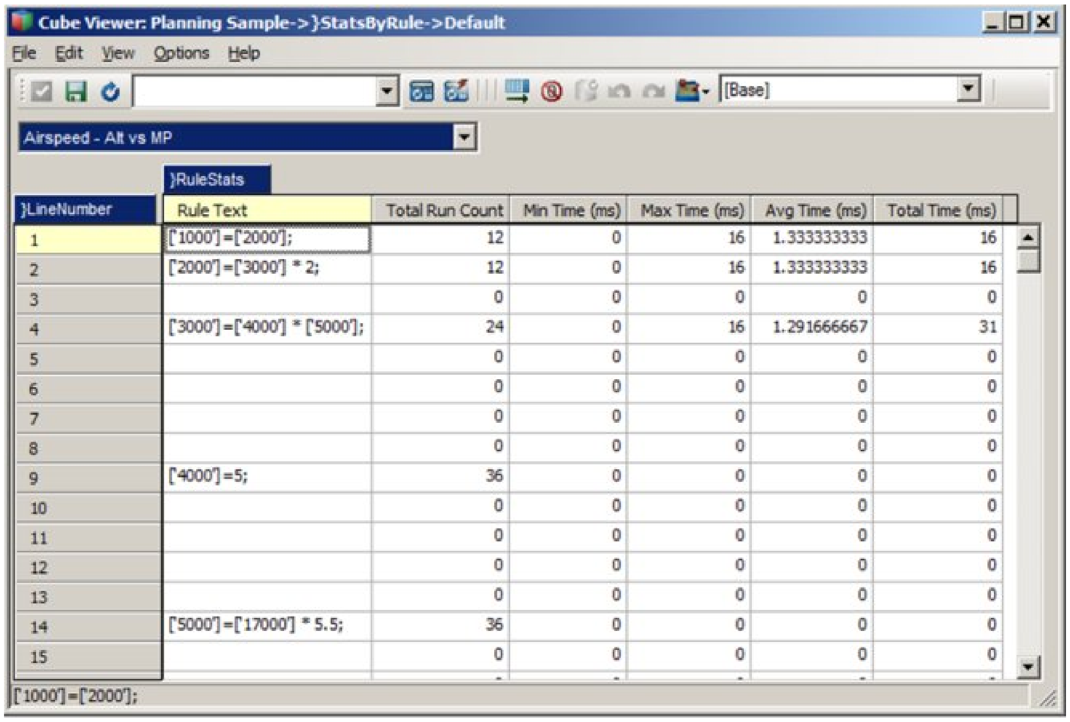
}Cubes	SLICERMEMBERS	DATARESERVATIONMODE	CALCULATIONTHRESHOLD	ALLOW_PERSISTENT_HOLDS	RULE_STATS
Airspeed - Alt vs M					YES
Speed - Alt vs RPM					YES
}Capabilities					
}ClientCAMAssocia					

   Note: RULE_STATS is a dynamic parameter. It does not require a server restart to take effect, but there may be a delay of up to 60 seconds before the property is applied.

The TM1 server is now collecting statistics for the rules where the RULE_STATS property is YES. Any execution of the rules from this point forward will result in data stored in the }StatsByRule control cube.

When you want to disable the collection of rule statistics, set the RULE_STATS property to NO.

3. Open the }StatsByRule control cube.
4. Review the statistics stored for each statement in your TM1 rule. These statistics can help you identify which statements might be running more frequently than intended or are taking a long time to run. You can use this information to modify your rules.


Cube Viewer: Planning Sample->}StatsByRule->Default
File Edit View Options Help
[Base]
Airspeed - Alt vs MP
}RuleStats

}LineNumber	Rule Text	Total Run Count	Min Time (ms)	Max Time (ms)	Avg Time (ms)	Total Time (ms)
1	['1000']=['2000'];	12	0	16	1.333333333	16
2	['2000']=['3000'] * 2;	12	0	16	1.333333333	16
3		0	0	0	0	0
4	['3000']=['4000'] * ['5000'];	24	0	16	1.291666667	31
5		0	0	0	0	0
6		0	0	0	0	0
7		0	0	0	0	0
8		0	0	0	0	0
9	['4000']=5;	36	0	0	0	0
10		0	0	0	0	0
11		0	0	0	0	0
12		0	0	0	0	0
13		0	0	0	0	0
14	['5000']=['17000'] * 5.5;	36	0	0	0	0
15		0	0	0	0	0

['1000']=['2000'];

The }LineNumber elements in this view correspond to the line numbers in the associated .rux file. If a rule statement occupies multiple lines in the .rux file, the line number shown in this view is the line on which the rule statement starts.

The times recorded for Min Time, Max Time, Avg Tme, Last Run time, and Total Time are in milliseconds (one one-thousandth of a second). Some rule statements execute faster than 1 millisecond, resulting in an entry of 0 for the time. It's possible for a simple rule statement to run multiple times, while the Total Time shows as 0.

5. To view statistics for a different rule, select the associated cube name from the }Cubes dimension at the top of the view.

Chapter 2. A Simple Rule

The remaining sections of this guide follow the transformation of data for a fictional company called Fishcakes International, which purchases different types of fish from markets around the world and uses that fish to make fishcakes.

Fishcakes International purchases fish in local currency. Costs in local currency must be calculated based on quantities purchased and the local currency cost per Kg. Local currency costs must then be transformed into US dollars using exchange rates. All these calculations are done using rules.

Fishcakes International also uses rules to calculate inventory levels and inventory depletion, as well as to allocate costs for individual fish types to individual fishcake types and to calculate final production costs.

In this section, you will begin the transformation of the Fishcakes International data by creating a rule that calculates purchase costs in local currency.

Setting Your Local Data Directory for Use with this Guide

Before you can proceed with developing the examples in this guide, you must first set your local data directory to use the appropriate sample data.

Follow the steps below to set your local IBM Cognos TM1 data directory:

Procedure

1. Select **File**, **Options** in the Server Explorer.

   The **TM1 Options** dialog box displays.
2. Click **Browse**.
3. Navigate to the Custom\TM1Data\Rules_Guide_Data directory.

   If you accepted the default installation directory when installing TM1 , the full path to this directory is

   ```
   C:\Program Files\Cognos\TM1\Custom\TM1Data\Rules_Guide_Data.
   ```
4. Click **Open**.
5. Click **OK** on the **TM1 Options** dialog box.

   When you change the local data directory, TM1 automatically restarts the local server with the new directory. Before restarting the local server, TM1 prompts you to save any current changes.
6. Click **OK**.

   TM1 restarts the local server using the Rules_Guide_Data directory, which contains the Fishcakes International data.

Purchase Cost Calculation

Fishcakes International buys fish in several markets around the world. The company tracks these purchases by fish type, by market, and by date.

For each purchase, the number of kilos and the price per kilo in local currency are input directly into a cube named Purchase.

The following table lists the four dimensions related to fish purchases in the Purchase cube with their names and descriptions.

Dimension Name	**Description**
Fishtype	Contains elements for each type of fish purchased. There are 19 leaf elements in the dimension, as well as a Total Fish Types consolidation.
Market	Contains elements for the markets in which fish are purchased. There are seven leaf elements in the dimension, as well as the Total Markets consolidation.
Date	Contains elements for the dates on which fish are purchased. The dimension contains 366 leaf elements, as well as multiple consolidations at varying levels.
PurMeas	Contains five leaf elements that measure fish purchases. Measure include Purchase Cost - LC, Quantity Purchased - Kgs, and Price/Kg - LC.

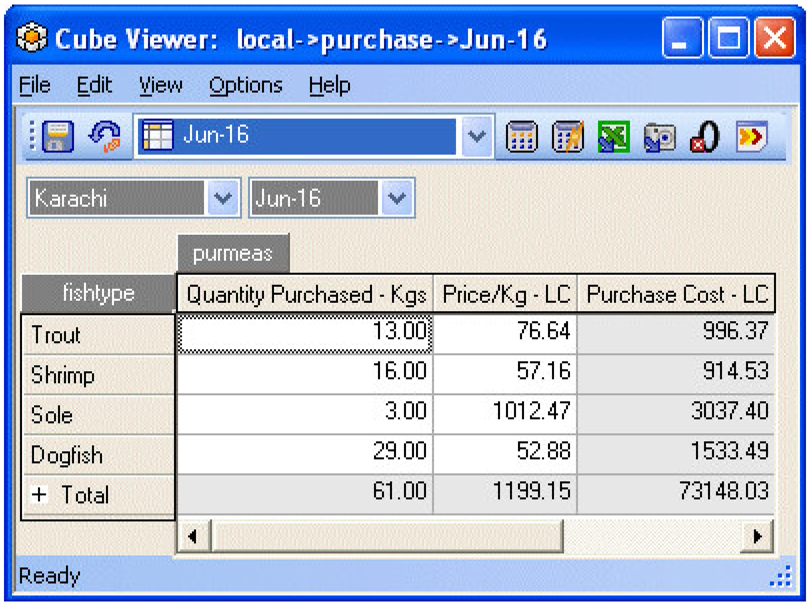
Using rules, you need to calculate the purchase cost in local currency by writing one rule statement that defines purchase cost as the product of price and amount purchased. This one statement will apply to all fish, for all markets and all dates in the Purchase cube.

Using the element names from the PurMeas dimension and applying the rules syntax presented in the first section of this book, you can create the following calculation statement:

```
['Purchase Cost - LC'] = ['Quantity Purchased - Kgs']*['Price/Kg
- LC'];
```

This statement says that if the TM1 server is asked for any value that matches the area Purchase Cost - LC, it will multiply the corresponding values for Quantity Purchased - Kgs and Price/Kg - LC and return the answer.

When this calculation statement is in place, you can see that Purchase Cost - LC values are calculated in the Purchase cube:


Cube Viewer: local->purchase->Jun-16
File Edit View Options Help
Jun-16
Karachi
Jun-16
purmeas
Ready

fishtype	Quantity Purchased - Kgs	Price/Kg - LC	Purchase Cost - LC
Trout	13.00	76.64	996.37
Shrimp	16.00	57.16	914.53
Sole	3.00	1012.47	3037.40
Dogfish	29.00	52.88	1533.49
+ Total	61.00	1199.15	73148.03

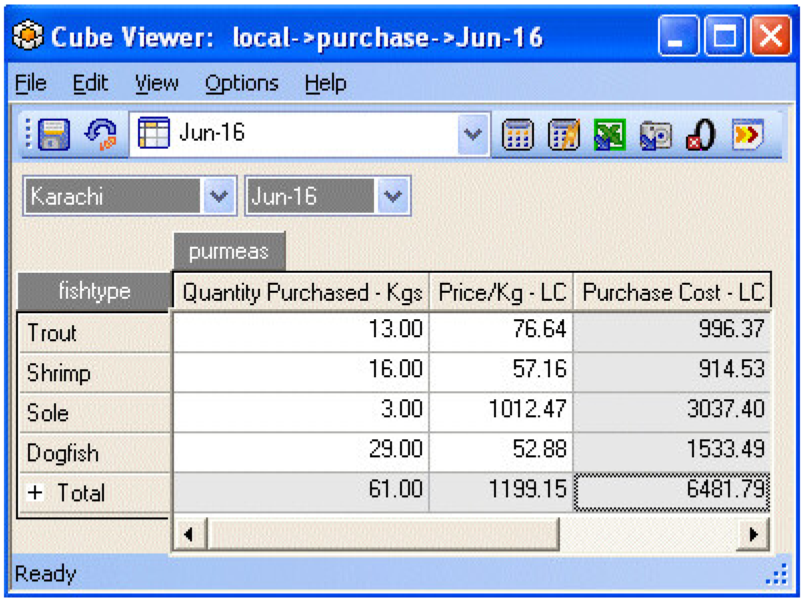
But what about the value in the lower right cell? The calculation statement is multiplying Total, Quantity Purchased - Kgs by the Total, Price/Kg - LC. Of course, prices per Kg do not sum, so the value for Total, Purchase Cost - LC is incorrect.

The best way to generate the total Purchase Cost - LC is to sum purchase cost values across fish, which TM1 can do automatically through its dimensional consolidations.

You can specify that a rule applies only to leaf cells and not to consolidations such as Total. This is done by inserting an N: in front of the formula portion of the calculation statement:

```
['Purchase Cost - LC'] = N: ['Quantity Purchased - Kgs']*
['Price/Kg
- LC'];
```

When the statement is corrected as above, consolidations are not calculated by the rule. Accordingly, the total Purchase Cost - LC consolidation is correctly calculated by summing the values of the members of the consolidation, as shown in the following figure.



fishtype	Quantity Purchased - Kgs	Price/Kg - LC	Purchase Cost - LC
Trout	13.00	76.64	996.37
Shrimp	16.00	57.16	914.53
Sole	3.00	1012.47	3037.40
Dogfish	29.00	52.88	1533.49
+ Total	61.00	1199.15	6481.79

Writing the Rule

The following example guides you through the creation of the rule described in the example, which calculates the value for Purchase Cost - LC.

Opening the Rules Editor

Follow the steps below to begin creating the rule.

Procedure

1. Open your Local server in the **Server Explorer**.

   Your Local server should be configured to use the Fishcakes International sample data. For more information, see "Setting Your Local Data Directory for Use with this Guide" on page 29.
2. Select the **Purchase** cube.

   In this example, **Purchase** is the cube for which you want to create a rule. Rules are always associated with a specific cube.
3. Click **Cube**, **Create Rule**.

   The **Rules Editor** displays.

   The **Rules Editor** is a text editor that helps you create accurate cube references, with toolbar buttons that let you insert commonly used rule syntax.

Creating the Rule

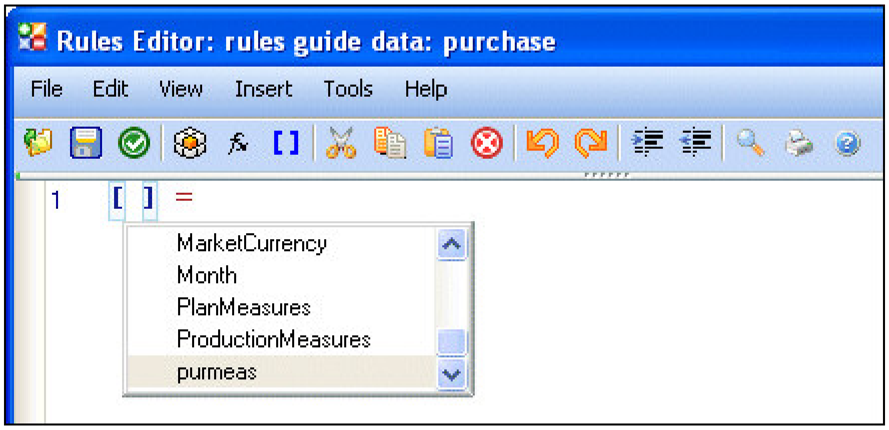
You can create a rule by typing the entire text of the rule directly in the **Rules Editor**. By using the menus and toolbar buttons in the **Rules Editor**, you can ensure that cube references are structured correctly.

Follow the steps below to create the example calculation statement.

Procedure

1. Click the **Brackets** button.

   The **Brackets** list displays in the **Edit** pane.

   This list lets you define a reference to the cube with which the rule is associated. You first select a dimension and then use the **Subset Editor** to select an element.

   For this example, you want to create a cube reference that defines the area of the rule as `['Purchase Cost - LC']`, an element of the PurMeas dimension.
2. Select the **PurMeas** dimension from the **Brackets** list.

   The **Subset Editor** displays.
3. Select **Purchase Cost - LC** in the list of elements.

   You could define an area with a selection of single elements from any combination of dimensions. For instance, you could let the statement apply only to one market and one fish type on a certain date.

   In this example, though, you want Purchase Cost - LC to be calculated the same way for all leaf cells, so the area definition is complete.
4. Click **OK**.

   The **Rules Editor** displays the area definition, which is formatted correctly with brackets and single quotes.

   The area definition is complete. Now you can write the remainder of the calculation statement.
5. Enter the following text after the area definition:

   ```
   N:
   ```

   The N: qualifier is necessary in the statement because you want the statement to apply only to leaf elements.

   Now you must create the formula that calculates values for the specified area.
6. Click **Brackets**.
7. Select the **PurMeas** dimension from the Brackets list.
8. In the **Subset Editor**, select **Quantity Purchased - Kgs** and click **OK.**

   The **Rules Editor** should now contain the following statement.

   ```
   ['Purchase Cost - LC']=
   N:['Quantity Purchased - Kgs']
   ```

9. Enter the multiplication symbol * at the end of the statement.
10.Click **Brackets**.
11.Select the **PurMeas** dimension from the **Brackets** list.
12.In the **Subset Editor**, select **Price/Kg - LC** and click **OK.**

The **Rules Editor** should now contain the area definition, leaf qualifier, and formula for the rule.

The calculation statement is nearly complete. All that's missing is the semicolon (;) indicating the end of the statement.

13.Enter a semicolon ; at the end of the statement.

Note: Always remember to insert the trailing semicolon! Forgetting the semicolon is the most common syntax error in rules, even for long-time users.

Compiling the Rule

Before the rule can calculate values, it must be compiled and saved.

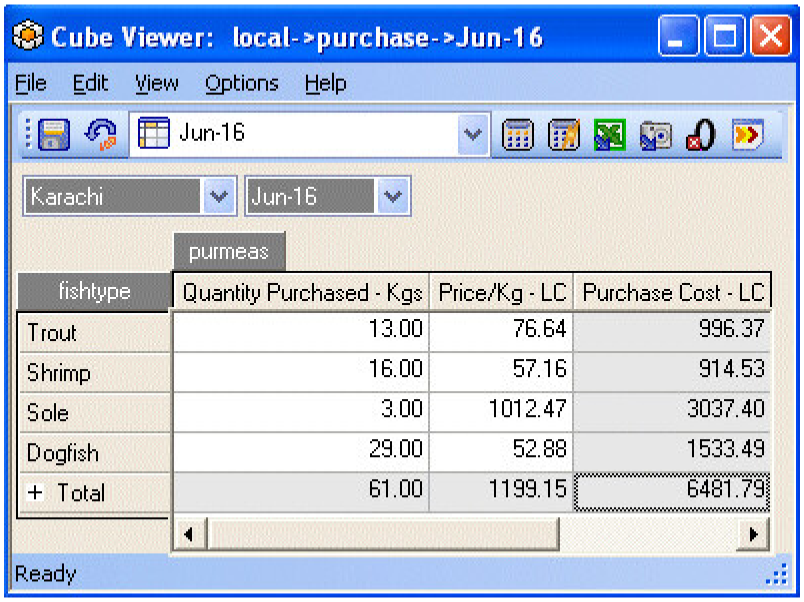
Do the following step to compile and save:

Procedure

Click **Save** to compile and save the rule.

Viewing the Results

To view the results of the new rule, open the Jun -16 view of the Purchase cube.


Cube Viewer: local->purchase->Jun-16
File Edit View Options Help
Jun-16
Karachi
Jun-16
purmeas
Ready

fishtype	Quantity Purchased - Kgs	Price/Kg - LC	Purchase Cost - LC
Trout	13.00	76.64	996.37
Shrimp	16.00	57.16	914.53
Sole	3.00	1012.47	3037.40
Dogfish	29.00	52.88	1533.49
+ Total	61.00	1199.15	6481.79

This view confirms that for individual fish types, values for Purchase Cost - LC are calculated by multiplying Quantity Purchased Kgs by Price/Kg - LC. The value of Total, Purchase Cost - LC is calculated by the consolidation of individual fish, as defined in the FishType dimension.

In the **Server Explorer**, a rule icon now displays beneath the Purchase cube.

You can double-click the rule icon to open a rule for editing in the **Rules Editor**.

Chapter 3. Exchange Rates

Fishcakes International purchases fish in multiple markets using different local currencies.

To keep accurate track of costs, the company needs to convert these purchases into a common currency. Fishcakes International translates all local currency purchases into US dollars, which it derives through exchange rates.

Accurately modeling currency conversions can often present problems because exchange rates can change very quickly, with a heavy impact on the bottom line. Many tools are negatively impacted by the burden of calculating daily exchange rates, or require that the rates be applied monthly. Calculation on demand, though, combined with the efficient indexing of TM1 , makes it easy to calculate detailed models with daily exchange rates.

Exchange Rate Calculation within a Cube

One way to address exchange rates would be to add an Exchange Rate element to the Purmeas dimension. Then, as analysts add information on sales, they would enter the exchange rate for the sale at the same time. You could then write a rule to calculate the price per Kg in US dollars.

One problem with this approach is that it requires a lot of excess data entry, with the exchange rate being duplicated for each fish type. The exchange rate does not actually vary by fish type, but rather by the day and the currency in use at a particular market. The duplicate data also invites errors. It is easy to make a mistake with one or another of the many identical entries.

Though storing exchange rates in the cube where you want to calculate the price in US dollars is a valid approach, it is not the most efficient solution.

Exchange Rate as an Attribute of the Market Dimension

You could also calculate the price per Kg in US dollars by creating element attributes of the Date dimension that track exchange rates for individual markets.

You could then create a rule that uses the ATTRN rules function to retrieve the necessary attribute values and calculate costs in US dollars. This is a perfectly valid approach, particularly if attribute values are regularly updated through a TurboIntegrator process.

Exchange Rate Tracked by Market

A third way to solve this problem is to use a two-dimensional exchange rate cube MarketExchange, comprised of the dimensions Market and Date, and store exchange rates in that cube.

You can link the MarketExchange cube with the Purchase cube to calculate price per Kg and total purchase price in US dollars. This is the solution you will develop, as it provides an excellent example of inter-cube calculations with the use of DB functions.

The DB Formula

The DB function allows you to write a rule that references values in any other cube on the same TM1 server .

TM1 supports multiple interrelated cubes of differing but overlapping dimensionality, so there are all kinds of applications that benefit from DB functions.

In the following examples, you will connect purchases in the Purchase cube (dimensioned by fish, market, and date) to exchange rates in the ExchangeRate cube (dimensioned only by market and date). In other applications, you might apply commission rates varying by employee to a sales database dimensioned by employee, product and time. Or you could use a cube containing percentages of active ingredients in pharmaceutical products to derive production statistics in another cube, which contains shipments by product.

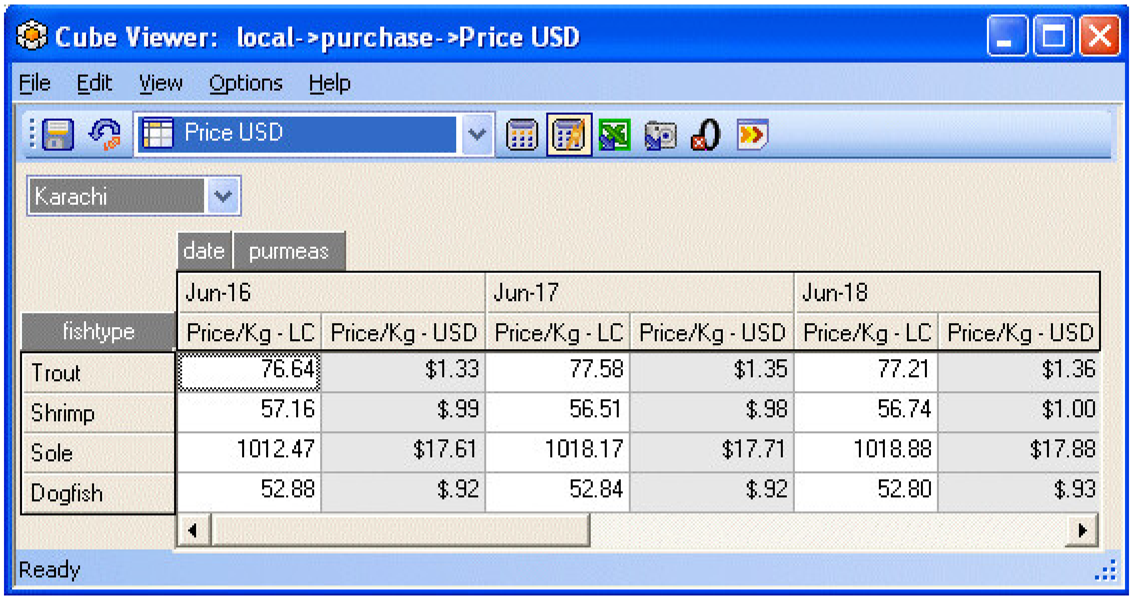
The Rule

To calculate price per Kg in US dollars, you create a rule statement in the rule for the Purchase cube that divides the price per Kg in local currency by the exchange rate for the current market and date.

The exchange rate, which is stored in the MarketExchange cube, is retrieved by a DB function.

```
['Price/Kg - USD']=['Price/Kg - LC']\DB('MarketExchange',!market,!date);
```

When you compile the rule and browse the cube you can see that the rule indeed returns price per Kg in USdollars (USD).

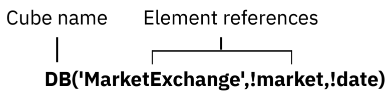

Cube Viewer: local->purchase->Price USD
File Edit View Options Help
Price USD
Karachi
date purmeas
fishtype
Ready

fishtype	Jun-16		Jun-17		Jun-18	
	Price/Kg - LC	Price/Kg - USD	Price/Kg - LC	Price/Kg - USD	Price/Kg - LC	Price/Kg - USD
Trout	76.64	$1.33	77.58	$1.35	77.21	$1.36
Shrimp	57.16	$.99	56.51	$.98	56.74	$1.00
Sole	1012.47	$17.61	1018.17	$17.71	1018.88	$17.88
Dogfish	52.88	$.92	52.84	$.92	52.80	$.93

How a DB Formula Works

Let's take a closer look at the DB function in the preceding rule statement.

The function has three arguments: the name of the cube from which values are retrieved and two element references. Each element reference is preceded with an exclamation point, often called a "bang."



To understand the "bang", keep in mind that the rule is executed only when a value is requested. The expression "`!market`" tells the rules interpreter to use the element of the Market dimension in the MarketExchange cube that matches the Market element for the requested value in the Purchase cube.

When using DB functions, note the distinction between the *source* and *target* cubes. The target cube is the cube whose rule contains the function, the one receiving the value specified by the rule. The source cube is the cube providing values, the one referenced by the DB function.

When writing DB functions always keep in mind the dimensional structure of the target cube for which the rule is being written as well as that of the source cube referenced in the DB functions. When TM1 evaluates a DB function it looks at the cube specified by the first argument to the function and expects to find an argument for each dimension of that cube, in the correct order. Each argument can be any TM1 expression, but it must evaluate to a valid element of the corresponding dimension.

To better understand DB functions and the use of the !*dimension* argument, consider what happens when the server receives a request for the Price/Kg - USD of Trout in Karachi on June 16th:

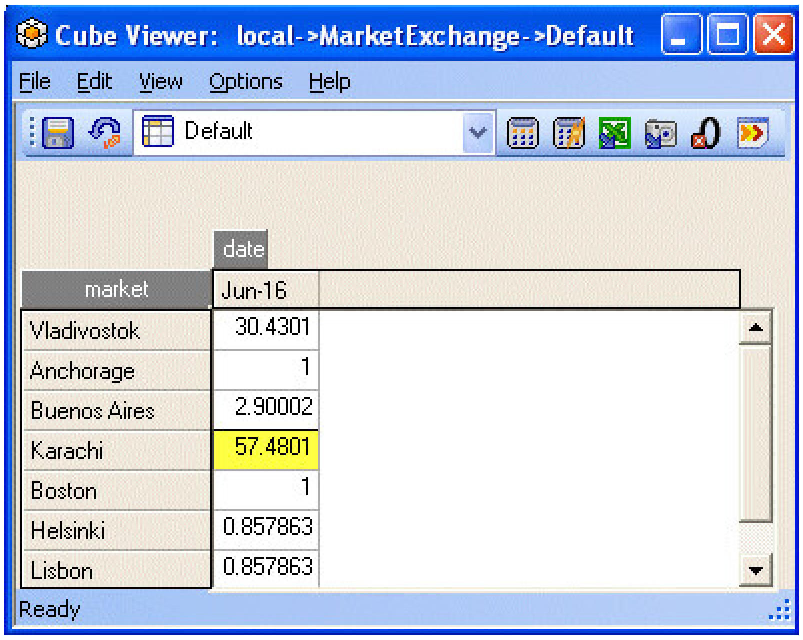
Procedure

1. The server determines if the value requested matches any rule area definitions.

```
['Price/Kg - USD']=['Price/Kg - LC']\DB('MarketExchange',!market,!date)
```

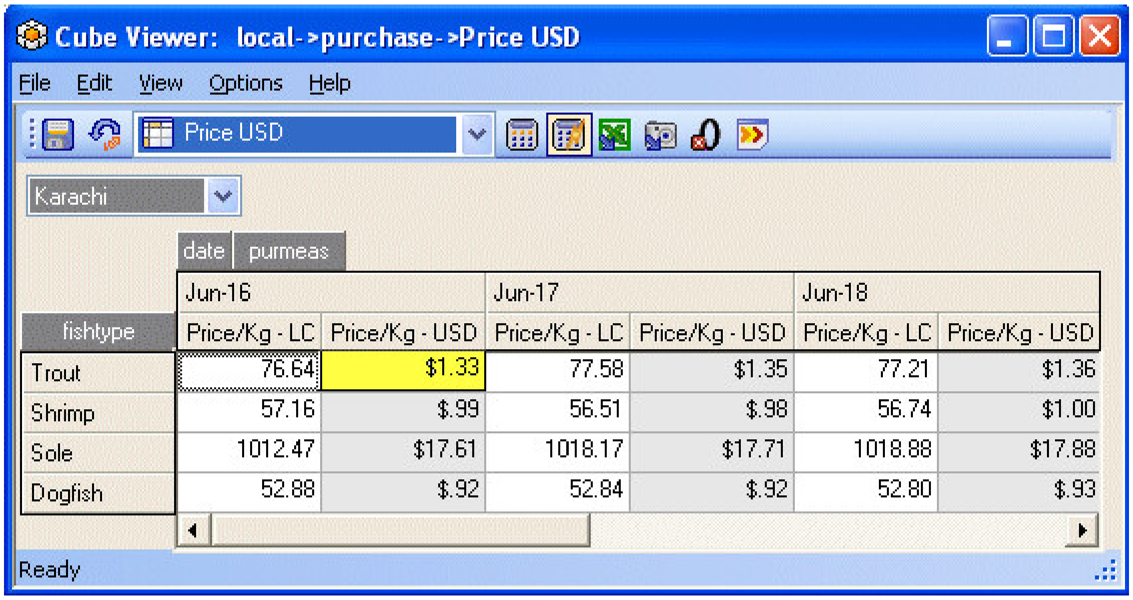
2. The request does match the area definition of the above rule statement, so TM1 begins to evaluate the formula portion of the statement.
3. TM1 retrieves the value of Price/Kg - LC, keeping all other dimension elements constant (Trout, Karachi, Jun -16). In the view above, this value is 76.64.
4. TM1 evaluates the function `DB('MarketExchange',!market,!date)`.

   - The first argument to the DB function tells TM1 to look in the MarketExchange cube.
   - The second argument (`!market`) tells TM1 to use the element of the Market dimension that corresponds to the value being requested. In this case, the element is Karachi.
   - The third argument (`!date`) tells TM1 to use the element of the Date dimension that corresponds to the value being requested. In this case, the element is Jun -16.
   - The DB function evaluates to `DB('MarketExchange','Karachi','Jun-16')`. TM1 retrieves the value for Karachi and Jun -16 from the MarketExchange cube.



market	Jun-16
Vladivostok	30.4301
Anchorage	1
Buenos Aires	2.90002
Karachi	57.4801
Boston	1
Helsinki	0.857863
Lisbon	0.857863

   The exchange rate for Karachi on Jun -16 is 57.4801.
5. The formula portion of the rule evaluates to 76.64\57.4801.
6. TM1 calculates the value of Price/Kg - USD for Trout in Karachi on June 16 to be $1.33.


Cube Viewer: local->purchase->Price USD
File Edit View Options Help
Price USD
Karachi
date
purmeas
Ready

fishtype	Jun-16		Jun-17		Jun-18	
	Price/Kg - LC	Price/Kg - USD	Price/Kg - LC	Price/Kg - USD	Price/Kg - LC	Price/Kg - USD
Trout	76.64	$1.33	77.58	$1.35	77.21	$1.36
Shrimp	57.16	$.99	56.51	$.98	56.74	$1.00
Sole	1012.47	$17.61	1018.17	$17.71	1018.88	$17.88
Dogfish	52.88	$.92	52.84	$.92	52.80	$.93

Writing the Exchange Rate Rule Statement

Follow these steps to create the rule statement that calculates values for Price/Kg - USD.

Procedure

1. In the **Server Explorer**, right-click the Purchase cube, and click **Edit Rule**.
2. Move the cursor to the beginning of a new line beneath the existing statement in the **Rules Editor**.
3. Write the area definition and the first part of the formula. When you are done, the partial statement should appear as follows:

   ```
   ['Price/Kg - USD']=['Price/Kg - LC']
   ```

   For more information about the toolbar buttons in the **Rules Editor** and techniques, see Chapter 2, "A Simple Rule," on page 29.
4. Enter the backslash character \ to represent a special type of division operator.

   This character is a division operator, but it is different from the normal division operator, the forward slash (/). When you use the backslash operator, any rule that results in division by zero returns the value 0 rather than the N/A error.
5. Click the **Insert Cube Reference** button.

   The **Insert Cube Reference** dialog box displays. This dialog box lets you select the cube from which the DB formula retrieves values.
6. In the **Select Cube** list, select MarketExchange.

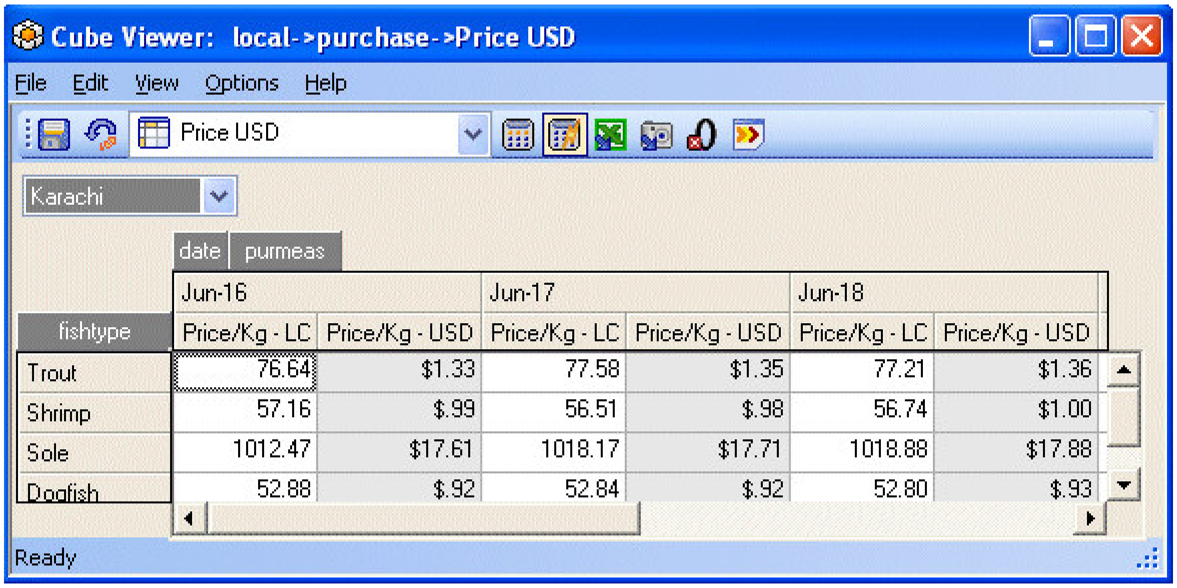
   The dimensions for the MarketExchange cube appear in the **Dimension** list.

   You want this reference to apply to all markets and all dates, so there is no need to narrow the reference definition.
7. Click **OK** to return to the **Rules Editor**.
8. Enter a semicolon (**;**) at the end of the rule statement.

   The completed rule statement should appear as follows:

   ```
   ['Price/Kg - USD']=['Price/Kg - LC']\DB('MarketExchange',!market,!date);
   ```
9. Click **Save**.
10. Open the Price USD view of the Purchase cube to see the result of the new rule statement.

    Note: All values for Price/Kg - USD are now calculated by the rule.


Cube Viewer: local->purchase->Price USD
File Edit View Options Help
Price USD
Karachi
date
purmeas
Ready

fishtype	Jun-16		Jun-17		Jun-18	
	Price/Kg - LC	Price/Kg - USD	Price/Kg - LC	Price/Kg - USD	Price/Kg - LC	Price/Kg - USD
Trout	76.64	$1.33	77.58	$1.35	77.21	$1.36
Shrimp	57.16	$.99	56.51	$.98	56.74	$1.00
Sole	1012.47	$17.61	1018.17	$17.71	1018.88	$17.88
Dogfish	52.88	$.92	52.84	$.92	52.80	$.93

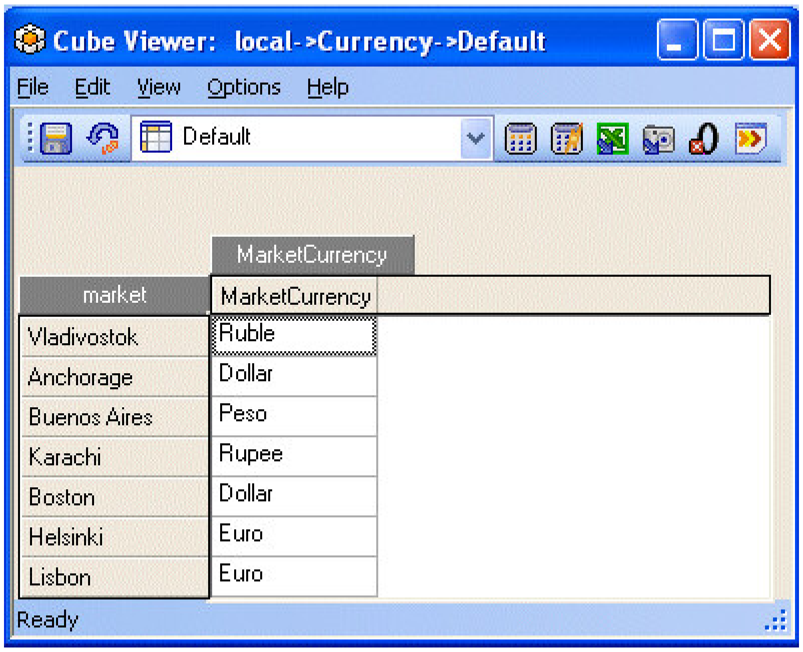
Exchange Rate Calculation Using a Lookup Cube

The preceding example is a good first step towards developing a rules-based solution to currency conversion, but the DB function for currency conversion does not reflect the fact that exchange rates vary by currency, not by market.

For example, both the Helsinki and Lisbon markets use the Euro currency, and the exchange rate for the Euro fluctuates independent of the market in which it is used.

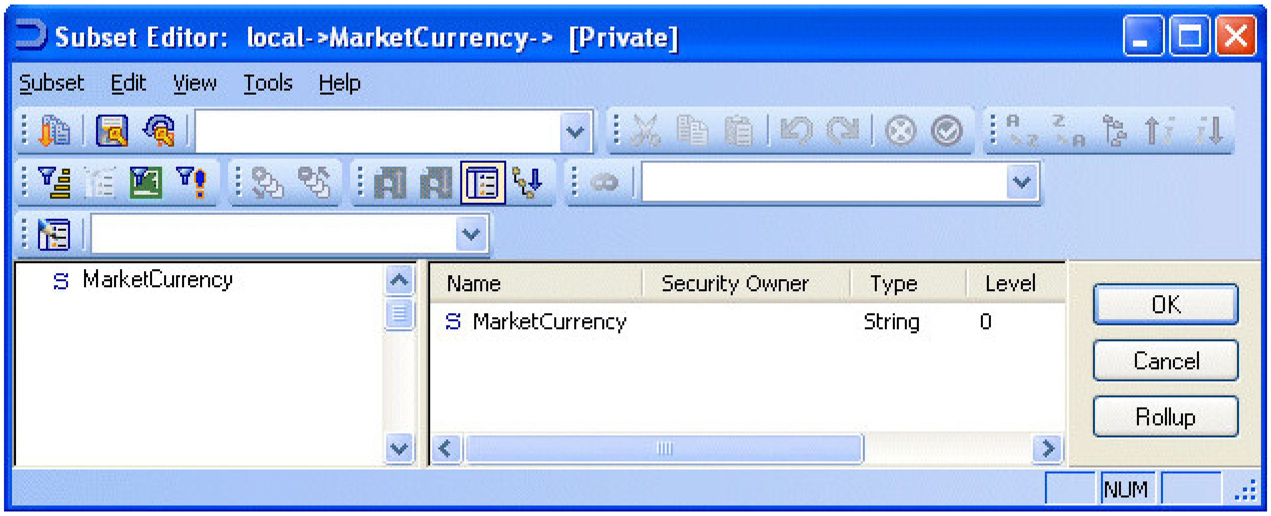
The preceding solution requires the identical exchange rate for the Euro to be stored twice; once for Helsinki and once for Lisbon. Ideally, a currency conversion solution stores the exchange rate for a currency in a single location.

You can solve this problem of redundant data by using a cube that stores the relationship of market to currency. The sample data contains just such a cube, named Currency.


Cube Viewer: local->Currency->Default
File Edit View Options Help
Default
MarketCurrency
Ready

market	MarketCurrency
Vladivostok	Ruble
Anchorage	Dollar
Buenos Aires	Peso
Karachi	Rupee
Boston	Dollar
Helsinki	Euro
Lisbon	Euro

This cube is comprised of two dimensions: the Market dimension, and a dimension named MarketCurrency, which contains a single string element (also named MarketCurrency).



Name	Security Owner	Type	Level
MarketCurrency		String	0

When used in conjunction with a lookup cube that stores exchanges rates by currency (CurrencyExchangeRate), you can develop a rule statement that calculates Price/Kg -USD for each market based on the daily exchange rate for a given currency. The next section illustrates the creation of such a statement.

Using a Nested DB Function

The rule statement in the rule for the Purchase cube uses nested DB functions to calculate the value of Price/Kg - USD based on the current exchange rate in a given market.

```
['Price/Kg - USD']=['Price/Kg - LC']\DB('CurrencyExchangeRate',DB('Currency',
```

```
!market,'MarketCurrency'),!date);
```

How It Works

The trick is that even though the source cube (Production) does not have a Month dimension, the DB function scans the referenced cube for dimension elements whose names match those of elements in the equivalently ordered dimension in the target cube.

In this case, the DB function looks for elements in the Month dimension that match element names in the Date dimension.

Procedure

1. As always, the TM1 server determines if the value requested matches any rule statement area definitions.

   ```
   ['Price/Kg - USD']=['Price/Kg - LC']\DB('CurrencyExchangeRate',
   ```

   ```
   DB('Currency',!market,'MarketCurrency'),!date);
   ```

2. The request does match the area definition of the above rule statement, so TM1 begins to evaluate the formula portion of the statement.
3. TM1 retrieves the value of Price/Kg - LC, keeping all other dimension elements constant (Trout, Karachi, Jun -16.) In the Purchase cube, this value is 76.64.
4. TM1 evaluates the nested DB functions

   ```
   DB('CurrencyExchangeRate',DB('Currency',!market,'MarketCurrency'),!date).
   ```

   - The first argument to the outer DB function tells TM1 to look in the CurrencyExchangeRate cube.

- The second argument to the outer function, which must resolve to a valid element of the Currency dimension, is another DB function:

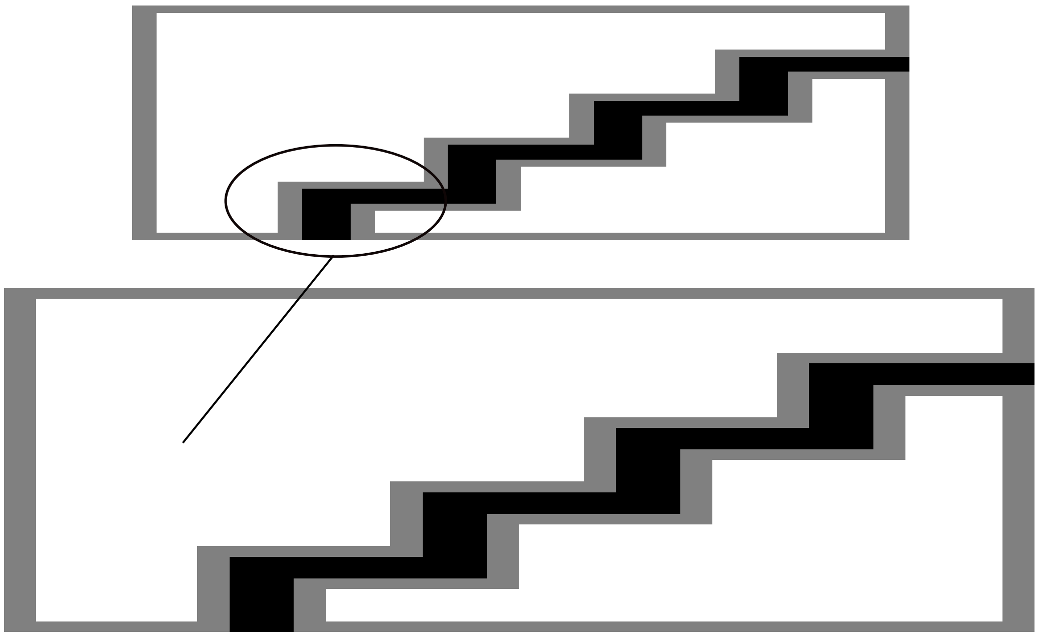
  ```
  DB('Currency',!market,'MarketCurrency')
  ```

  This nested DB function returns the value from the Currency cube identified by the market for the value being requested (!market) and the element MarketCurrency. In this case, the TM1 server is processing a request for a value in the Karachi market, so the function returns the string 'Rupee'.

  Rupee is a valid element of the Currency dimension.

  Rupee is the second argument to the outer DB function.
- The third argument to the outer DB function is `!date`, which evaluates to Jun -16, the date of the requested value.

5. The outer DB function evaluates to DB('CurrencyExchangeRate','Karachi','Jun-16'). TM1 retrieves the value for Karachi and Jun -16 from the MarketExchange cube.
6. The formula portion of the rule statement evaluates to 76.64\57.4801.
7. TM1 calculates the value of Price/Kg - USD for Trout in Karachi on June 16 to be $1.33, as shown in the following figure.



Writing Nested DBs

With nested DB functions, you must ensure that the inner DB returns a valid argument to the outer DB.

Before you begin

The following procedure, which steps you through the creation of the preceding rules statement example, illustrates one good approach to creating nested DB functions.

Procedure

1. In the **Server Explorer**, right-click the Purchase cube, and click **Edit Rule**.

   The **Rules Editor** should already contain a statement to calculate Price/Kg - USD, which you created earlier in this section.
2. Insert a pound symbol **#** at the beginning of the existing statement.

   The pound symbol (#) is the comment character for rules.

3. Create the following text in the **Rules Editor** on a new line beneath the existing statement. You can enter the text manually or use the toolbar buttons.

```
['Purchase Cost - LC']=N:['Quantity Purchased - Kgs']*['Price/Kg
-LC']
```

```

#['Price/Kg - USD']=['Price/Kg - LC']\DB('MarketExchange',!market,!date);
```

```

['Price/Kg - USD']=['Price/Kg - LC']\
```

4. Click the **Insert Cube Reference** button.

   The **Insert Cube Reference** dialog box displays.
5. In the **Select Cube** list, select the **CurrencyExchangeRate** cube.

   The dimensions for the CurrencyExchangeRate cube appear in the **Dimension** list.
6. Click **OK**.
7. Insert a semi-colon (**;**) at the end of the statement.

   The rule statement should now appear as follows.

```
['Purchase Cost - LC']=N:['Quantity Purchased - Kgs']*['Price/Kg
-LC']
```

```

#['Price/Kg - USD']=['Price/Kg - LC']\DB('MarketExchange',!market,!date);
```

```
['Price/Kg - USD']=['Price/Kg - LC']\DB('CurrencyExchangeRate',!Currency,!date);
```

   Note that the DB function includes the argument !Currency, a reference to the current element of the Currency dimension. However, the rule statement is being written for the Purchase cube, which is composed of the dimensions FishType, Market, Date, and PurMeas. The Purchase cube does not include the Currency dimension; the DB formula as it now exists cannot resolve all arguments and displays an error message if you attempt to save the rule.

   Line 3: Syntax error on or before: !Currency,!date); invalid string expression Rule could not be attached to the cube, but changes were saved.

   You must replace `!Currency` with an argument that resolves to a dimension element of the Purchase cube.
8. Select the text `!Currency` in the **Rules Editor**.
9. Click the **Insert Cube Reference** button.

   The **Insert Cube Reference** dialog box displays.
10. In the **Select Cube** list, select the Currency cube.

    The dimensions for the Currency cube appear in the dialog.
11. Click the **Subset Editor** button for the **Market Currency** dimension.

    The **Subset Editor** opens with the sole element of the MarketCurrency dimension.
12. Select the MarketCurrency element and click **OK**.
13. Click **OK** in the **Insert Cube Reference** dialog box.

The complete rule statement should appear as follows.

```
['Purchase Cost - LC']=N:['Quantity Purchased - Kgs']*['Price/Kg
-LC']
```

```
#['Price/Kg - USD']=['Price/Kg - LC']\DB('MarketExchange',!market,!date);
```

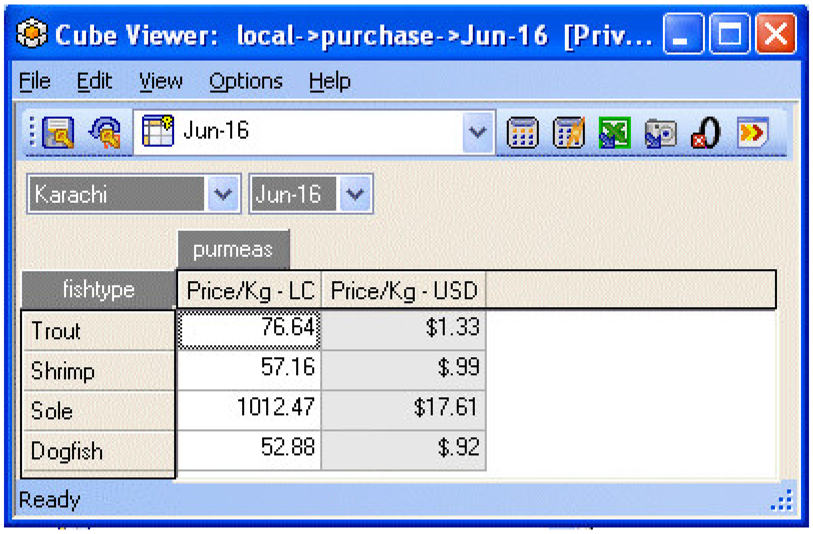
```
!market,!date);
```

```
['Price/Kg - USD']=['Price/Kg - LC']\DB('CurrencyExchangeRate',DB('Currency',
```

```
!market,'MarketCurrency'),!date);
```

14.Click **Save**.

15.Open the Jun -16 view of the Purchase cube to confirm that the new rule statement calculates values for Price/Kg - USD.


Cube Viewer: local->purchase->Jun-16 [Priv...
File Edit View Options Help
Jun-16
Karachi
Jun-16
purmeas
fishtype
Ready

fishtype	Price/Kg - LC	Price/Kg - USD
Trout	76.64	$1.33
Shrimp	57.16	$.99
Sole	1012.47	$17.61
Dogfish	52.88	$.92

This model using nested DB functions is more complicated than the first model used to calculate Price/Kg - USD, but it has the significant advantage of storing exchange rates for each currency in just one place and using a lookup cube to determine the correct currency for each market.

Other Lookup Cube Examples

Exchange rates are only one example of the many uses of lookup cubes in TM1.

All of these depend on the DB formula. Here are a few other lookup applications that have been developed in TM1:

- Products are sold in packages of different sizes. A Sales cube holds information on sales by product package; a lookup cube holds the amount of product held in each package. A rule yields the total amount of product sold.

  The rule in the Sales cube:

  ```
  ['Products sold'] = ['Packages sold']*
  DB('PkgComp',!Product,'NoProd');
  ```

- Labor costs depend on staffing and on salary, which varies by job grade. An Expenses cube holds information on headcount by job grade, broken out by month, division and location. A second lookup cube holds information on average salary for each job grade. A rule yields salary expense by month, division and location.

The rule in the Expenses cube:

```
['Labor costs'] = ['headcount']*
DB('AvgSal',!JobGrade,'Salary');
```

Calculating Purchase Cost - USD

The statement to calculate purchase cost in US dollars is similar to the statement that calculates price per Kg in U.S. dollars.

The statement divides purchase cost in local currency by the exchange rate for the local currency:

```
['Purchase Cost - USD']=N:['Purchase Cost - LC']\ DB('CurrencyExchangeRate',
DB('Currency',!market,'MarketCurrency'),!date);
```

Follow the steps below to add this calculation statement to the Purchase rule.

Procedure

1. Double-click the **Purchase** rule in the **Server Explorer**.

   The **Rules Editor** displays.
2. Add the above statement immediately following the last calculation statement in the rule. The complete Purchase rule should now appear as follows:

   ```
   ['Purchase Cost - LC']=N:['Quantity Purchased - Kgs']*['Price/Kg
   -LC'];
   ```

   ```
   #['Price/Kg - USD']=['Price/Kg - LC']\DB('MarketExchange',!market,!date);
   ```

   ```
   ['Price/Kg - USD']=['Price/Kg - LC']\DB('CurrencyExchangeRate',
   DB('Currency',!market,'MarketCurrency'),!date);
   ```

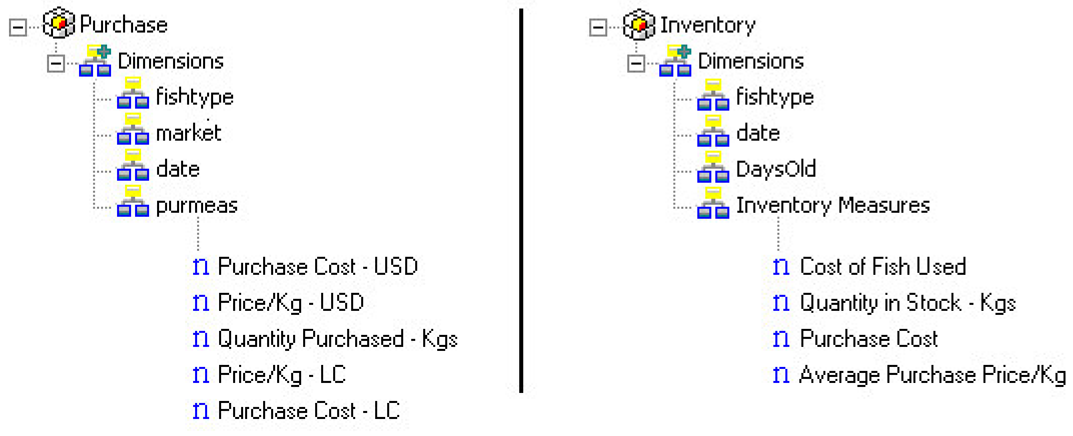
   ```
   ['Purchase Cost - USD']=N:['Purchase Cost - LC']\DB('CurrencyExchangeRate',
   DB('Currency',!market,'MarketCurrency'),!date);
   ```
3. Click **Save** to save the rule.

Chapter 4. Using DB Functions to Move Data Between Cubes

As you have seen, Fishcakes International tracks fish purchases by market. However, as fish moves into inventory, you no longer need to track data by market. Instead you need to transform purchase data into inventory data, and then track inventory levels, inventory age, and the total daily cost of each type of fish in inventory.

Data on inventory, as tracked in the Inventory cube, is dimensioned differently from that of purchases. Purchases, as tracked in the Purchase cube, are broken out by market, and each day's purchases are new events. Inventory doesn't track the market in which fish was purchased, but since fish spoils quickly, you need to know how old each batch of fish is. Local currency is important for purchases, but inventory is only concerned with the US dollar price.

To better understand the differences between the two cubes, consider the dimensions and measures used by each cube, as shown in the following figure. The cubes share only the FishType and Date dimensions and the cubes track completely different measures. Yet despite these differences, you can use DB functions to easily move data from the Purchase cube to the Inventory cube.



Creating Rules Statements for the Inventory Cube

The Fishcakes International model fills inventory from purchases, and depletes inventory as production proceeds.

To fill inventory, you need to create several rules statements for the Inventory cube that accurately define how to transform purchase data. The statements must accomplish the following:

- Only values for Total Markets in the Purchase cube are moved to the Inventory cube. This is because the Inventory cube does not track values by individual market and does not include the Market dimension.
- Values for each fish type in the Purchase cube must be moved to the corresponding fish type in the Inventory cube. The two cubes share the FishType dimension.
- Values for each date in the Purchase cube goes to the same date in the Inventory cube. The two cubes share the Date dimension.
- Values for Quantity Purchased - Kg in the Purchase cube are moved to Quantity in Stock - Kg in the Inventory cube.
- For any given date, values for newly purchased fish in the Purchase cube are moved to element 1 of the DaysOld dimension in the Inventory cube.

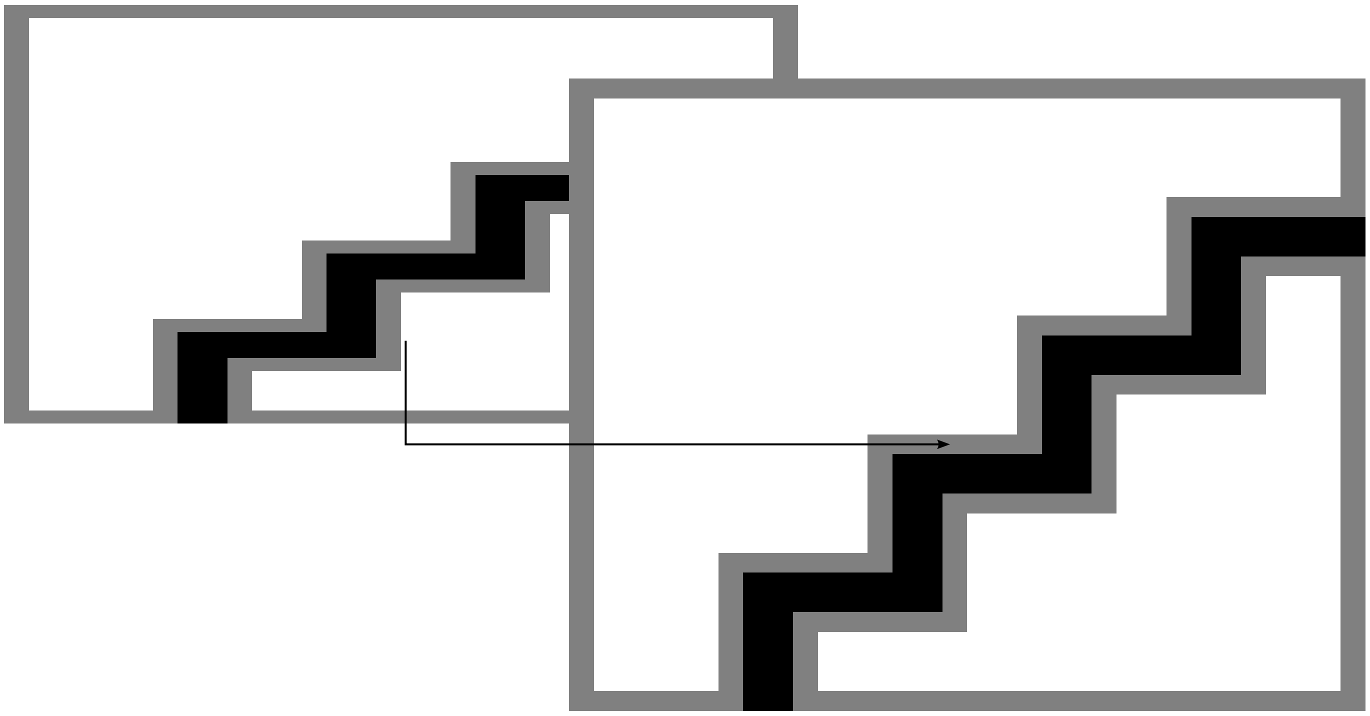
The DaysOld dimension contains elements that reflect the age of fish in inventory. Newly purchased fish is considered to be one day old. In a later section, you develop rules to cycle inventory through the DaysOld dimension on a daily basis as the inventory ages.

Calculating Quantity in Stock - Kgs

In the following statement, the value of Quantity in Stock - Kgs for DaysOld element 1 is calculated.

```
['1','Quantity in Stock - Kgs']= N:DB('Purchase',!FishType,'Total Markets',!
Date,'Quantity
Purchased - Kgs');
```

This statement says that for any given fish type on any given date, the value of Quantity in Stock - Kgs for DaysOld batch 1 is calculated by retrieving the value of Quantity Purchased - Kgs for Total Markets from the Purchase cube.

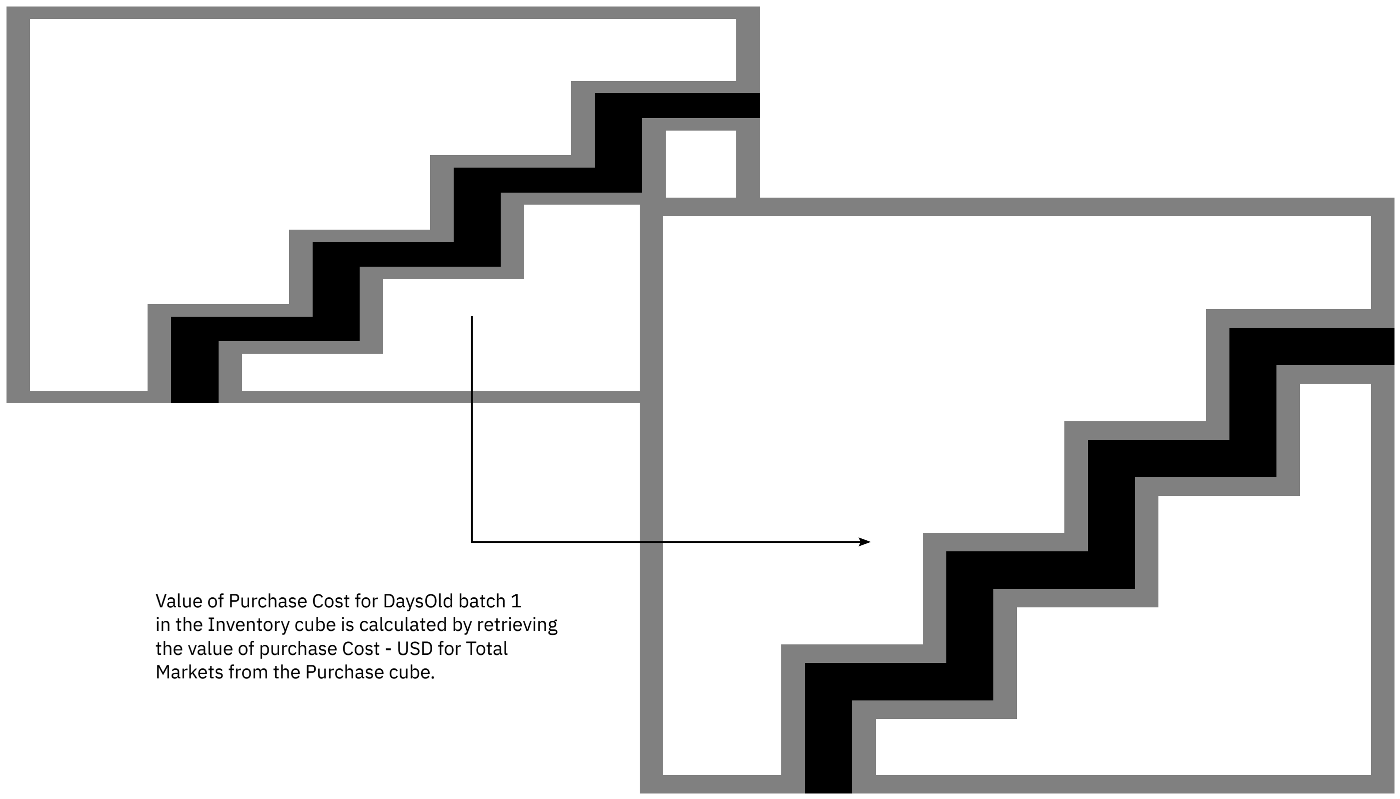


Calculating Purchase Cost

The following statement calculates the value of Purchase Cost for DaysOld element 1.

```
['1','Purchase Cost'] = N:DB('Purchase',!FishType,'Total Markets',!Date,'Purchase
Cost - USD');
```

This statement says that for any given fish type on any given date, the value of Purchase Cost for DaysOld batch 1 is calculated by retrieving the value of Purchase Cost - USD for Total Markets from the Purchase cube.


Value of Purchase Cost for DaysOld batch 1
in the Inventory cube is calculated by retrieving
the value of purchase Cost - USD for Total
Markets from the Purchase cube.

Calculating Average Purchase Price/Kgs

The following statement calculates the value of Average Purchase Price/Kg for DaysOld element 1.

```
['1','Average Purchase Price/Kg'] = N:['1','Purchase
Cost'] \ ['1','Quantity in Stock - Kgs'] ; C:0 ;
```

This statement says that for any given fish type on any given date, the value of Average Purchase Price/Kg for DaysOld batch 1 is calculated by dividing the Purchase Cost of the batch by the Quantity in Stock - Kgs for the batch.

Adding the Statements to the Rule for the Inventory Cube

You already have all the knowledge required to construct the rules statements for the Inventory rule.

Before you begin

For more information about how to construct DB functions using the features of the **Rules Editor**, see Chapter 3, "Exchange Rates," on page 35.

Procedure

1. Right-click the **Inventory** cube in the **Server Explorer,** and click **Create Rule**.

   The **Rules Editor** displays. You have not yet written any rules statements for the Inventory cube, so the editor is empty.

2. Create the following three statements in the **Rules Editor**.

   ```
   ['1','Quantity in Stock - Kgs']= N:DB('Purchase',!FishType,'Total
   Markets',!Date,'Quantity Purchased - Kgs');
   ```

   ```
   ['1','Purchase Cost'] = N:DB('Purchase',!FishType,'Total Markets',!Date,'Purchase
   Cost - USD');
   ```

   ```
   ['1','Average Purchase Price/Kg'] = N:['1','Purchase
   Cost'] \ ['1','Quantity in Stock - Kgs'] ; C:0 ;
   ```
3. Click **Save**.
4. Open the Jun - 16 inventory view of the Inventory cube to confirm that these statements calculate values as expected.

   Note: The rules statements calculate values only for DaysOld batch 1. In later sections, you develop additional statements for the Inventory rule that calculate values for all other DaysOld batches.

Chapter 5. Improving Performance with Feeders

This section examines feeders, a feature of IBM Cognos TM1 rules that can greatly enhance performance.

Rules are a flexible and powerful tool for working with multiple cubes of different dimensionality. However, the flexibility and power come with a cost -- decreased consolidation speed.

Feeders allow you to restore fast consolidation when using rules, but require a bit of specialized knowledge and careful implementation. This section provides you with the knowledge you need to improve the performance of your TM1 applications that use rules.

See also "Persistent Feeders" in *TM1 Operations* and *TM1 for Developers* "Advanced Calculations" topics for more information on feeders.

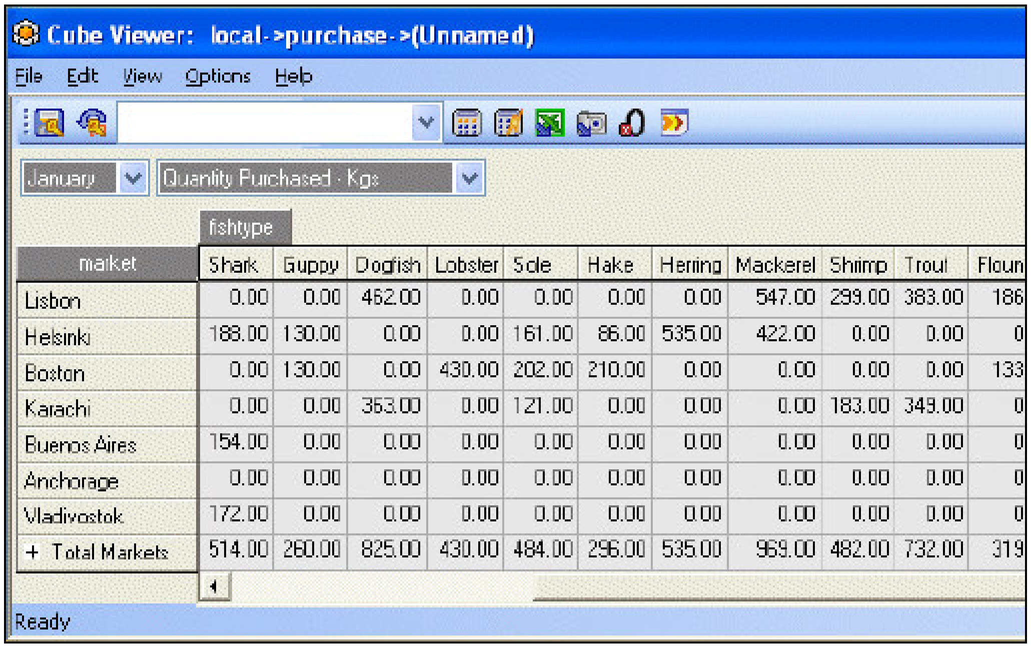
Sparsity

Before using feeders, you must understand the typical sparse distribution of multidimensional data and TM1 consolidation logic.

During consolidations, TM1 uses a sparse consolidation algorithm to skip over cells that contain zero or are empty. This algorithm speeds up consolidation calculations in cubes that are highly sparse. A sparse cube is a cube in which the number of populated cells as a percentage of total cells is low.

When consolidating data in cubes that have rules defined, TM1 turns off this sparse consolidation algorithm because one or more empty cells may be calculated by a rule. Skipping rules-calculated cells will cause consolidated totals to be incorrect. When the sparse consolidation algorithm is turned off, every cell is checked for a value during consolidation. This can slow down calculations in cubes that are very large and sparse.

Typically, multidimensional cubes contain many more cells with zeros than with values. For example, consider the Purchase cube, where each type of fish is purchased in just a few markets. A typical view of the Purchase cube looks like this:



market	Shark	Guppy	Dogfish	Lobster	Sole	Hake	Herring	Mackerel	Shrimp	Trout	Floun
Lisbon	0.00	0.00	462.00	0.00	0.00	0.00	0.00	547.00	299.00	383.00	186
Helsinki	188.00	130.00	0.00	0.00	161.00	86.00	535.00	422.00	0.00	0.00	0
Boston	0.00	130.00	0.00	430.00	202.00	210.00	0.00	0.00	0.00	0.00	133
Karachi	0.00	0.00	363.00	0.00	121.00	0.00	0.00	0.00	183.00	349.00	0
Buenos Aires	154.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
Anchorage	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
Vladivostok	172.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
+ Total Markets	514.00	260.00	825.00	430.00	484.00	296.00	535.00	969.00	482.00	732.00	319

Most values here are zeros, an indication that this cube is relatively sparse. Multidimensional cubes are almost always sparse.

Note: There is a semantic distinction between a zero and a value that is missing or non-applicable. Sparsity may apply to any of these possible states, with possibly different results. In TM1 though, values can only be real numbers, and the value zero is used to represent all of these states.

The impact of sparsity on calculations can be tremendous. Consider the consolidated value (8433) at the intersection of Total Markets and Total Fish Types, located in the lower right corner of the view above. If you add up every possible cell required to calculate this value, you must add values from 119 different cells. However, if you add only the cells with non-zero values, the number of components of the calculation drops to 46.

If you change the element of the Date dimension, you see even more sparsity, since not all markets are open on every date, and not all fish are bought on every date. For example, if you change the view and drill down to the single date Jun -16, you see that the number of components of the Total Markets/Total Fish Types consolidation drops to just 20!

On average, the more dimensions a cube has, the greater the degree of sparsity.

Sparsity and Rules Calculation

When you attach a rule to a cube, it becomes very difficult to use sparse consolidation.

This is because the TM1 consolidation engine does not know how to locate rules-derived values. Its only recourse is to scan every member cell of a consolidation to see if it contains a number. And as you just saw, this can be extremely inefficient.

Because consolidations will be incorrect if they skip values derived from rules, the default behavior is to turn off sparse consolidation when rules are defined for a cube. This ensures that all values are correct, but speed can plummet by several orders of magnitude (roughly proportional to the degree of sparsity of the cube data). If you build applications of even modest size that work with sparse data, you will have to write feeders. The process of creating feeders is described in detail in this section.

Each subsequent section in this guide contains instructions for creating feeders to accompany calculation statements.

Note: Rule-generated values are not displayed when zero suppression is applied to the TM1 view unless the value resides in a cell that is fed.

Skipcheck and Feeders

Some small, dense rules applications can work in TM1 default mode (without sparse consolidations), but applications of any size begin to run very slowly.

To solve this problem, TM1 lets you restore sparse consolidation and instruct the consolidation engine where to look for rules-derived values. To do this, you use a set of instructions in the rule for the cube:

After inserting a SKIPCHECK; statement in a rule, you can no longer take for granted that calculations are accurate. Even if rules are correctly defined, consolidation results will be wrong if feeders do not accurately describe where rules-derived values are located.

Your goal as a rules writer is to exactly specify all cells that contain rules-derived values. When this is not practical, you can "overfeed," or feed a somewhat larger set of cells than is strictly required. Overfeeding does not produce wrong values, but it can slow down the system, especially when defining feeders for consolidated cells (feeding a consolidated cell automatically feeds all children of the consolidation). However, "underfeeding" (failing to feed cells that contain rules-derived values) will result in incorrect values and must be avoided at all costs.

Procedure

1. In the **Rules Editor**, insert a SKIPCHECK declaration, which forces TM1 to use the sparse consolidation algorithm in all cases.

   ```
   SKIPCHECK;
   ```

2. Create feeder statements, which cause placeholder values to be stored in rules-calculated cells, so that the cells are not skipped during consolidations.

   ```
   Feeder area => reference to rules-calculated
   value;
   ```

The feeder area identifies a component of a rules-derived value. When this component contains a non-zero value, a placeholder is fed to cells containing the rules-calculated value identified on the right-hand side of the statement. This placeholder alerts the TM1 consolidation engine to include the rules-derived value when performing consolidations.

3. Precede the feeder statements with the FEEDERS declaration:

   ```
   FEEDERS;
   ```

4. If your rules define string values for any cells, you must also insert a FEEDSTRINGS declaration as the first line in your rule:

   ```
   FEEDSTRINGS;
   ```

   The FEEDSTRINGS declaration ensures that cells containing rules-derived strings are fed. If these cells are not fed, you cannot view rules-derived string values when zero-suppression is applied to a view, nor can you reliably reference the cells in other rules. Feeders for string cells should be inserted following the FEEDERS declaration in your rule.

Single-Cube Feeders

Generally speaking, every calculation statement in a rule should have a corresponding feeder statement.

The simplest feeders are those that apply to calculations within one dimension of one cube, where the presence of a non-zero value in one element implies the presence of a non-zero value in another element.

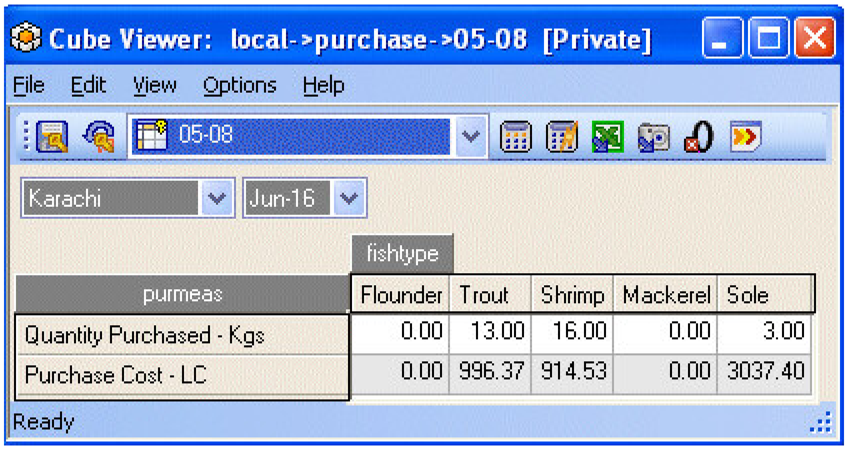
To illustrate this, recall the first calculation statement you created for the Purchase cube in the previous section.

```
['Purchase Cost - LC']=N:['Quantity Purchased - Kgs']*['Price/Kg
- LC'];
```

This statement calculates values for the area Purchase Cost - LC. Looking at the statement, you can surmise that if the value for Quantity Purchased - Kgs is zero, the rule returns zero for Purchase Cost - LC. If the value for Quantity Purchased - Kgs is non-zero, then the rule returns a non-zero value as well. (This assume that for a given Quantity Purchased - Kgs there is always a non-zero value for Price/Kg - LC.) In this case, Purchase Cost - LC contains a value only when Quantity Purchased - Kgs contains a value.

To write accurate feeders, you need to focus on which component(s) of a formula determine when a non-zero value is returned to the area.

For example, in the calculation statement above, Purchase Cost - LC is derived from a product involving Quantity Purchased - Kgs. A view of the Purchase cube might look like this:



purmeas	Flounder	Trout	Shrimp	Mackerel	Sole
Quantity Purchased - Kgs	0.00	13.00	16.00	0.00	3.00
Purchase Cost - LC	0.00	996.37	914.53	0.00	3037.40

If nothing is bought (Quantity Purchased - Kgs is zero), then nothing is paid (Purchase Cost - LC is also zero). In this case, Quantity Purchased - Kgs is the component of the formula that determines when the rule returns a non-zero value.

Knowing this fact, you can construct the required feeder as follows:

```
['Quantity Purchased - Kgs']=>['Purchase Cost - LC'];
```

This feeder says that when there is a non-zero value for Quantity Purchased - Kgs, TM1 should feed a placeholder to Purchase Cost LC.

You can use this same approach to create feeders for the other calculation statements in the rule for the Purchase cube.

The second calculation statement in the rule calculates values for the area Price/Kg - USD.

```
['Price/Kg - USD']=N:['Price/Kg -LC']\DB('CurrencyExchangeRate', DB('Currency',
!market,'MarketCurrency'),!date);
```

Here, the value of Price/Kg - USD is dependent upon the value of Price/Kg -LC. Specifically, Price/Kg - USD is a quotient dependent upon the dividend Price/Kg -LC. You should use Price/Kg -LC to feed Price/Kg - USD.

```
['Price/Kg -LC']=>['Price/Kg - USD'];
```

The final calculation statement in the Purchase rule calculates values for the area Purchase Cost - USD.

```
['Purchase Cost - USD']=N:['Purchase Cost - LC']\DB('CurrencyExchangeRate',
DB('Currency',!market,'MarketCurrency'),!date);
```

The value of Purchase Cost - USD is dependent upon the value of Purchase Cost - LC. Similar to the preceding calculation statement, Purchase Cost - USD is a quotient dependent upon the dividend Purchase Cost - LC. You should use 'Purchase Cost - LC to feed Purchase Cost - USD.

```
['Purchase Cost - LC']=>['Purchase Cost - USD'];
```

Adding Feeders to the Rule for the Purchase Cube

The following steps restore the sparse consolidation algorithm to the Purchase cube.

Procedure

1. Double-click the rule for the Purchase cube in the **Server Explorer**.
2. Insert the SKIPCHECK; declaration at the top of the **Rules Editor**. The SKIPCHECK; declaration must be the first line in the rule.
3. Add the FEEDERS; declaration after the last calculation statement.
4. Enter the following three feeder statements immediately following the FEEDERS; declaration.

   ```
   ['Quantity Purchased - Kgs']=>['Purchase Cost - LC'];['Price/Kg -LC']=>['Price/Kg
   - USD'];['Purchase Cost - LC']=>['Purchase Cost - USD'];
   ```

The complete rule for the Purchase cube should now appear as follows:

```
SKIPCHECK;
```

```
['Purchase Cost - LC']=N:['Quantity Purchased - Kgs']*['Price/Kg
-
LC'];
```

```
['Price/Kg - USD']=['Price/Kg - LC']\DB('CurrencyExchangeRate',
DB('Currency',!market,'MarketCurrency'),!date);
```

```
['Purchase Cost - USD']=N:['Purchase Cost - LC']\DB('CurrencyExchangeRate',
DB('Currency',!market,'MarketCurrency'),!date);
```

```
FEEDERS;
```

```
['Quantity Purchased - Kgs']=> N:['Purchase Cost - LC'];
```

```
['Price/Kg - LC']=>['Price/Kg - USD'];
```

```
['Purchase Cost - LC']=>['Purchase Cost - USD']'
```

5. Click **Save**.

Feeding One Cube from Another

Things become a little more difficult when you need to write feeders for inter-cube calculation statements, such as the calculation statements in the rule for the Inventory cube that retrieve values from the Purchase cube.

As you recall, the calculation statements are relatively simple. Residing in the rule for the Inventory cube, they look like this:

```
['1','Quantity in Stock - Kgs']= N:DB('Purchase',!FishType,'Total Markets',!
Date,'Quantity
Purchased - Kgs');['1','Purchase Cost'] = N:DB('Purchase',!FishType,'Total
Markets',!Date,'Price/Kg - USD');
```

The feeders for these statements are simple, but they must reside in the rule for the Purchase cube, not the Inventory cube. When working with inter-cube rules, calculation statements reside in the target cube, while feeder statements always reside in the source cube.

The following example guides you through the creation of feeder statements for the Inventory cube.

Procedure

1. Open the rules for both the Inventory and Purchase cube.

   When creating inter-cube feeders, for best practice, have the **Rules Editor** for both the source cube and target cube open in your workspace.
2. Position the cursor at the beginning of a new line after the last feeder statement in the Purchase rule.
3. Enter the following text:

```
#Feeders for the Inventory cube follow
```

   This line is a comment that makes the rule more understandable.

4. On a new line beneath the comment, enter the following feeder statement, either by typing or by using the toolbar buttons in the **Rules Editor**:

   ```
   ['Total Markets','Quantity Purchased - Kgs']=>DB('Inventory',!FishType,!Date,'
   1','Quantity in Stock - Kgs');
   ```

   This feeder statement ensures that any location in the Purchase cube identified by the elements Total Markets and Quantity Purchased - Kgs feeds the analogous location in the Inventory cube identified by the elements 1 and Quantity in Stock - Kgs.

   The feeder is basically the inverse of the calculation statement in the Inventory cube that requires the feeder:

   ```
   ['1','Quantity in Stock - Kgs']= N:DB('Purchase',!FishType,'Total
   Markets',!Date,'Quantity Purchased - Kgs');
   ```

5. On a new line below the feeder statement you just created, enter the next feeder statement:

   ```
   ['Total Markets','Price/Kg - USD']=>DB('Inventory',!fishtype,!date,'1','Purchase
   Cost');
   ```

   Again, note that this feeder statement is the inverse of the calculation statement that requires the feeder:

   ```
   ['1','Purchase Cost'] = N:DB('Purchase',!FishType,'Total Markets',!Date,'Price/Kg
   - USD');
   ```

   The Rules Editor for the Purchase rule should now appear as follows:

   ```
   SKIPCHECK;
   ```

   ```
   ['Purchase Cost - LC']=N:['Quantity Purchased - Kgs']*['Price/Kg
   -
   LC'];
   ```

   ```
   ['Price/Kg - USD']=['Price/Kg - LC']\DB('CurrencyExchangeRate',DB('Currency',!
   market,'
   MarketCurrency'),!date);
   ```

   ```
   ['Purchase Cost - USD']=N:['Purchase Cost - LC']\DB('CurrencyExchangeRate',
   DB('Currency',!market,'MarketCurrency'),!date);
   ```

   ```
   FEEDERS:
   ```

   ```
   ['Quantity Purchased - Kgs']=N:['Purchase Cost - LC'];
   ```

   ```
   ['Price/Kg - LC']=>['Price/Kg - USD'];
   ```

   ```
   ['Purchase Cost - LC']=>['Purchase Cost - USD']'
   ```

   ```
   #Feeders for the Inventory cube follow
   ```

   ```
   ['Total Markets','Quantity Purchased - Kgs']=>DB('Inventory',!FishType,!Date,
   '1','Quantity in Stock - Kgs');
   ```

   ```
   ['Total Markets','Price/Kg - USD']=>DB('Inventory',!fishtype,!date,'1','Purchase
   Cost');
   ```

6. Click **Save** in the **Rules Editor** for the Purchase rule.

Multi-threaded feeders

Multi-threaded feeders improve the performance of bulk feeder construction and cube feeder updates by leveraging the amount of available CPU cores.

Important:

Multi-threaded feeders are **not supported** when your Cognos TM1 model uses conditional feeders. When issues are detected, you **must** disable the multi-threaded loading of individual cubes.

Multi-threaded query capabilities extend to feeders by the parallelization of processing steps. Configuration settings in the tm1s.cfg file enable/disable multi-threaded feeders and feeder construction at start-up. To apply multi-threaded feeder functionality, feeders must be well-defined to arrive at the same result, no matter what sequence of cell changes led to feeder generation.

For example, setting CellA = N and then setting CellB = M should render the same feeder results as setting CellB = M and then setting CellA = N.

While this requirement is not unique to multi-threaded feeders, it can have a more prominent effect when MTFeeders is enabled. When cell values are modified by users or Turbo Integrator, feeders are updated in response to individual cell changes and follow the sequence of cell updates. However, feeder regeneration (for example, during server startup) processes cells in the order they were stored in the cube. This example shows how sequence-dependent feeder definitions can lead to different generated feeder values even in single threaded mode.

For more information on multi-threaded feeder configuration options, refer to "Parameters in the tm1s.cfg File" in *Planning Analytics Installation and Configuration*.

Troubleshooting Feeders

TM1 provides a tool called the **Rules Tracer** to assist in the development and debugging of rules.

When developing feeders, the **Rules Tracer** lets you:

- Trace feeders, to ensure that selected leaf cells are feeding rules-calculated cells properly
- Check feeders, to ensure that the children of a selected consolidated cell are fed properly

Rules Tracer functionality is available only in the **Cube Viewer**.

Tracing Feeders

The Rules Tracer lets you trace the way a selected cell feeds other rules-calcluated cells.

Because you can only feed from a leaf element, this option is not available for consolidated cells. The option is, however, available for leaf cells defined by rules.

Procedure

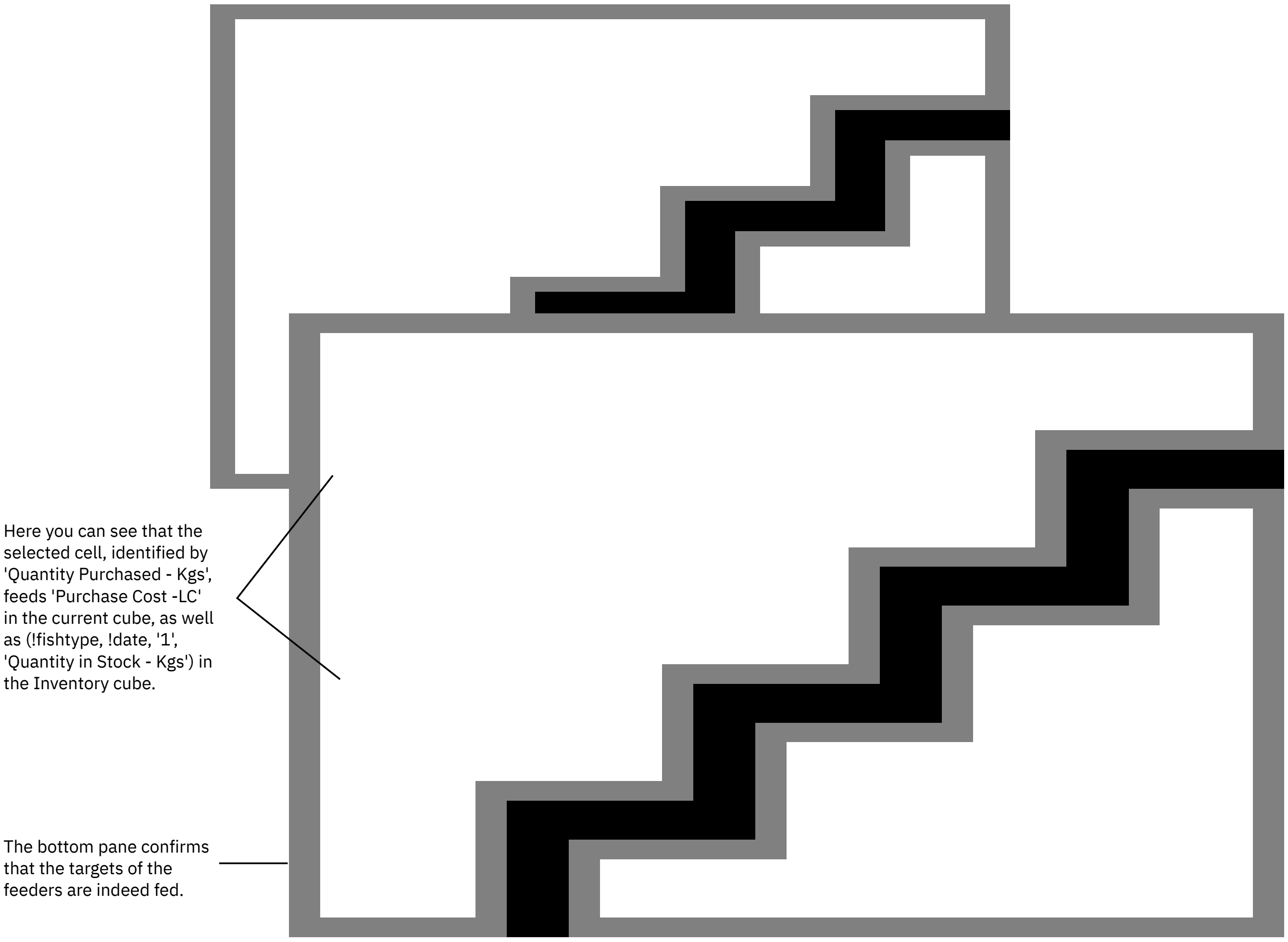
1. In the **Cube Viewer**, right-click the cell you want to trace.
2. Select **Trace Feeders**.

   The **Rules Tracer** window opens. This window contains two panes.

   The top pane displays the definition of the current cell location, as well as the feeder statements associated with the current cell.

   The bottom pane displays the locations fed by the current cell.

   The following example shows the result of selecting the **Trace Feeders** option from the highlighted cell in the Jun - 16 view of the Purchase cube. (This example shows a feeder statement that feeds cells in the Inventory cube. You'll create this feeder statement in a later section.)


Here you can see that the selected cell, identified by 'Quantity Purchased - Kgs', feeds 'Purchase Cost -LC' in the current cube, as well as (!fishtype, !date, '1', 'Quantity in Stock - Kgs') in the Inventory cube.
The bottom pane confirms that the targets of the feeders are indeed fed.

3. Double-click a location in the bottom pane.

   This location becomes the current cell location in the top pane, and the bottom pane displays any locations fed by the new current cell.
4. Continue double-clicking locations in the bottom pane until you have traced the feeders to the level you require.

Checking Feeders

If a cube has a rule attached that uses SKIPCHECK and FEEDERS, you can use the **Rules Tracer** to check that the components of consolidations that are subject to rules are properly fed.

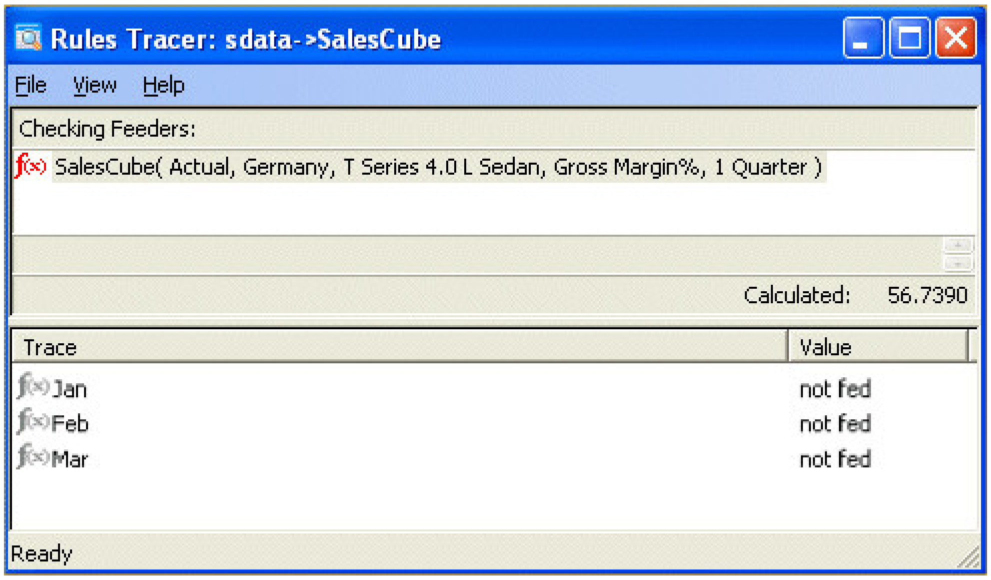
Before you begin

You can check feeders **ONLY** from a consolidated cell; the **Check Feeders** option is not available from leaf cells.

Procedure

1. In the **Cube Viewer**, right-click the consolidated cell you want to check.
2. Select **Check Feeders**.

   The **Rules Tracer** window opens. This window contains two panes.



Checking Feeders:

SalesCube(Actual, Germany, T Series 4.0 L Sedan, Gross Margin%, 1 Quarter)

Calculated: 56.7390

Trace	Value
Jan	not fed
Feb	not fed
Mar	not fed

Ready

The top pane displays the definition of the current cell (consolidation).

The bottom pane displays all components of the consolidation that are not properly fed.

If the bottom pane is empty, the consolidation is fed properly and cube values are accurate.

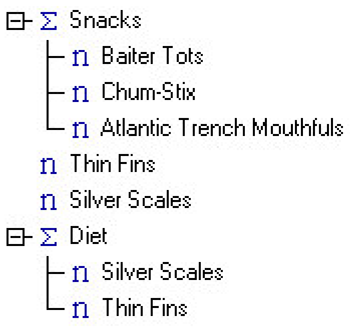
If the bottom pane displays components of the consolidation, you must edit the rule associated with the current cube to contain feeder statements that feed all the listed components.

The example shows the result of checking feeders for the '1 Quarter' consolidation. The bottom pane shows that none of the children of the consolidation ('Jan', 'Feb', or 'Mar') are fed. When the children of a consolidation that is subject to rules are not fed, cube values can be inaccurate!

Chapter 6. Moving Data and Changing Levels

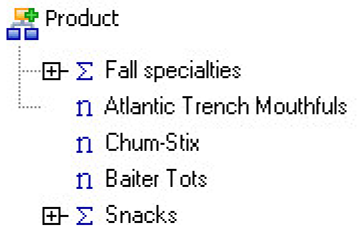
In you moved data between cubes that shared some common dimensions. In many cases, cubes have dimensions that are identical except for level.

For example, one cube may use a product dimension with leaf elements at the individual level like this:



```
Snacks
    Baiter Tots
    Chum-Stix
    Atlantic Trench Mouthfuls
Thin Fins
Silver Scales
Diet
    Silver Scales
    Thin Fins
```

A second cube may use a similar dimension, but omit the individual product elements and have leaf elements at the product level:



```
Product
    Fall specialties
        Atlantic Trench Mouthfuls
        Chum-Stix
        Baiter Tots
    Snacks
```

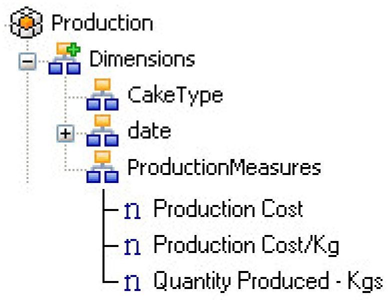
Chapter 4, “Using DB Functions to Move Data Between Cubes,” on page 45 Such a situation often arises when separate cubes are used for planning and actuals data. Typically, actuals data is kept in great detail, while planning data is kept in a less detailed form.

In this section, you'll discover how you can use rules to move data from more-detailed cubes to less-detailed cubes and to enter data at multiple levels of detail. This type of scenario often arises when a model includes budgeting and planning for production data where the granularity of production is more detailed than that of the budget (monthly versus daily).

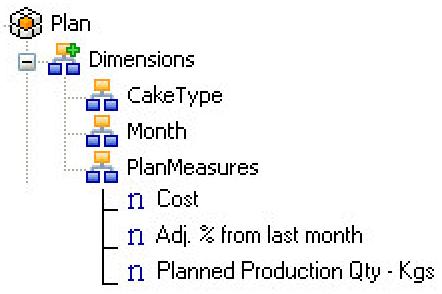
The Production and Plan Cubes

It is not a coincidence that Fishcakes International has exactly such a problem.

Their database keeps information on production by day, but they only plan by month. Thus they have a pair of cubes, Production and Plan:



```
Production
    Dimensions
        CakeType
        date
        ProductionMeasures
            Production Cost
            Production Cost/Kg
            Quantity Produced - Kgs
```



```
Plan
    Dimensions
        CakeType
        Month
        PlanMeasures
            Cost
            Adj. % from last month
            Planned Production Qty - Kgs
```

The key dimensional difference between these cubes is in their handling of time. Actual values in the Production cube are tracked by day in the Date dimension, but values in the Plan cube are tracked only by month in the Month dimension.

As the months go by, Fishcakes International executives want to see previous months data from the Production cube appear in the appropriate months in the Plan cube. Thus, January values from Production, which are consolidated, should appear as January values in the Plan cube, where January is a leaf element. The reason Fishcakes International maintains a separate cube for Plan data is not to see the monthly aggregates created in the Production cube (this you could easily do directly in the Production cube) but rather to allow for the manual entry of plan numbers at the month level directly into the Plan cube.

Creating a Calculation Statement for Planned Production Qty - Kgs

Using the **Rules Editor**, you could write the following calculation statement for the Plan cube.

This statement attempts to calculate values for Planned Production Qty - Kgs by retrieving analogous values for Quantity Produced - Kgs from the Production cube:

```
['Planned Production Qty - Kgs']=DB('Production',!CakeType,!Date,'Quantity
Produced - Kgs');
```

However, when you attempt to compile this statement, the following error message displays.

```
Line 1: Syntax error on or before: !Date,'Quantity Prod invalid string expression Rule
could not be attached to the cube, but changes were saved.
```

What Is the Problem?

The preceding rule statement includes a DB function that references the Date dimension.

(The !Date argument to the DB function instructs the TM1 server to use the current element of the Date dimension.) Unfortunately, the Plan cube for which the rule is written does not include the Date dimension. The server is unable to resolve the !Date argument, resulting in the error.

The Plan cube does contain the Month dimension, which shares common element names with the Date dimension. If you substitute a reference to the Month dimension for the original reference to the Date dimension, the rule compiles and correctly moves data from the Production cube into the Plan cube.

```
['Planned Production Qty - Kgs']=DB('Production',!CakeType,!Month,'Quantity
Produced - Kgs');
```

Follow the steps below to create the calculation statement for the Plan cube.

Procedure

1. Right-click the Plan cube in the **Server Explorer**, and click **Create Rule**.
2. Enter the calculation statement in the **Rules Editor**.
3. Click **Save** to save the rule.
4. Confirm that the rule works by opening the Jan - Mar view for both the Production and Plan cube. The Jan - Mar view of the Plan cube pulls values from the identically structured view of the Production cube.

How It Works

The trick is that even though the source cube (Production) does not have a Month dimension, the DB function scans the referenced cube for dimension elements whose names match those of elements in the equivalently ordered dimension in the target cube. In this case, the DB function looks for elements in the Month dimension that match element names in the Date dimension.

For example, when the following rule statement exists in the rule for the Plan cube, suppose the TM1 server processes a request for the value in the Plan cube at the coordinates Total Fish Cake Types, January, Planned Production Qty - Kgs.

```
['Planned Production Qty - Kgs']=DB('Production',!CakeType,!Month,'Quantity
Produced  - Kgs');
```

When this statement is executed, the server performs the following actions:

Procedure

1. It checks if the requested value from the Plan cube corresponds to the area definition of the rule statement. In this example, the requested value lies at the intersection of Total Fish Cake Types, January, Planned Production Qty - Kgs, so it does correspond to the area definition.
2. The server then looks in the Production cube for a value whose CakeType element is Total Fish Cake Types (the CakeType element for the value being requested in the Plan cube), and whose second dimension element corresponds to the Month element for the value being requested. In this case, the second dimension in the Production cube is Date, which contains the element January. This element exactly corresponds to the current Month element for the value being requested in the Plan cube.

   Note that the server does not try to verify that the Month dimension is part of the Production cube. It simply verifies that the current element in the Month dimension is a valid argument to the DB function.
3. The server uses Quantity Produced - Kgs as the final argument to the DB function.
4. The DB function returns a value for Planned Production Qty - Kgs.

Writing the Required Feeder

As with previous rules you've written, you now need to write a feeder that lets the consolidation engine know where there are rules-calculated values in the Plan cube.

Keeping in mind the Date/Month discrepancy reviewed above, the following feeder statement in the rule for the Production cube (recall that the feeder in an inter-cube rule always goes in the rule for the source cube) is accurate:

```
['Quantity Produced - Kgs']=>DB('Plan',!CakeType,!Date,'Planned
Production
Qty - Kgs');
```

For a given CakeType and Date, this feeder statement feeds Planned Production Qty - Kgs in the Plan cube when the Production cube contains a value for Quantity Produced - Kgs. Note that the DB function in the feeder statement references the Date dimension, which is not a member of the Plan cube. (Month is actually the second dimension of the Plan cube.) As with the calculation statement above, this works because the Date and Month dimensions share common element names.

Procedure

1. Double-click the Plan rule in the **Server Explorer**.
2. Enter a SKIPCHECK; declaration at the top of the rule.
3. Click **Save** in the **Rules Editor**.
4. Right-click the Production cube in the **Server Explorer** and click **Create Rule**.

   The **Rules Editor** displays.
5. Enter a FEEDERS; declaration in the **Rules Editor**.
6. Enter the following feeder statement immediately following the FEEDERS; declaration:

   ```
   ['Quantity Produced - Kgs']=>DB('Plan',!CakeType,!Date,'Planned
   Production
   Qty - Kgs');
   ```
7. Click **Save** in the **Rules Editor**.

Chapter 7. Rules for Time-Based Calculations

In this section, you learn how you can use IBM Cognos TM1 rules functions to implement a variety of sophisticated time calculations. Time-based calculations appear in nearly all TM1 applications. Budgeting, planning, forecasting, trending moving averages, and cumulative sales totals all require time calculations.

While rules do not have explicit functions to do leads and lags, to calculate time series or step through months and years, you can implement all of these functions by using combinations of functions.

In this section, you practice the use of the DIMIX function and its inverse, the DIMNM function. DIMIX returns the index of a given dimension element - its position in the dimension; DIMNM returns the dimension element given the index.

You also learn about the DNEXT function, which returns the next dimension element, and TIMVL, which does some date conversions. You also examine the use of the STET function to allow data input.

To illustrate the flexibility of these functions, you solve one problem several ways, each time approaching the problem from a different angle.

The Problem

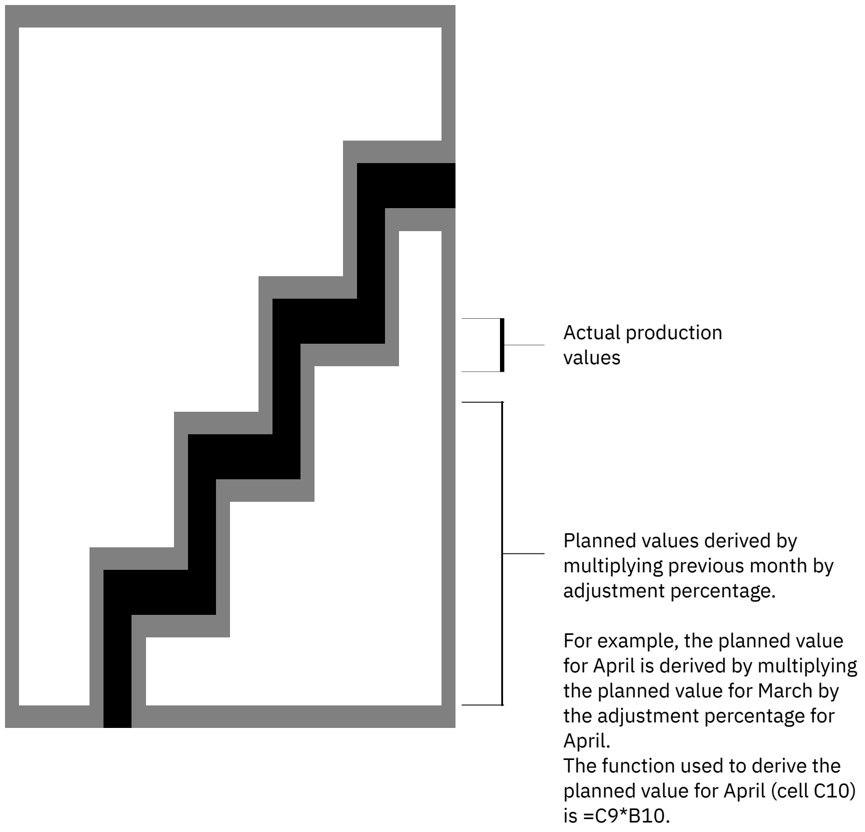
You have moved actual Production data into the Plan cube in the Fishcakes International model, but you still have more to do.

The point of a plan is not simply to restate actuals, but to estimate the future. This requires doing various types of time series calculations.

Many companies like to produce reports combining actuals with projections. Numbers for past months represent actual performance; those for future months are best-guess projections.

Fishcakes International puts both types of data into one Plan cube. Actual values are derived through calculation statements that reference the Production cube; future months are calculated based on performance so far, combined with a month-by-month percentage change in production. Each future month's projection is calculated by applying the standard percentage change for that month to the value for the previous month.

In a spreadsheet, the calculation to adjust values on a monthly basis looks like this. In this section, you create such projections in TM1 using rules.


Actual production values
Planned values derived by multiplying previous month by adjustment percentage.
For example, the planned value for April is derived by multiplying the planned value for March by the adjustment percentage for April.
The function used to derive the planned value for April (cell C10) is =C9*B10.

Four Solutions

To illustrate the options available, this section explores several ways to tackle this problem, presented in order of increasing complexity.

Each solution has different strengths and weaknesses. In the real world, no two applications are identical; any one of these approaches may be more appropriate for one situation than for another.

First Try: Sequential Monthly Statements

The simplest solution is to write a separate rule statement for each month.

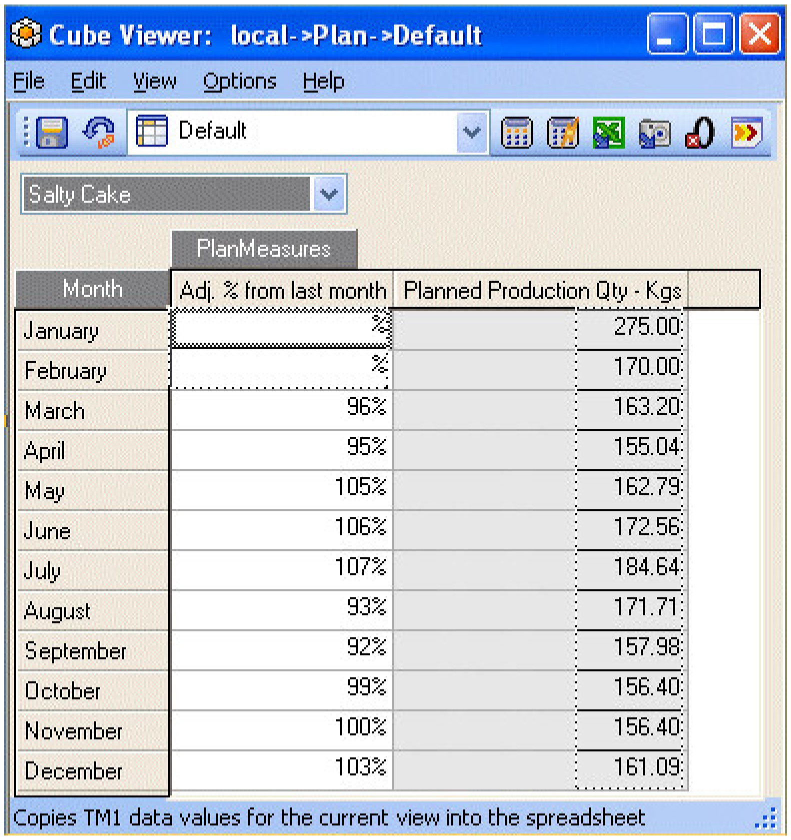
For previous months, you want to bring in actuals from the Production cube; for future months, you want to generate estimates within the Plan cube by applying the Adj. % from last month number. Assuming that you are writing the rules in March, this means that you take actuals values for January and February from the Production cube, then generate the rest of the values from the adjustment percentages.

The rule statements for the Plan cube would look like this:

```
['January','Planned Production Qty - Kgs']=
DB('Production',!CakeType,!Month,'Quarterly Producd - Kgs');

['February','PlannedProduction Qty - Kgs']=
DB('Production',!CakeType,!Month','Quarterly Produced - Kgs');

['March','Planned Production Qty - Kgs']=
['February','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'March','Adj. % from last month');

['April','Planned Production Qty - Kgs']=
['March','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'April'.'Adj. % from last month');
```

```
['May','Planned Production Qty - Kgs']=
['April','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'May','Adj. % from last month');

['June','Planned Production Qty - Kgs']=
['May','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'June','Adj. % from last month');

['July','Planned Production Qty - Kgs']=
['June','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'July','Adj. % from last month');

['August','Planned Production Qty - Kgs']=
['July','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'August','Adj. % from last month');

['September','Planned Production Qty - Kgs']=
['August','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'September','Adj. % from last month');

['October','Planned Production Qty - Kgs']=
['September','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'October', 'Adj. % from last month');

['November','Planned Production Qty - Kgs']='
['October','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'November'.'Adj. % from last month');

['December','Planned Production Qty - Kgs']=
['November','Planned Production Qty - Kgs']
*DB('Plan',!CakeType,'December'.'Adj. % from last month');
```

As the following view of the Plan cube illustrates, this rule accomplishes the job of moving data into the Plan cube from the Production cube for the months of January and February. The rule also correctly calculates planned Production Quantity Values for the months of March through December.


Cube Viewer: local->Plan->Default
File Edit View Options Help
Default
Salty Cake
PlanMeasures

Month	Adj. % from last month	Planned Production Qty - Kgs
January	%	275.00
February	%	170.00
March	96%	163.20
April	95%	155.04
May	105%	162.79
June	106%	172.56
July	107%	184.64
August	93%	171.71
September	92%	157.98
October	99%	156.40
November	100%	156.40
December	103%	161.09

Copies TM1 data values for the current view into the spreadsheet

This solution has the advantage of being easy to write and understand, but has significant limitations:

- It is excessive to have to write twelve separate statements, one for each month.
- The statements need to be changed each month as the year advances (new values from the Production cube have to be brought into the Plan cube each month, requiring a change to the statements).
- If the associated feeders are similarly hard-coded, they too will need to change each month.

A Second Way: Using the DIMNM - DIMIX Idiom

TM1 dimension handling functions allow you to write much more powerful, and easy-to-maintain rules than those in the example.

A pair of functions, DIMIX and DIMNM, when used in combination, allow you to move forwards and backwards along dimensions.

The DIMIX function returns a number corresponding to the position, or index, of a given element in a dimension. The DIMNM function is its inverse, returning a dimension element given the index.

For example, `DIMIX ('Date', 'Nov -17')` returns the index of the element Nov -17 in the Date dimension, which happens to be 322. Correspondingly, `DIMNM ('Date', 322)` returns Nov -17, the 322nd element of the Date dimension.

The real power of these functions, though, is when they are combined.

`DIMNM('Date',DIMIX('Date', 'Nov -17'))` uses the DIMIX function to retrieve the index of Nov -17, then passes that index to the DIMNM function to retrieve the element with that index. Thus, it returns what it started with, Nov -17.

`DIMNM('Date',DIMIX('Date', 'Nov -17') -1)` retrieves the index of Nov -17 (322), subtracts 1 from it to get 321, and returns the element with that index. It therefore returns Nov -16, the element immediately preceding Nov -17.

Similarly, `DIMNM('Date',DIMIX('Date', 'Nov -17') +1)` returns Nov -18, the element immediately following Nov -17.

Perhaps most useful, though, a DB function containing the expression `DIMNM('Date',DIMIX('Date', !Date) - 1)` will "step back" through every element of the Date dimension and relative to each date element, retrieve the previous date.

Using this construction, you can replace the ten month-specific statements in the Plan rule that calculate future planned production quantities with one (somewhat more complex) statement.

```
['January','Planned Production Qty - Kgs']=DB(Production',!CakeType,!Month,'Quarterly
Produced - Kgs');
```

```
['February','Planned Production Qty - Kgs']=DB(Production',!CakeType,!Month,'Quarterly
Produced - Kgs');
```

```
['Planned Production Qty - Kgs']=DB('Plan',!CakeType,DIMNM('Month',
(DIMIX('Month',!Month)-1)),['Planned Production Qty - Kgs'] *
['Adj. % from last month'];
```

The single calculation statement

```
['Planned Production Qty - Kgs']=DB('Plan',!CakeType,DIMNM('Month',
(DIMIX('Month',!Month)-1)),['Planned Production
Qty - Kgs'] * ['Adj. % from last month'];
```

accurately retrieves Planned Production Qty - Kgs values for the months March through December. This statement uses a DB formula to refer to the current cube. (As you may recall, the DB formula is much more flexible than bracket notation. In this case, a DIMNM function returns the argument for the Month dimension to the DB formula. Such formulas cannot be used in bracket notation.)

To better understand how this statement works, consider what happens when the server receives a request for the value of Planned Production Qty - Kgs of Salty Cake in the month of March:

Procedure

1. The server evaluates `!Month` to March, the month that is being requested.
2. `DIMIX('Month', 'March')` evaluates to 3, the index value of March in the Month dimension.
3. 3-1 evaluates to 2.
4. `DIMNM('Month',2)` evaluates to February.
5. `!CakeType` evaluates to Salty Cake, the caketype of the value that is being requested.
6. `DB('Plan', 'Salty Cake', 'February', 'Planned Production Qty - Kgs')` evaluates to 170.00.
7. `['Adj. % from last month']` for Salty Cake and March evaluates to 96%. (Remember, the server is processing a request for a value in March.)
8. The server multiplies the value of 170.00 by 96% to return 163.20, the Planned Production Qty - Kgs of Salty Cake in the month of March.

   This new statement has the benefit of replacing ten statements with one. However, the rule for the Plan cube still includes two month-specific statements that pull values from the Production cube, and a new month-specific statement will have to be added to the rule each month to pull the latest actual values into the Plan cube.

   Ideally, the rule for the Plan cube should use a single statement to retrieve values for all months, and it should not be necessary to edit the rule from month-to-month.

A Third Approach: Using DIMIX for Comparisons

This next example replaces all the calculation statements in the rule for the Plan cube with a single conditional statement that calculates allPlanned Production Qty - Kgs values, whether actual or projected.

Because the months are arranged in sequential order in the Month dimension, the index values for the earlier months of the year are smaller than the index values for later months. You can use this fact to your advantage in constructing a statement that pulls values for all months before March from the Production cube, while calculating values for March and later months directly in the Plan cube.

```
['Planned Production Qty - Kgs']=IF(DIMIX('Month',!Month)<DIMIX('Month',
'March'),DB('Production',!CakeType,!Month,'Quantity
Produced - Kgs'),DB('Plan',!CakeType,DIMNM('Month',(DIMIX('Month',
!Month)-1)),'Planned Production Qty - Kgs')*['Adj. % from last month']);
```

This single statement accurately retrieves and calculates values for all months. If the value requested is for a month prior to March, the function `DB('Production',!CakeType,!Month,'Quantity Produced - Kgs')` retrieves the value from the Production cube. If the value requested is for the month of March or later, the formula `DB('Plan',!CakeType,DIMNM('Month',(DIMIX('Month',!Month) -1) ),'Planned Production Qty - Kgs')* ['Adj. % from last month']` calculates the value.

This single statement is an improvement over previous solutions, but it uses a literal month name as an argument to the second DIMIX function. You must edit this month name as you progress from month to month.

Ideally, though, you should not need to edit a rules statement on a monthly basis.

One possible solution would be to use the TIMVL function, which returns the numeric value of a component of a date/time value, in combination with the NOW function, which returns the date/time value of the current time. For example, `TIMVL(NOW,'M')` returns the numeric value of the month for the current time. Any time during the month of March, this function returns 3. This numeric value nicely coincides with the index value for the March element in the Month dimension. For more information on all rules functions, see *TM1 Reference*.

You can now fine-tune the single calculation statement in the rule for the Plan cube so that values for all months before the current month are pulled from the Production cube, while values for the current month and later are calculated in the Plan cube.

```
['Planned Production Qty - Kgs']=IF(DIMIX('Month',!Month)<TIMVL(NOW,'M'),
DB('Production',!CakeType,!Month,'Quantity
Produced - Kgs'),DB('Plan',!CakeType,DIMNM('Month',(DIMIX('Month',!Month)-1)),
'Planned Production Quantity - Kgs')*[Adj. % from last month']);
```

Using the TIMVL function eliminates the monthly rule change requirement. This type of solution is appropriate where you are interested in moment-by-moment time changes, for example in stock market applications.

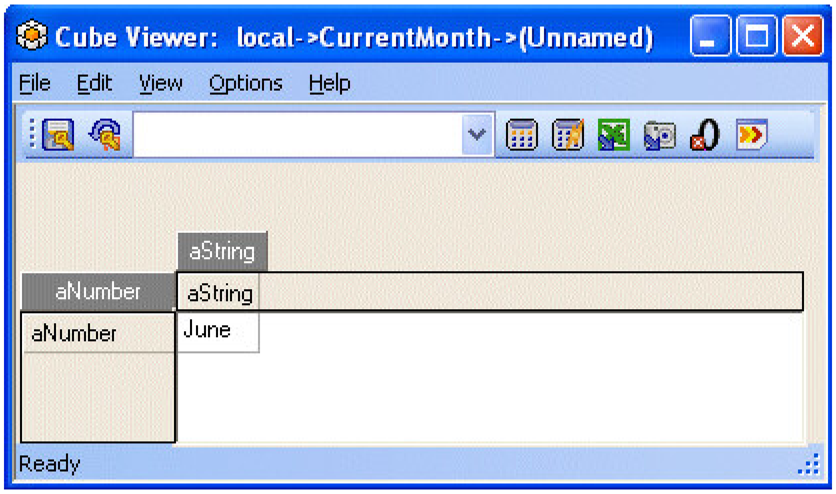
For a monthly budgeting application, though, this solution is less than ideal because you might want to control the month for which data is queried. After all, most companies do not have all their actuals ready on the first of the month. Also, you might want to set the month back later, to review projections as they appeared on a previous month.

Best Solution: Using a String Cube to Store a Variable

Instead of hard-coding the month or using a function to retrieve the current month, it would be best to create a variable with a user-controllable value, so that you can specify the current month.

This can be done with a string cube that is just big enough to hold the variable.

For example, the cube CurrentMonth is comprised of two dimensions, aNumber and aString. Each dimension contains a single element, so the cube contains just a single cell that can accept the name of the month that you want to set as current.



	aString
aNumber	aString
aNumber	June

To retrieve this value, you can use the function:

```
DB('CurrentMonth', 'aNumber', 'aString')
```

You can then pass the DB function as an argument to the DIMIX function to determine the index value of the month as follows:

```
DIMIX('MONTH',DB('CurrentMonth', 'aNumber', 'aString'))
```

This returns the index of the month in the CurrentMonth cube.

You can now further refine the rule statement to incorporate this reference to the CurrentMonth cube.

```
['Planned Production Qty - Kgs']=IF (DIMIX('Month',DB('CurrentMonth',
'aNumber','aString')), DB('Production',!CakeType,!Month,'Quantity
Produced - Kgs'), DB('Plan',!CakeType,DIMNM('Month',(DIMIX('Month',
!Month)-1)),'Planned Production Qty - Kgs')*[Adj. % from last month']);
```

This statement says that if the month index for the Planned Production Qty -Kgs value being requested is less than the index of the month contained in the CurrentMonth cube, retrieve the value from the Production cube, otherwise calculate the value by multiplying the Planned Production Qty -Kgs value of the previous month by the Adj. % from last month.

You can now control the calculation of values in the Plan cube by changing the string in the CurrentMonth cube, which is part of the sample data for this guide.

This solution satisfies all the requirements of a well-written rule. It uses only one rule statement for all months, and it makes it unnecessary for you or other users to edit the statement on a regular basis. Additionally, this solution allows you or other users to set the current month as required.

Creating the Calculation Statement for Planned Production Qty - Kgs

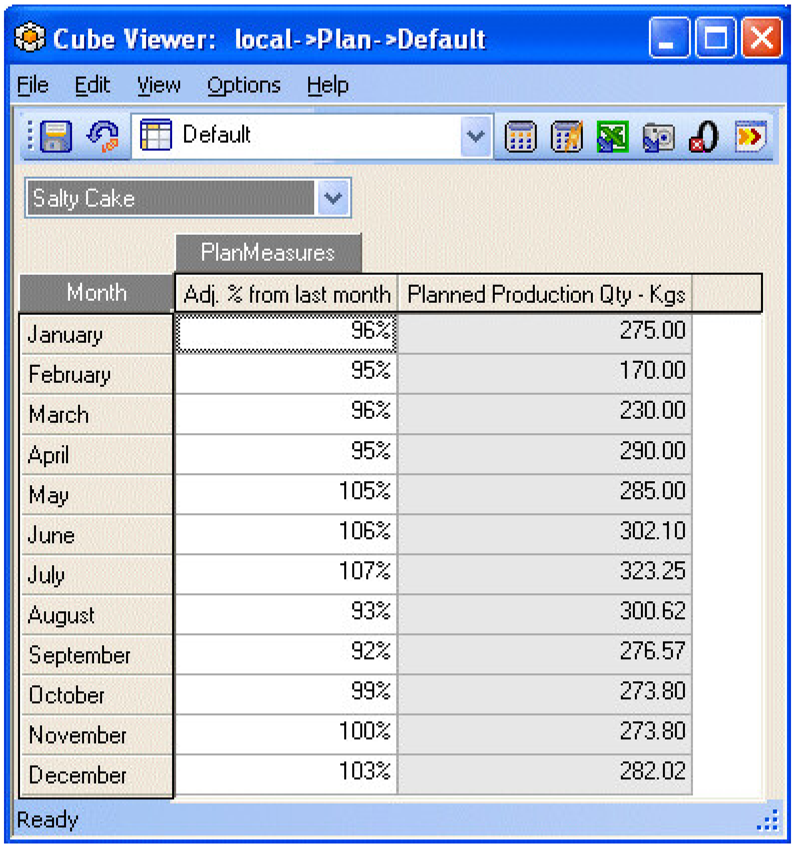
The following steps illustrate how to create the calculation statement that determines the value of Planned Production Qty -Kgs.

Procedure

1. Right-click the Plan cube in the **Server Explorer** and click **Edit Rule**.
2. Insert a pound sign **#** at the beginning of the existing calculation statement for ['Planned Production Qty - Kgs'].
3. Enter the following calculation statement below the last existing calculation statement in the **Rules Editor**.

```
['Planned Production Qty - Kgs'] = IF (DIMIX('Month',!Month)<DIMIX('Month',
DB('CurrentMonth','aNumber','aString')),DB('Production',!CakeType,
!Month,'Quantity Produced - Kgs'), DB('Plan',!CakeType,
DIMNM('Month',(DIMIX('Month',!Month)-1)),'Planned
Production
Qty - Kgs')* ['Adj. % from last month']);
```

4. Click **Save**.
5. Open the Salty Cake view of the Plan cube to see the result of the statement.



Cube Viewer: local->Plan->Default
File Edit View Options Help
Default
Salty Cake
PlanMeasures
Ready

Month	Adj. % from last month	Planned Production Qty - Kgs
January	96%	275.00
February	95%	170.00
March	96%	230.00
April	95%	290.00
May	105%	285.00
June	106%	302.10
July	107%	323.25
August	93%	300.62
September	92%	276.57
October	99%	273.80
November	100%	273.80
December	103%	282.02

Note that the values in this view vary depending on the month specified in the CurrentMonth cube. The above view reflects values when the CurrentMonth cube contains the string June. All Planned Production Qty - Kgs values for months prior to June are retrieved from the Production cube. All other Planned Production Qty - Kgs values are calculated by applying the Adj. % from last month to the previous monthly value.

Feeding Time Series

There are a variety of ways to feed time series calculations.

You are somewhat more limited in your options with feeder statements than with calculation statements, though, because you cannot limit the scope of the feeders to N: or C: elements. One consequence of this is that you must frequently create statements that result in overfeeding.

Overfeeding does not result in erroneous results; it simply forces the consolidation engine to scan more cells than necessary, resulting in longer calculation times. Depending on cube size and sparsity, this can have a very large effect or no visible effect at all. Before spending great efforts on avoiding over-feeding, it is worthwhile to give some thought to just how important an impact it will have.

Hard-Coded Feeders

Recall that the first attempt to create rules for the Plan cube in this section used sequential statements with hard-coded month names.

Analogous to those rules statements would be a set of feeders in the Plan rule as follows:

```
FEEDERS;
```

```
['February','Planned Production Qty - Kgs']=>['March','Planned
Production
Qty - Kgs'];
```

```
['March', 'Planned Production Qty - Kgs']=>['April','Planned
Production
Qty - Kgs'];
```

```
['April', 'Planned Production Qty - Kgs'] => ['May','Planned
Production
Qty - Kgs'];
```

```
['May', 'Planned Production Qty - Kgs'] => ['June','Planned
Production
Qty - Kgs'];
```

```
['June','Planned Production Qty - Kgs'] => ['July','Planned
Production
Qty - Kgs'];
```

. . . and so on until all months are fed.

The Produce cube would also require statements to feed January and February values in the Plan cube:

```
FEEDERS;
```

```
['January','Planned Production Qty -Kgs']=>DB('Plan',!caketype,
'January',
'Planned Production Qty -Kgs');
```

```
['February','Planned Production Qty -Kgs']=>DB('Plan',!caketype,
'February',
'Planned Production Qty -Kgs');
```

Like the calculation statements they accompany, the drawback to the above feeder statements is that they must be must be edited each month. As you proceed from month to month, the top feeder statement in the rules for the Plan cube must be deleted and a new statement must be added to the feeders for the Production cube, reflecting the fact that new monthly values are being pulled into the Plan cube from the Production cube.

It is worth noting, however, that this method has a significant advantage as well: the feeders are very precise with no overfeeding at all. In very large, sparse, calculation-intensive applications, feeders like this may produce significant performance improvement.

Deliberate Overfeeding

At the risk of possible massive overfeeding, you can reduce the number of feeder statements.

You do this by simply feeding the entire Month dimension with one feeder statement in the rule for the Plan cube:

```
['Planned Production Qty - Kgs'] => ['Planned Production
Qty - Kgs', 'Total Year'];
```

Feeding the consolidated element Total Year feeds all of its components; thus any value for Planned Production Qty - Kgs in any month will feed all the rest of the months.

In a small, dense cube like the Plan cube, the effect of overfeeding may well be negligible; and such a feeder has the great advantage of being very simple.

Using DNEXT for Feeding

Of course, the optimal feeder statement for the Plan cube share common characteristics of the optimal calculation statement for the cube.

A single statement should feed all months and it should be unnecessary to edit the statement on a regular basis.

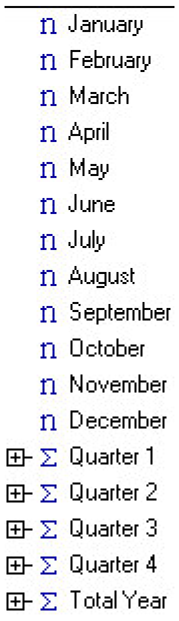
TM1 provides a rules function, DNEXT, which returns the next element of the dimension, given a specific dimension and element. For more information, see the *TM1 Reference* guide.

Using DNEXT, it is tempting to write the following feeder:

```
['Planned Production Qty -Kgs'] =>DB('Plan', !CakeType,DNEXT('Month',!Month),'Planned
Production Qty -Kgs');
```

When there is a Planned Production Qty -Kgs value for a given month, this statement does feed the following month. But on closer inspection, you can identify a problem with the statement.

Following December, the Month dimension contains the elements Quarter 1, Quarter 2, Quarter 3, Quarter 4, and Total Year.



The feeder would feed Q1 from December, Q2 from Q1 and so on through Total Year. Keeping in mind that feeding consolidated elements feeds all their components, this statement ends up feeding the entire cube.

One way to solve the problem that is evident with this feeder, is to incorporate an IF function so that when the index value for the current Month element is less than 12, the following month is fed, otherwise December is fed.

```
['Planned Production Qty -Kgs'] =>DB('Plan', !CakeType, IF(DIMIX('month',!Month)<12,
DNEXT('Month',!Month), 'December'),'Planned Production Qty -Kgs');
```

If there is sparse data in the Plan cube, this feeder significantly reduces overfeeding relative to the feeder statements considered previously. This feeder has the additional benefit of never feeding any element beyond December, thus avoiding the possibility of attempting to feed a non-existent element.

Creating the Feeder Statement to Feed
Follow the steps below to create the feeder statement that feeds Planned Production Qty -Kgs in the Plan cube.

Procedure

1. Double-click the Plan cube in the **Server Explorer**.
2. If it does not already exist, insert a SKIPCHECK; declaration immediately preceding the single calculation statement in the rule.
3. Insert a FEEDERS; declaration following the calculation statement.
4. Enter the following feeder statement immediately following the FEEDERS; declaration:

   ```
   ['Planned Production Qty -Kgs'] =>DB('Plan', !CakeType, IF(DIMIX('month',!Month)<12,
   DNEXT('Month',!Month), 'December'),'Planned Production Qty - Kgs');
   ```

   The complete rule for the Plan cube should now appear as follows:

   ```
   SKIPCHECK;
   ```

   ```
   ['Planned Production Qty - Kgs']=IF)(DIMIX('Month',!Month)<DIMIX('MONTH',
   DB('CurrentMonth','aNumber','aString')),DB('Production',!CakeType,
   !Month,'Quantity Produced - Kgs'),DB('Plan',!CakeType,DIMNM('Month',
   (DIMIX('Month',!Month)-1)),'Planned Production Qty - Kgs')*
   [Adj. % from last month']); FEEDERS;
   ```

   ```
   ['Planned Production Qty - Kgs']=>DB('Plan',!CakeType,IF(DIMIX('month',
   !Month)<12,DNEXT('Month',!Month),'December'),'Planned
   Production Qty - Kgs');
   ```

5. Click **Save**.

Chapter 8. Fixed Allocations

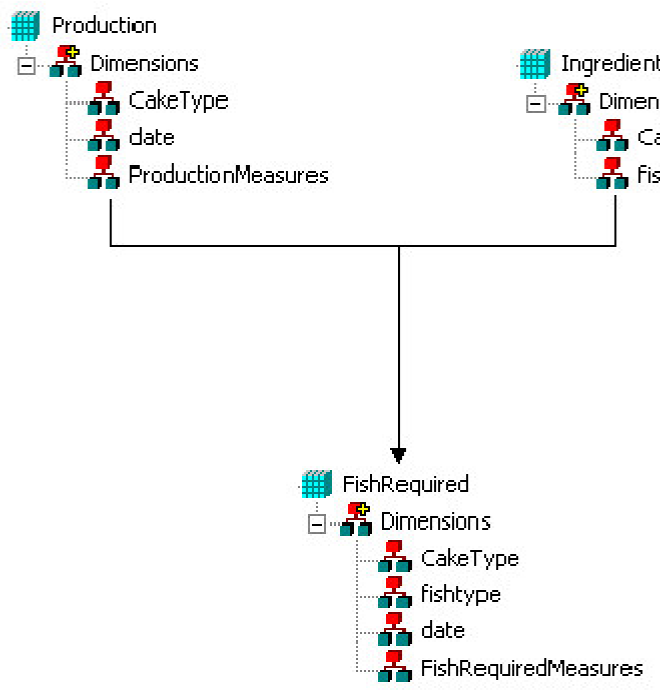
Fishcakes International wants to know the true input costs of each of its fishcakes every day. This is not a simple matter because each fishcake uses fish of various types from different markets, with different prices every day. Sometimes fish is used several days after it is purchased.

The company has all the information it needs to calculate these costs. As you saw in earlier sections, the company has a complete record of fish purchases. Further, the Production cube contains information on the amount of each type of fishcake produced each day. The challenge is to allocate quantities of fish to the production of individual fishcake types. Once this is accomplished, you can then allocate costs to fishcakes, as you will in the final section of this guide.

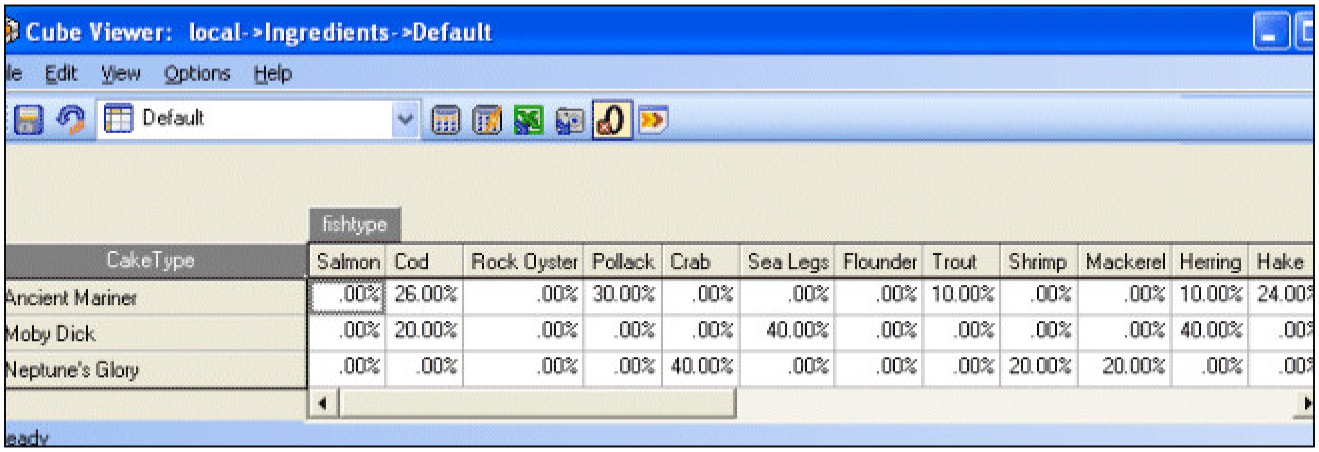
Calculating the Quantities of Fish Required by Fishcake Type

To allocate costs for each fish type to individual cake types, you must start by calculating how much of each fish type is used for the daily production of fishcakes.

A FishRequired cube holds this information, derived by applying "recipes" from a cube called Ingredients to the fishcake production values in the Production cube.



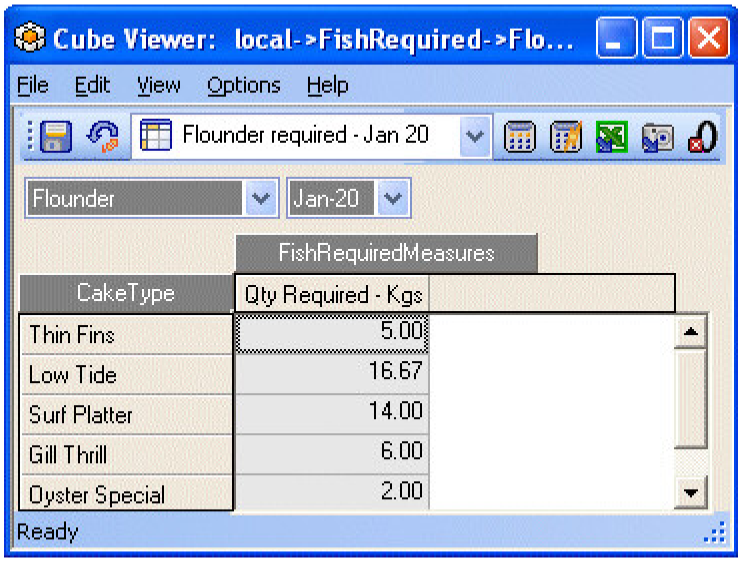
Ingredients is a two-dimensional cube that stores the percentage of each fish type used in the composition of fishcakes produced by Fishcakes International. For example, the following view of the Ingredients cube shows that the composition of the Ancient Mariner fishcake is 26% cod, 30% pollack, 10% trout, 10% herring, and 24% hake.



CakeType	Salmon	Cod	Rock Oyster	Pollack	Crab	Sea Legs	Flounder	Trout	Shrimp	Mackerel	Herring	Hake
Ancient Mariner	.00%	26.00%	.00%	30.00%	.00%	.00%	.00%	10.00%	.00%	.00%	10.00%	24.00
Moby Dick	.00%	20.00%	.00%	.00%	.00%	40.00%	.00%	.00%	.00%	.00%	40.00%	.00
Neptune's Glory	.00%	.00%	.00%	.00%	40.00%	.00%	.00%	.00%	20.00%	20.00%	.00%	.00

The FishRequired cube is a workspace where the amount of fish going into each fishcake is calculated. (This cube will later be used to calculate the materials cost of each fishcake.) Its dimensions are CakeType, FishType, Date and FishRequiredMeasures. The only measure you'll work with here is Qty Required - Kgs.

In the FishRequired cube, Qty Required - Kgs varies by fishcake type, fish type, and date. For example, the following view shows the amount of flounder required in the production of several different cake types on January 20.



Flounder Jan-20

CakeType	Qty Required - Kgs
Thin Fins	5.00
Low Tide	16.67
Surf Platter	14.00
Gill Thrill	6.00
Oyster Special	2.00

To correctly calculate the amount of each fish type required for a given fishcake type on a given day, you need to create a calculation statement for the FishRequired cube that multiplies the amount of each cake type produced (from the Production cube) by the percentage of each fish used in the cake type (from the Ingredients cube).

The statement that calculates Qty Required - Kgs values for the FishRequired cube therefore looks like this:

```
['Qty Required - Kgs']=N:DB('Production',!CakeType,!Date,'Quantity
Produced
- Kgs') * DB('Ingredients',!CakeType,!FishType);
```

Creating the Calculation Statement for Qty Required - Kgs

Follow the steps below to create this calculation statement in the rule for the FishRequired cube.

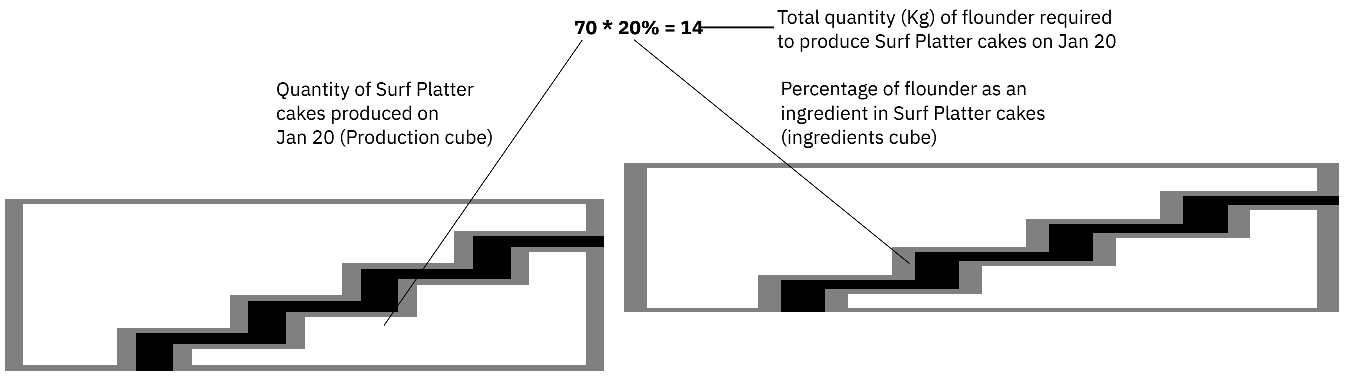
Procedure

1. Right-click the FishRequired cube in the **Server Explorer** and click **Create Rule**.

   The **Rules Editor** opens.
2. Enter the calculation statement on the first line of the **Rules Editor**.

3. Click **Save** to save the rule.

   If the TM1 server receives a request for the quantity (in Kgs) of flounder required to make Surf Platter fish cakes on January 20th, the rule calculates as follows.



70 * 20% = 14
Total quantity (Kg) of flounder required to produce Surf Platter cakes on Jan 20
Quantity of Surf Platter cakes produced on Jan 20 (Production cube)
Percentage of flounder as an ingredient in Surf Platter cakes (ingredients cube)

4. Open the Flounder required - Jan 20 view of the Fish Required cube to confirm that the statement calculates Qty Required - Kgs values.

Feeding Qty Required - Kgs

Feeders belong in the cube where the values in a calculation statement reside. Because the formula in this calculation statement multiples values from both the Production and Ingredients cubes, the feeders could reside in the rule for either cube.

The most efficient place for the feeders is in the rule for the Production cube. This is because every CakeType is not produced on every date. A feeder in the rule for the Production cube will be triggered only when there is a value for a given cake type on a given date. The appropriate feeder statement looks like this:

```
['Quantity Produced - Kgs']=>DB('FishRequired', !CakeType,'Total
Fish
Types', !Date, 'Qty Required - Kgs');
```

Creating the Feeder Statement for Qty Required - Kgs

Follow the steps below to add this feeder to the rules for the Production cube.

Procedure

1. Double-click the FishRequired rule in **Server Explorer**.
2. Insert a SKIPCHECK; declaration immediately before the existing calculation statement in the rule.
3. Click **Save** to save the rule and exit the **Rules Editor**.
4. Double-click the Production rule in **Server Explorer**.
5. Click **Edit Rule**.

   The **Rules Editor** opens. The rules for the Production cube already contain a feeder statement that feeds the Plan cube.
6. Enter the feeder statement on a new line following the existing feeder statement.

The feeders for the Production rule should now appear as follows.

```
FEEDERS;
```

```
['Quantity Produced - Kgs']=>DB('Plan',!CakeType,!Date,'Planned
Production
Qty - Kgs');
```

```
['Quantity Produced - Kgs']=>DB('FishRequired',!CakeType,'Total
Fish
Types',!Date,'Qty Required - Kgs')'
```

7. Click **Save** to save the Production rule.

   This new feeder overfeeds the FishType dimension, because feeding Total also feeds all FishTypes elements for every CakeType produced on a given date. This is not uncommon when feeding from one cube to another, because the dimensionality of the target cube is typically greater than that of the source cube. In this example, the target cube FishRequired has the additional FishType dimension that is not part of the Production cube. In such a situation, overfeeding is inevitable.

Chapter 9. Stocks and Flows

In this section, you look at some ways to use IBM Cognos TM1 to keep track of stocks and flows. Many applications require some form of such calculations: P&L and balance sheet applications, inventory and production costing, and loan amortization, to name just a few.

You might think that such calculations - which are easiest to describe in a step-by-step, ''procedural" manner - cannot be expressed in a declarative language. In fact, they can be expressed economically and calculated efficiently using rules, and there are many large TM1 applications that perform such functions.

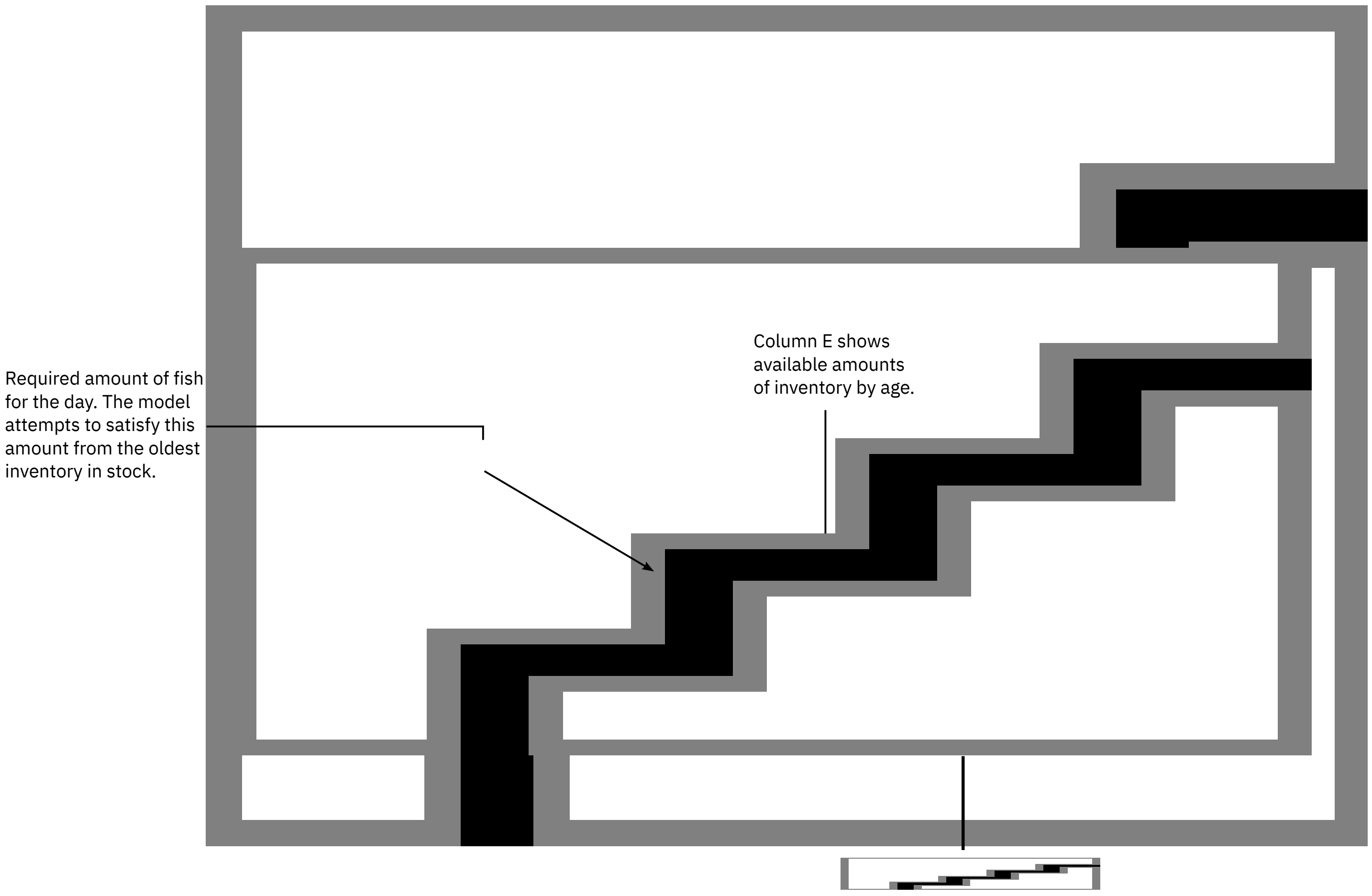
In earlier sections, you saw how Fishcakes International moves information on purchases into the Inventory cube, and how information on production from the Production cube is allocated in the FishRequired cube.

In this section, we use that information to calculate the daily cost of fishcake production as determined by the cost of each separately priced batch of fish coming from inventory. Remember that fish price varies from day to day and from market to market.

To do this, you take information from the FishRequired cube to calculate the daily requirements for each fish type. You then use information from the Inventory cube to determine the available fish in stock. Finally, a new cube called Depletion calculates how inventory is used to satisfy the daily requirements.

Depletion with a Spreadsheet

The following figure illustrates how the depletion calculation works with declarative logic in a spreadsheet. Row 15 shows the formulas used to calculate values in each column.


Required amount of fish for the day. The model attempts to satisfy this amount from the oldest inventory in stock.
Column E shows available amounts of inventory by age.

The spreadsheet works this way:

Procedure

1. Starting with the required amount of fish for the day (cell B6, which is referenced by cell D10), the model tries to satisfy the requirement using available fish, starting with the oldest fish.
2. The oldest fish available is 3 days old. The quantity available is 10.25 Kgs (cell E13), which is less than the quantity required, so the model uses all 10.25 Kgs (cell F13).
3. The model subtracts the quantity used from the quantity required and determines that 33.13 Kgs (cell H13) are still required.
4. The model then moves to the 2 days old batch of fish. The quantity still required from the 3 days old batch becomes the current required quantity (cell D14)for the 2 days old batch.
5. The quantity available from the 2 days old batch is 39 Kgs (cell E14), which is greater than the quantity required, so the model uses only the quantity required (cell F14).
6. The model subtracts the quantity used from the quantity required and determines that 0 Kgs (cell H14) are still required.
7. The depletion is complete. The model has successfully satisfied the requirements for 43.38 Kgs of trout on Jan. 10 by depleting inventory from the oldest available batches of fish.
8. The quantity of 2 days old trout remaining after depletion (5.87 Kgs, cell G14) will become the available quantity of 3 days old trout on Jan. 11.

Implementing a Depletion Model Using Rules

All the tools required to implement this plan are at your disposal in rules. Though the plan requires data from three cubes, most of the rules work is done in the Depletion cube.

You start by taking in the day's requirement for a fish type from the FishRequired cube. This requirement goes directly into DaysOld element 6, since we deplete starting with the oldest fish first.

```
['6','Required']=DB('FishRequired','Total Fish Cake Types',!FishType,!Date,'Qty
Required - Kgs');
```

This is a single number for any fish type on any date. For example, the total need for sole on January 1 is 30 Kgs. This rule statement puts that amount into the Depletion cube as the amount of sole required from DaysOld batch 6. As you further develop rules for the Depletion cube, you then take as much sole as possible from DaysOld batch 6 (6 day-old fish). If batch 6 cannot satisfy the requirements for sole, you then turn to DaysOld batch 5, and so on, from oldest to newest until the daily requirement is met.

The plan also requires that you create a rule statement for the Depletion cube that establishes available amounts of fish from the Inventory cube:

```
['Available']=DB('Inventory',!FishType,!Date,!DaysOld,'Quantity
in Stock - Kgs');
```

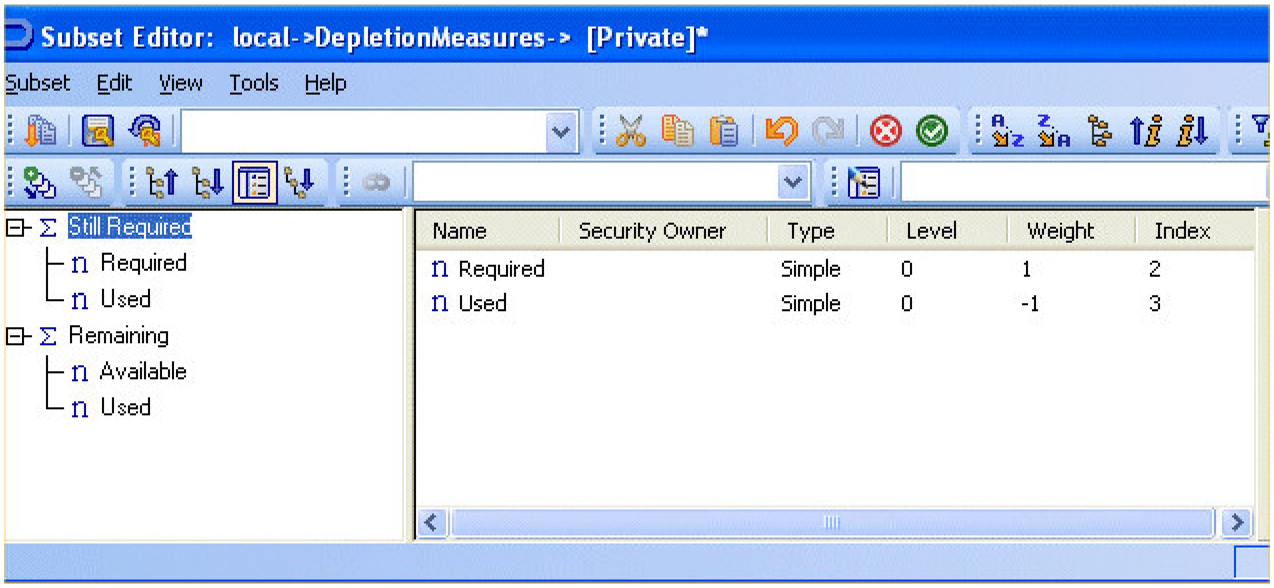
The Fishcakes International model keeps inventory batched by age with a DaysOld dimension, so the statement pulls in values for six separate batches of fish, by age. For example, available Herring on Jan. 1 falls into six batches: 20 kilos - 6 days old, 14 kilos - 5 days old, 10 kilos - 4 days old, 10 kilos - 3 days old, 84 kilos - 2 days old and 0 kilos - 1 day old.

Now you can create a statement that calculates the amount of each batch of fish that should be used to satisfy the required amount:

```
['Used'] = N:IF( ['Available'] >= ['Required'], ['Required'], ['Available']);
```

In English, this statement says "if what's available is greater than or equal to what's required, then take what's required; otherwise take what's available."

The calculation of amounts remaining and amounts still required is a matter of subtraction. This subtraction is accomplished through the use of element weights in consolidations found in the DepletionMeasures dimension. For example, by applying a weight of -1 to the Used element in the Remaining consolidation, the value of the Remaining consolidation is calculated by subtracting the value of Used from the value of Available.



Name	Security Owner	Type	Level	Weight	Index
Required		Simple	0	1	2
Used		Simple	0	-1	3

But you still don't have a Required amount for the DaysOld batches less than 6. The amount required when the model gets to the 5-day old fish is the amount still required after using all available 6-day-old fish; similarly, the amount required of 4-day old fish is the amount still required after using up 5-day-old fish, and so on. The statement to calculate required amounts is:

```
['Required']=N:DB('Depletion',!FishType,!Date,DNEXT('DaysOld',!DaysOld),
'Still Required');
```

The complete rule statements for the Depletion cube now look like this:

```
['6','Required']=DB('FishRequired','Total Fish Cake Types',!FishType,!Date,'Qty
Required - Kgs');
```

```
['Available']=DB('Inventory',!FishType,!Date,!DaysOld,'Quantity
in Stock - Kgs');
```

```
['Used']=N:IF(['Available'] >= ['Required'], ['Required'],
['Available']);
```

```
['Required']=N:DB('Depletion',!FishType,!Date,DNEXT('DaysOld', !DaysOld),'Still
Required');
```

Note how the rule for `['6','Required']` comes before the rule for `['Required']`. This is in keeping with the practice of placing rules statements in order from most-restrictive area definition to least-restrictive area definition.

Follow the steps below to create the rule for the Depletion cube.

Procedure

1. Right-click the Depletion cube in **Server Explorer** and click **Create Rule**.

   The **Rules Editor** opens.
2. Enter the statements in the **Rules Editor**.

When you are done, the **Rules Editor** should appear as follows.

```
['6','Required']=DB('FishRequired','Total Fish Cake Types',!FishType,!Date,'Qty
Required - Kgs');
```

```
['Available']=DB('Inventory',!FishType,!Date,!DaysOld,'Quantity
in
Stock - Kgs');
```

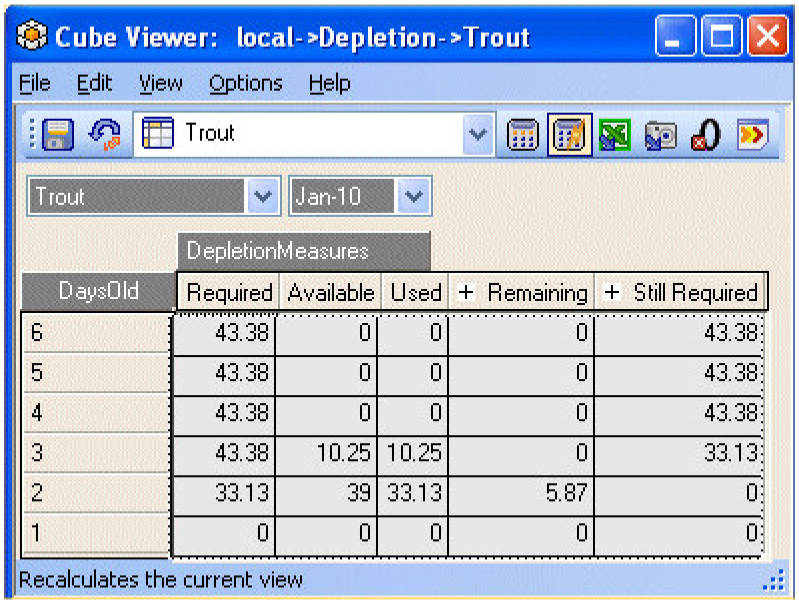
```
['Used']=N:IF(['Available']>= ['Required'],['Required'],['Available']);
```

```
['Required']=N:DB('Depletion',!FishType,!Date,DNEXT('DaysOld', !DaysOld),'Still
Required');
```

3. Click **Save** to compile and save the rule.

   Note that the rule does not include a SKIPCHECK; declaration. You add this declaration later when you develop feeders for the rule.

4. Open the Trout view of the Depletion cube to understand the execution of the rule. All values in the Trout view are derived through the rules you just created.



DaysOld	Required	Available	Used	+ Remaining	+ Still Required
6	43.38	0	0	0	43.38
5	43.38	0	0	0	43.38
4	43.38	0	0	0	43.38
3	43.38	10.25	10.25	0	33.13
2	33.13	39	33.13	5.87	0
1	0	0	0	0	0

Recalculates the current view

TM1 executed the rules you created in the following way:

5. TM1 determined that the amount of trout required on Jan. 10 is 43.38 Kgs. This value is derived by the statement `['6','Required']=DB('FishRequired','Total Fish Cake Types',!FishType,!Date,'Qty Required - Kgs')`, which pulls the value 43.38 from the FishRequired cube. TM1 first attempts satisfy this requirement from the DaysOld batch 6.
6. For the given DaysOld batch, the TM1 server then uses the statement `['Available']=DB('Inventory',!FishType,!Date,!DaysOld,'Quantity in Stock - Kgs')` to determine the amount of trout available.
7. For the given DaysOld batch, the statement `['Used']=N:IF(['Available'] >= ['Required'], ['Required'], ['Available'])` calculates the amount of trout used. When the amount available is greater than or equal to the amount required, only the amount required is used; otherwise, the amount available is used.
8. The value for the Remaining consolidation is defined as Available minus Used in the DepletionMeasures dimension.
9. The value for the Still Required consolidation is defined as Required minus Used in the DepletionMeasures dimension.
10. The TM1 server then uses the statement `['Required']=N:DB('Depletion',!FishType,!Date,DNEXT('DaysOld', !DaysOld), 'Still Required')` to determine the amount of trout required from the next DaysOld batch.

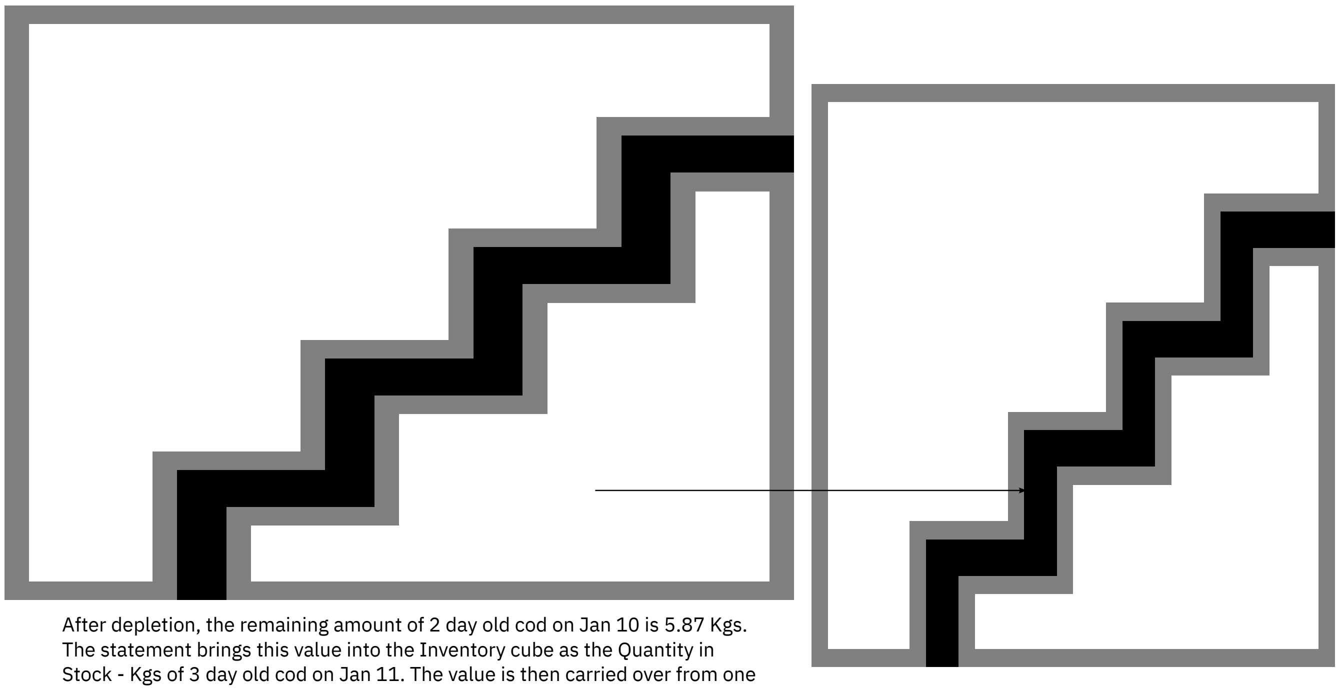
TM1 can now perform all the calculations necessary to satisfy the required amount of trout by depleting inventory from the oldest available batches of fish.

You're almost done with the rules required to complete the depletion model. All that's left is to create a statement that takes any remaining amount for a given DaysOld batch and moves it into inventory for the following day, while also incrementing the DaysOld age of the batch by 1. This statement must reside in the rules for the Inventory cube. The following statement accomplishes the goal.

```
['Quantity in Stock - Kgs']= N: IF(DIMIX('Date', !Date)=1,
0, DB('Depletion',!FishType,DIMNM('Date', DIMIX('Date', !Date)-1),
DIMNM('DaysOld',DIMIX('DaysOld', !DaysOld) -1),'Remaining'));
```

This rules say "if the current date is the first date in the Date dimension, then Quantity in Stock - Kgs is 0; otherwise Quantity in Stock - Kgs is equal to yesterday's Remaining value from the Depletion cube for the immediately previous DaysOld batch.

As the following view of the Inventory cube shows, the above rule statement results in a diagonal pattern of numbers, as quantities are carried over from one day to the next day, but aging one day in the process. You can see how this statement works in the following figure.



After depletion, the remaining amount of 2 day old cod on Jan 10 is 5.87 Kgs. The statement brings this value into the Inventory cube as the Quantity in Stock - Kgs of 3 day old cod on Jan 11. The value is then carried over from one day to the next day, but aging one day in the process, resulting in the diagonal pattern shown in the Inventory cube.

To add this statement to the Inventory cube:

11.Double-click the Inventory cube in **Server Explorer.**

The **Rules Editor** displays.

12.Enter the above statement on a new line in the **Rules Editor**.

When you are done, the rule for the Inventory cube should appear as follows.

```
SKIPCHECK;
```

```
['1','Quantity in Stock - Kgs']=N:DB('Purchase',!FishType,'Total
Markets',!Date,'Quantity Purchased - Kgs');
```

```
['1','Purchase Cost']=N:DB('Purchase',!FishType,'Total Markets',!Date,'Purchase
Cost - USD');
```

```
['1','Average Purchase Price/Kg']=N:['1','Purchase Cost']\['1','Quantity
in Stock - Kgs'];C:0;
```

```
['Quantity in Stock - Kgs']= N: IF(DIMIX('Date', !Date)=
1, 0, DB('Depletion',!FishType,DIMNM('Date', DIMIX('Date', !Date)-1),
DIMNM('DaysOld',
DIMIX('DaysOld',!DaysOld)-1),'Remaining'));
```

13.Click **Save** to compile and save the rule.

Feeding the Depletion Process

After you have the depletion model working, you must add SKIPCHECK and FEEDERS statements to optimize the performance of the model.

First, you must add a SKIPCHECK statement to the beginning of the rule for the Depletion cube.

Procedure

1. Double-click the Depletion rule in **Server Explorer**.
2. Insert the SKIPCHECK; statement on the first line of the Rules Editor.
3. Click **Save**.

   Recall that the rule for the Depletion cube is comprised of the following statements:

```
['6','Required']=DB('FishRequired','Total Fish Cake Types',!FishType,!Date,'Qty
Required - Kgs');
```

```
['Available']=DB('Inventory',!FishType,!Date,!DaysOld,'Quantity
in
Stock - Kgs');
```

```
['Used']=N:IF(['Available'] >= ['Required'], ['Required'], ['Available']);
```

```
['Required']=N:DB('Depletion',!FishType,!Date,DNEXT('DaysOld',!DaysOld),'Still
Required');
```

Feeding the First Statement

The first statement, which brings values into the Depletion cube from the FishRequired cube, demands a feeder in the rule for the FishRequired cube.

```
['Total Fish Cake Types','Qty Required - Kgs']=>DB('Depletion',
```

```
!fishtype,!date,'6','Required');
```

You can easily invert the statement to construct the following feeder:

This feeder says that whenever there is a value for Qty Required - Kgs for Total Fish CakeTypes, feed cells in the Depletion cube identified by 6 and Required.

Follow the steps below to add this statement to the rule for the FishRequired cube.

Procedure

1. Double-click the FishRequired rule in **Server Explorer**.
2. Insert a FEEDERS; declaration on a line below the existing rule statement in the Rules Editor.
3. Add the following statement immediately following the FEEDERS; declaration.

   ```
   ['Total Fish Cake Types','Qty Required - Kgs']=>DB('Depletion',!fishtype,
   !date,'6','Required');
   ```

   The FishRequired rule should now appear as follows:

   ```
   SKIPCHECK;
   ```

   ```
   ['Qty Required - Kgs']=N:DB('Production',!CakeType,!Date,'Quantity
   Produced - Kgs')*DB('Ingredients',!CakeType,!FishType);
   ```

   ```
   FEEDERS;
   ```

   ```
   ['Total Fish Cake Types','Qty Required - Kgs']=>DB('Depletion',
   ```

   ```
   !fishtype,!date,'6','Required');
   ```
4. Click **Save** to save the Fish Required rule.

Feeding the Second Statement

The second statement in the rule for the Depletion cube pulls data from the Inventory cube.

```
['Available']=DB('Inventory',!FishType,!Date,!DaysOld,'Quantity
in Stock - Kgs');
```

The statement requires an accompanying feeder in the rule for the Inventory cube. Again, the feeder can be constructed by inverting the calculation statement to arrive at the following:

```
['Quantity in Stock - Kgs']=>DB('Depletion',!FishType,!Date,!DaysOld,'Available');
```

Follow the steps below to add this statement to the rule for the Inventory cube.

Procedure

1. Double-click the Inventory rule in the Server Explorer.
2. Add the feeder immediately beneath the existing feeders statements in the rule.

The Inventory rule should now appear as follows:

```
SKIPCHECK;
```

```
['1','Quantity in Stock - Kgs']=N:DB('Purchase',!FishType,'Total
Markets',!Date,'Quantity Purchased - Kgs');
```

```
['1','Purchase Cost']=N:DB('Purchase',!FishType,'Total Markets',!Date,'Purchase
Cost - USD');
```

```
['1','Average Purchase Price/Kg']=N:['1','Purchase Cost']\['1','Quantity
in Stock - Kgs'];C:0;
```

```
['Quantity in Stock - Kgs']=N:IF(DIMIX('Date',!Date)=1,0,DB('Depletion',!FishType,
DIMNM('Date',DIMIX('Date;,!Date)-1),DIMNM('DaysOld',DIMIX('DaysOld',
!DaysOld)-1),'Remaining'));
```

```
FEEDERS;
```

```
['Quantity in Stock - Kgs']=>['Average Purchase Price/Kg'];
```

```
['Quantity in Stock - Kgs']=>['Purchase Cost'];
```

```
['Quantity in Stock - Kgs']=>DB('Depletion',!FishType,!Date,!DaysOld,'Available');
```

3. Click **Save** to save the Inventory rule.

Feeding the Remaining Statements

The final two calculation statements in the rule for the Depletion cube do not reference any other cube. Therefore, the feeders for these statements also reside in the Depletion rule.

The statement that calculates Used values references both Required and Available values:

```
['Used']=N:IF(['Available'] >= ['Required'], ['Required'],['Available']);
```

You might be tempted to write the following feeders to accompany this calculation statement:

```
['Required']=>['Used'];
```

```
['Available']=>['Used'];
```

However, for all instances where fish is used, fish is necessarily both required *and* available. Therefore, you only need to feed Used once, with either Required *or* Available. If you arbitrarily choose Available, you arrive at the feeder statement:

```
['Available']=>['Used'];
```

You can now add this feeder statement to the rule for the Depletion cube.

Procedure

1. Double-click the Depletion rule in the **Server Explorer**.
2. Insert a FEEDERS; declaration on a line below the existing rule statements in the **Rules Editor**.
3. Add the feeder immediately following the FEEDERS; declaration.
4. Click **Save**.

Now you must create feeders for the rule statement that calculates the value of Required.

```
['Required']=N:DB('Depletion',!FishType,!Date,DNEXT('DaysOld',!DaysOld),
'Still Required');
```

This statement calculates the Required value for a given DaysOld batch based on the Still Required value of the next-oldest DaysOld batch. And remember, this statement calculates Required values for all DaysOld batches except batch.

5. A separate rule calculates Required values for DaysOld batch. Accordingly, you do not need to feed the area

```
['6', 'Required'].
```

If there were a large number of elements in the DaysOld dimension you might want to create a single, complex feeder statement that incorporates a series of rules functions. However, the DaysOld dimension is exceptionally small, so the easiest way to create and maintain the required feeders is to write a separate feeder statement for each area that needs to be fed:

```
['6', 'Still required']=>['5', 'Required'];
```

```
['5', 'Still required']=>['4', 'Required'];
```

```
['4', 'Still required']=>['3', 'Required'];
```

```
['3', 'Still required']=>['2', 'Required'];
```

```
['2', 'Still required']=>['1', 'Required'];
```

You can now add these feeder statements to the rule for the Depletion cube.

6. Double-click the Depletion rule in the **Server Explorer**.
7. Add the feeder statements immediately following the existing feeders in the rule.
8. Click **Save**.

Feeding the Calculation for Quantity in Stock - Kgs in the Inventory Cube

The rule for the Inventory cube includes a calculation statement that calculates Quantity in Stock - Kgs based on Remaining values in the Depletion cube.

```
['Quantity in Stock - Kgs']= N: IF(DIMIX('Date', !Date)=
1, 0,DB('Depletion',!FishType,DIMNM('Date', DIMIX('Date', !Date)-1),
DIMNM('DaysOld', DIMIX( 'DaysOld', !DaysOld) -1),'Remaining'));
```

You could use the following feeder to feed Quantity in Stock - Kgs in the Inventory cube:

```
['Remaining']=>DB('Inventory', !FishType, DNEXT('Date',!Date), DNEXT('DaysOld',
!DaysOld),'Quantity in Stock - Kgs');
```

The DNEXT function cycles through an entire dimension, feeding both leaf and consolidated elements, resulting in massive overfeeding.

TM1 does include a rules function, DTYPE, that lets you determine the type of an element. The function, described in the *TM1 Reference* guide returns a single-character string: 'N' for leaf (or numeric) elements, 'C' for consolidated elements, and 'S' for string elements. You can use this function in a feeder statement to determine the type of the current element and to feed only 'N' type elements, as in the following feeder.

```
['Remaining']=>DB('Inventory', !FishType, IF( DTYPE('DATE', DNEXT('Date',!Date))
@= 'N', DNEXT('Date',!Date),'Dec-31' ), IF(DTYPE( 'DaysOld', DNEXT('DaysOld',!DaysOld))
@= 'N',DNEXT('DaysOld', !DaysOld),'1' ), 'Quantity in Stock - Kgs');
```

Note: The @ symbol is the string comparison operator.

The first IF statement says "if the type of the next element in the Date dimension is N, use the next element; otherwise, use Dec-31.

The second IF statement says "if the type of the next element in the DaysOld dimension is N, use the next element; otherwise use 1.

The result of these IF statements is that the feeder never feeds a consolidated element, but instead feeds either Dec-31 (Date element) or 1 (DaysOld element) when a consolidated element is encountered. This yields some minor overfeeding, but nothing of the magnitude that would result from feeding all the consolidated elements in the Date dimension.

You can now add the above statement to the feeders for the Depletion rule.

Procedure

1. Double-click the Depletion rule in the **Server Explorer**.
2. Add the feeder immediately beneath the existing feeders statement in the rule.
3. Click **Save** and compile the rule.

Final Rule for the Depletion Cube

You can now finish creating calculation statements and feeder statements for the Depletion rule.

After you finish creating calculation statements and feeder statements for the Depletion rule, the rule should appear as follows:

```
['6','Required']=DB('FishRequired','Total Fish Cake Types',!FishType,!Date,'Qty
Required - Kgs');
```

```
['Available']=DB('Inventory',!FishType,!Date,!DaysOld,'Quantity
in Stock - Kgs');
```

```
['Used']=N:IF(['Available']>=['Required'],['Available']);
```

```
['Required']=N:DB('Depletion',!FishType,!Date,DNEXT('DaysOld',!DaysOld),'Still
Required');
```

```
FEEDERS;
```

```
['Available']=>['Used'];
```

```
['6','Still Required']=>['5','Required'];
```

```
['5','Still Required']=>['4','Required'];
```

```
['4','Still Required']=>['3','Required'];
```

```
['3','Still Required']=>['2','Required'];
```

```
['2','Still Required']=>['1','Required'];
```

```
['Remaining']=>DB('Inventory',!FishType,IF(DTYPE('DATE',DNEXT('Date,!Date))
```

```
@='N',DNEXT('Date',!Date),'Dec-31'),IF(DTYPE('DaysOld',DNEXT('DaysOld;,
```

```
!DaysOld))in Stock - Kgs;)'
```

```
@='N',DNEXT('DaysOld',!DaysOld),'1'),'Quantity
```

Final Rule for the Inventory Cube

You can now finish creating calculation statements and feeder statements for the Depletion rule.

After you finish creating calculation statements and feeder statements for the Inventory rule, the rule should appear as follows:

```
SKIPCHECK;
```

```
['1','Quantity in Stock - Kgs']=N:DB('Purchase',!FishType,'Total Markets',!
Date,'Quantity
Purchased - Kgs');
```

```
['1','Purchase Cost - Kgs']=N:DB('Purchase',!FishType,'Total Markets',!Date,'Purchase
Cost - USD');
```

```
['1','Average Purchase Price/Kg']=N:['1','Purchase Cost']\['1','Quantity
in Stock - Kgs'];C:0;
```

```
['Quantity in Stock - Kgs']=N:IF(DIMIX('Date',!Date)=1,0,DB('Depletion;,
```

```
!FishType,DIMNM('Date',DIMIX('Date',!Date)-DIMNM('DaysOld',
```

```
DIMIX('DaysOld',!DaysOld)-1),'Remaining'));
```

```
FEEDERS;
```

```
['Quantity in Stock - Kgs']=>['Average Purchase Price/Kg'];
```

```
['Quantity in Stock - Kgs']=>['Purchase Cost'];
```

```
['Quantity in Stock - Kgs']=>DB('Depletion',!FishType,!Date,!DaysOld,'Available');
```

Chapter 10. Calculating Total Product Costs

One of the primary goals of the Fishcakes International model is to figure out what it costs, on a daily basis, to produce each type of fishcake. Between the Production, FishRequired and Inventory cubes, you have all the information necessary to calculate this daily cost in the Production cube.

- In the rule for the Inventory cube, you create statements that calculate the average price per Kg of each fish type, as well as the daily purchase cost for each fish type.
- In the rule for the FishRequired cube, you create a rule statement that calculates the cost of each fish required for a given fishcake on a daily basis, based on values in the Inventory cube.
- Finally, you create rule statements for the Production cube that calculate the daily production cost for each fishcake type as well as the cost per Kg of each fishcake type.

Calculating Daily Fish Costs in the Inventory Cube

Before you can calculate the daily production cost of fishcake types, you must first determine the daily cost of each individual fish type, taking account of the prices paid for each DaysOld batch and the amount of fish used from each batch.

Recall that in Chapter 4, "Using DB Functions to Move Data Between Cubes," on page 45 you created the following statements in the Inventory rule to calculate the values of Quantity in Stock - Kgs, Purchase Cost, and Average Purchase Price/Kg for DaysOld batch 1:

```
['1','Quantity in Stock - Kgs']= N:DB('Purchase',!FishType,'Total Markets',!
Date,'Quantity
Purchased - Kgs');['1','Purchase Cost'] = N:DB('Purchase',!FishType,'Total
Markets',!Date,'Purchase Cost - USD');['1','Average Purchase Price/Kg']=
N: ['Purchase Cost'] \ ['Quantity in Stock - Kgs'];
```

As each day passes, each DaysOld batch of fish ages by one day, and any unused fish is moved into the next DaysOld batch. To calculate the value of the unused fish each day, you need to multiply the quantity of unused fish by the Average Purchase Price/Kg for the relevant DaysOld batch.

At this point, though, you only have a statement in the rule for the Inventory cube to calculate the Average Purchase Price/Kg for DaysOld batch 1. You need to create a statement that causes the Average Purchase Price/Kg value to "travel" with the batch as it ages. For example, the value of Average Purchase Price/Kg for DaysOld batch 1 on Apr. 30 should "travel" to become the value of Average Purchase Price/Kg for DaysOld batch 2 on May 1. Adding the following statement to the rule for the Inventory cube yields the desired result.

```
['Average Purchase Price/Kg'] = IF(DIMIX('Date',!Date)=1,0, DB('Inventory',!FishType,
DIMNM('Date', DIMIX('Date', !Date)-1), DIMNM('DaysOld',DIMIX('DaysOld',!DaysOld)
-1), 'Average Purchase Price/Kg'));
```

An IF statement is required because on the first day of the year there is no fish that is more than one day old and because the formula references the value of Average Purchase Price/Kg for fish on the previous day, which is undefined for the first day of the year.

Recall that the rule for the Inventory cube already includes the statement ['1','Average Purchase Price/Kg'] = N: ['Purchase Cost'] \ ['Quantity in Stock - Kgs'], which calculates the value of Average Purchase Price/Kg for fish in DaysOld batch 1, even on the first day of the year.

The following steps illustrate how to add the statement to calculate Average Purchase Price/Kg to the Inventory rule.

Procedure

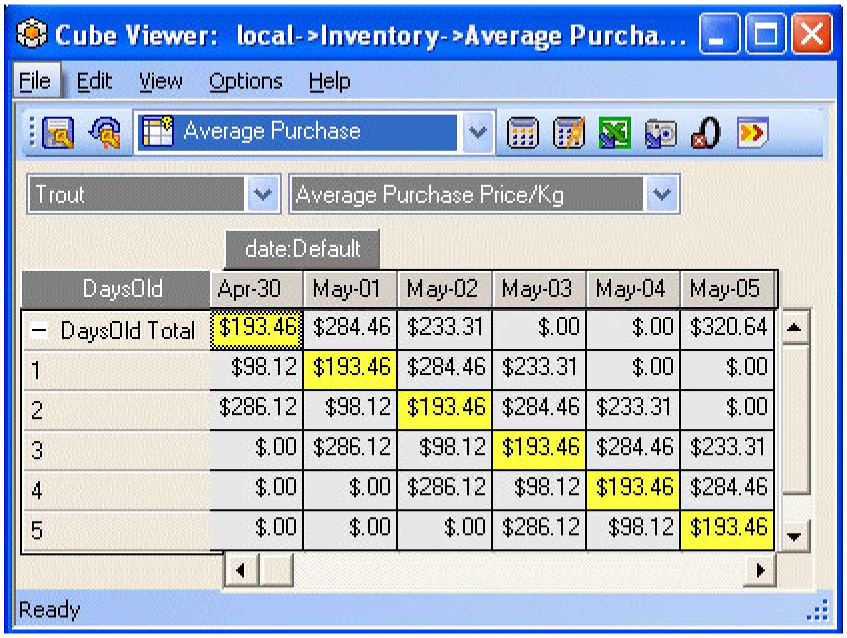
1. Double-click the Inventory rule in **Server Explorer**.

2. Add the statement

```
['Average Purchase Price/Kg'] = IF(DIMIX('Date',!Date)=1,0, DB('Inventory',!
FishType,
DIMNM('Date', DIMIX('Date', !Date)-1), DIMNM('DaysOld',DIMIX('DaysOld',!DaysOld)
-1), 'Average Purchase Price/Kg'));
```

   immediately following the last calculation statement in the rule.
3. Click **Save** to compile and save the rule.

   Values for Average Purchase Price/Kg now travel with each batch of fish as it ages:


Cube Viewer: local->Inventory->Average Purcha...
File
Edit
View
Options
Help
Average Purchase
Trout
Average Purchase Price/Kg
date:Default
Ready

DaysOld	Apr-30	May-01	May-02	May-03	May-04	May-05
− DaysOld Total	$193.46	$284.46	$233.31	$.00	$.00	$320.64
1	$98.12	$193.46	$284.46	$233.31	$.00	$.00
2	$286.12	$98.12	$193.46	$284.46	$233.31	$.00
3	$.00	$286.12	$98.12	$193.46	$284.46	$233.31
4	$.00	$.00	$286.12	$98.12	$193.46	$284.46
5	$.00	$.00	$.00	$286.12	$98.12	$193.46

   As this view shows, the value of Average Purchase Price/Kg for trout in DaysOld batch 1 on April 30 becomes the value for DaysOld batch 2 on May 1. This value then travels to DaysOld batch 3 on May 2, then DaysOld batch 4 on May 3, and so on.

Calculating Daily Purchase Costs

Having correctly defined how Average Purchase Price/Kg varies with time, you can now add the statement to the Inventory rule to calculate the purchase cost of inventory.

```
['Purchase Cost']=N: ['Average Purchase Price/Kg'] *['Quantity
in Stock - Kgs'];
```

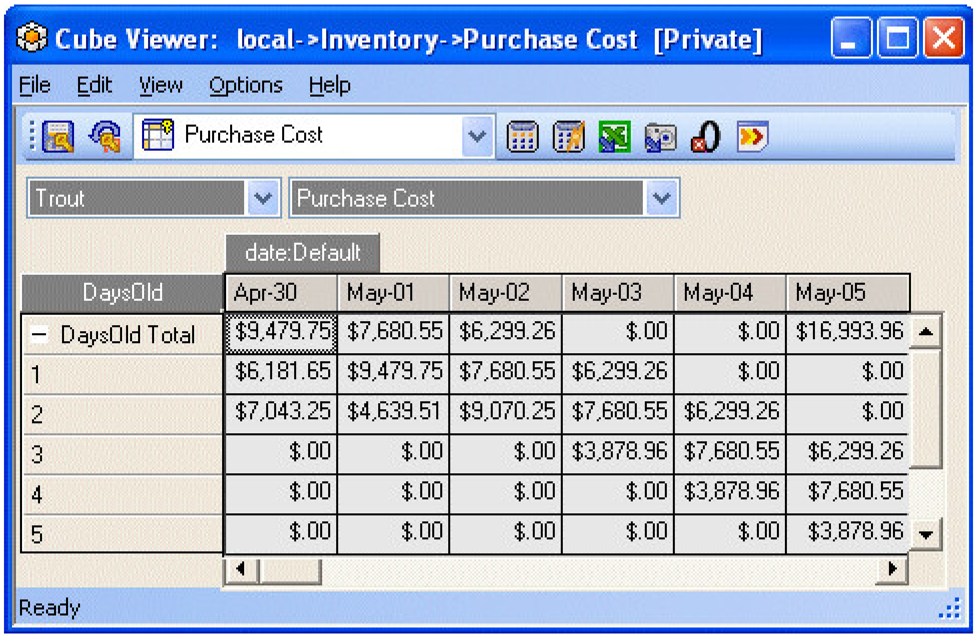
Keep in mind that the Inventory rule already includes a statement that retrieves the value of Purchase Cost for Daysold batch 1 from the Purchase cube. Accordingly, the statement applies only to DaysOld batches 2 through 6.

The following steps illustrate how to add the statement to calculate Purchase Cost to the Inventory rule.

Procedure

1. Double-click the Inventory rule in the **Server Explorer**.
2. Add the statement immediately following the last calculation statement in the rule.
3. Click **Save** to compile and save the rule.

   As the following view shows, the statement calculates Purchase Cost values.


Cube Viewer: local->Inventory->Purchase Cost [Private]
File Edit View Options Help
Purchase Cost
Trout
Purchase Cost
date:Default
Ready

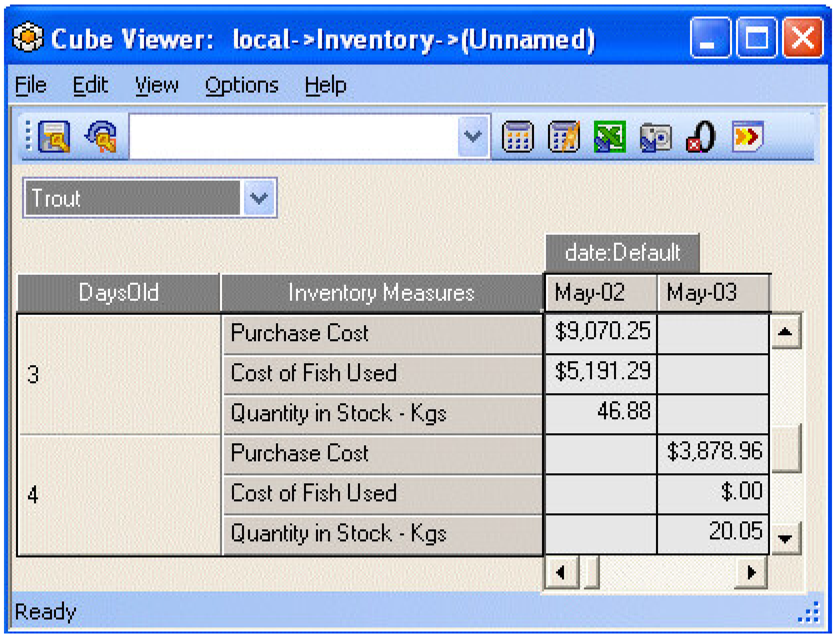
DaysOld	Apr-30	May-01	May-02	May-03	May-04	May-05
− DaysOld Total	$9,479.75	$7,680.55	$6,299.26	$.00	$.00	$16,993.96
1	$6,181.65	$9,479.75	$7,680.55	$6,299.26	$.00	$.00
2	$7,043.25	$4,639.51	$9,070.25	$7,680.55	$6,299.26	$.00
3	$.00	$.00	$.00	$3,878.96	$7,680.55	$6,299.26
4	$.00	$.00	$.00	$.00	$3,878.96	$7,680.55
5	$.00	$.00	$.00	$.00	$.00	$3,878.96

Calculating the Cost of Fish Used

The rule for the Inventory cube now contains statements to calculate the value of Purchase Cost for all dates and all DaysOld batches.

It's now a simple matter of subtraction to determine the value for Cost of Fish Used. Specifically, the Cost of Fish Used for any given date and DaysOld batch is determined by subtracting the Purchase Cost for the following date and DaysOld batch from the current date and DaysOld batch.

This sounds slightly confusing, so consider the following view of the Inventory cube. Remember that a batch of fish ages as days progress, so DaysOld batch 3 on May 2 becomes DaysOld batch 4 on May 3.


Cube Viewer: local->Inventory->(Unnamed)
File Edit View Options Help
Trout
date:Default
Ready

DaysOld	Inventory Measures	May-02	May-03
3	Purchase Cost	$9,070.25	
	Cost of Fish Used	$5,191.29	
	Quantity in Stock - Kgs	46.88	
4	Purchase Cost		$3,878.96
	Cost of Fish Used		$.00
	Quantity in Stock - Kgs		20.05

Here you see that the Purchase Cost of trout in DaysOld batch 3 on May 2 is $9,070.25. The quantity in stock is 46.88 Kgs. (Note that Purchase Cost and Quantity in Stock - Kgs are measures of fish at the beginning of the day.)

The Purchase Cost of trout in DaysOld batch 4 on May 3 is $3,878.96, and the quantity in stock is 20.05 Kgs. Because both Purchase Cost and Quantity in Stock are reduced at the start of the day on May 3, you can surmise that some trout was used to make fishcakes on May 2.

You can determine the cost of trout used on May 2 by subtracting the Purchase Cost value of trout in DaysOld batch 4 on May 3 from the Purchase Cost value of trout in DaysOld batch 3 on May 2.

```
$9,070.25 - $3,878.96 = $5,191.29
```

If you refer back to the preceding view, you see that $5,191.29is indeed the Cost of Fish Used for DaysOld batch 3 on May 2. The rule statement required to calculate this value is

```
['Cost of Fish Used']=N:DB('Inventory',!FishType,!Date,!DaysOld,'Purchase
Cost')- DB('Inventory',!FishType,DNEXT('Date', !Date),DNEXT('DaysOld',
!DaysOld),'Purchase Cost');
```

Calculating the Cost of Fish Used for DaysOld batch 6 requires a different rule statement, because there is no batch after 6. Once a batch of fish is 6 days old, it is (thankfully) removed from inventory and the entire purchase cost of the batch is charged to the cost of fish used. The following statement accomplishes this calculation.

```
['6','Cost of Fish Used']=['Purchase Cost'];
```

Recall that rules statements should always be ordered most-restrictive to least-restrictive according to area definition. In this case, the area ['6','Cost of Fish Used'] is more restrictive than the area ['Cost of Fish Used'], so the statements should appear in the following order:

```
['6','Cost of Fish Used']=['Purchase Cost'];
```

```
['Cost of Fish Used']=N:DB('Inventory',!FishType,!Date,!DaysOld,'Purchase
Cost')- DB('Inventory',!FishType,DNEXT('Date', !Date), DNEXT('DaysOld',
!DaysOld),'Purchase Cost');
```

The following steps illustrate how to add these statements to the rule for the Inventory cube.

Procedure

1. Double-click the Inventory rule in the **Server Explorer**.
2. Add the statements immediately beneath the last calculation statement in the rule.

   The statements must be in the order they appear in the example.
3. Click **Save** to compile and save the rule.

Required Feeders

The rules statements you've just created for the Inventory cube require just one new feeder statement, feeding Average Purchase Price/Kg and Cost of Fish Used.

```
['Purchase Cost']=>['Average Purchase Price/Kg'],['Cost
of Fish Used'];
```

Note that this single feeder statement feeds two elements, with each element separated by a comma.

You can now add this feeder statement to the rule for the Inventory cube.

Procedure

1. Double-click the Inventory rule in **Server Explorer**.
2. Add the feeder statements immediately beneath the last feeder statement in the rule.

The complete rule for the Inventory cube should now appear as follows:

```
SKIPCHECK;
```

```
['1','Quantity in Stock - Kgs']=N:DB('Purchase',!FishType,'Total
Markets',!Date,'Quantity Purchased - Kgs');
```

```
['1','Purchase Cost']=N:DB('Purchase',!FishType,'Total Markets',!Date,'Purchase
Cost - USD');
```

```
['1','Average Purchase Price/Kg;]=N:['1','Purchase Cost']\['1','Quantity
in Stock - Kgs'];C:0;
```

```
['Quantity in Stock - Kgs']=N:IF(DIMIX('Date',!Date)=1,0, DB('Depletion',
!FishType,DIMNM('Date',DIMIX('Date',!Date)-1),
```

```
    DIMNM('DaysOld',DIMIX('DaysOld',!DaysOld) -1),'Remaining'));
```

```
['Average Purchase Price/Kg']=IF(DIMIX('Date',!Date)=1,0, DB('Inventory',!FishType,
```

```
    DIMNM('Date',DIMIX('Date',!Date)-1),
```

```
    DIMNM('DaysOld',DIMIX('DaysOld',!DaysOld)-1),
```

```
    'Average Purchase Price/Kg'));
```

```
['Purchase Cost']=N: ['Average Purchase Price/Kg']*['Quantity
in
Stock - Kgs'];
```

```
['6','Cost of Fish Used']=['Purchase Cost'];
```

```
['Cost of Fish Used']=N:DB('Inventory',!FishType,!Date,!DaysOld,'Purchase Cost')
```

```
    -DB('Inventory',!FishType,DNEXT('Date',!Date),
```

```
    DNEXT('DaysOld',!DaysOld),'Purchase Cost')'
```

```
FEEDERS;
```

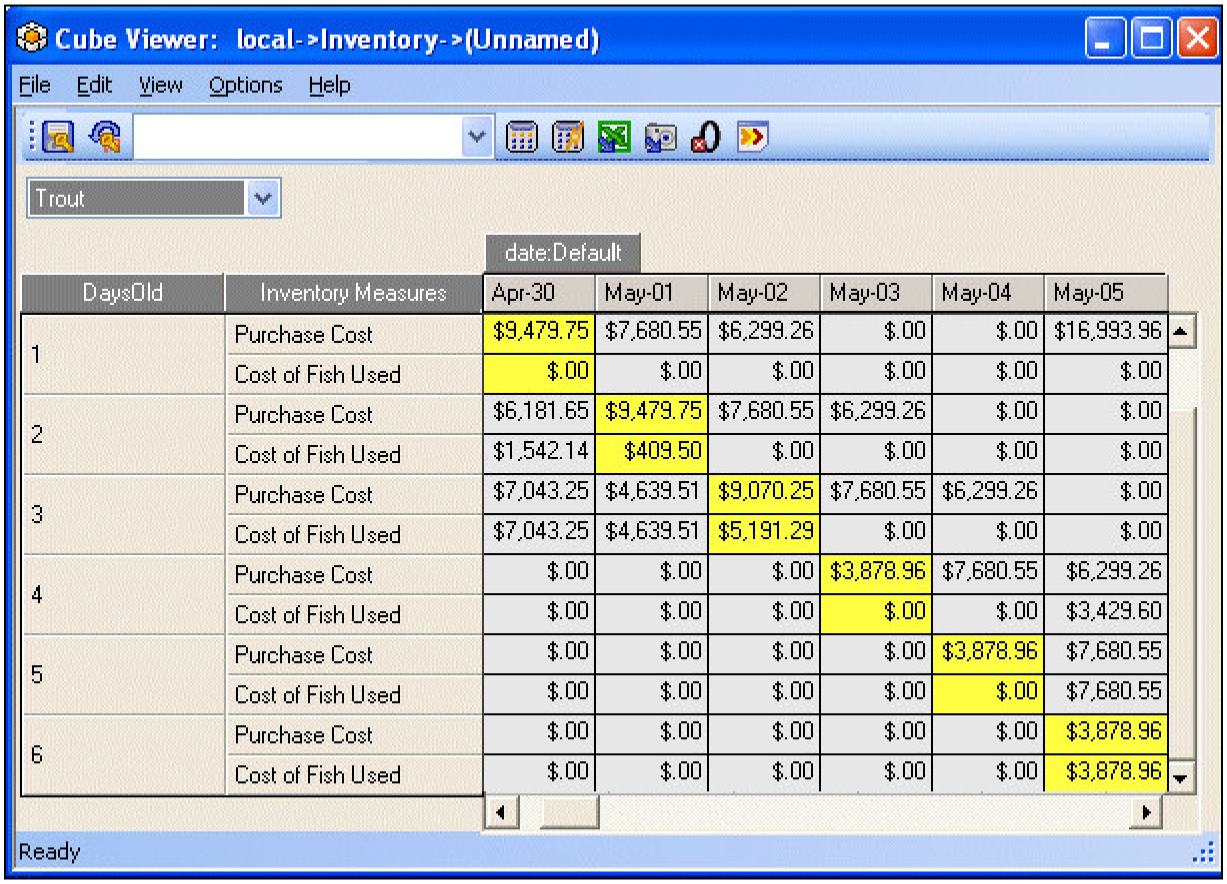
```
['Quantity in Stock - Kgs']=>['Average Purchase Price/Kg'];
```

```
['Quantity in Stock - Kgs']=>['Purchase Cost'];
```

```
['Quantity in Stock - Kgs']=>['DB('Depletion',!FishType,!Date,!DaysOld,'Available');
```

```
['Purchase Cost']=>['Average Purchase Price/Kg'],['Cost
of Fish Used'];
```

3. Click **Save** to compile and save the rule.
4. Open the Cost of trout used view of the Inventory cube to see the complete effect of the rules that calculate daily costs.



Cube Viewer: local->Inventory->(Unnamed)

Trout

date:Default

DaysOld	Inventory Measures	Apr-30	May-01	May-02	May-03	May-04	May-05
1	Purchase Cost	$9,479.75	$7,680.55	$6,299.26	$.00	$.00	$16,993.96
	Cost of Fish Used	$.00	$.00	$.00	$.00	$.00	$.00
2	Purchase Cost	$6,181.65	$9,479.75	$7,680.55	$6,299.26	$.00	$.00
	Cost of Fish Used	$1,542.14	$409.50	$.00	$.00	$.00	$.00
3	Purchase Cost	$7,043.25	$4,639.51	$9,070.25	$7,680.55	$6,299.26	$.00
	Cost of Fish Used	$7,043.25	$4,639.51	$5,191.29	$.00	$.00	$.00
4	Purchase Cost	$.00	$.00	$.00	$3,878.96	$7,680.55	$6,299.26
	Cost of Fish Used	$.00	$.00	$.00	$.00	$.00	$3,429.60
5	Purchase Cost	$.00	$.00	$.00	$.00	$3,878.96	$7,680.55
	Cost of Fish Used	$.00	$.00	$.00	$.00	$.00	$7,680.55
6	Purchase Cost	$.00	$.00	$.00	$.00	$.00	$3,878.96
	Cost of Fish Used	$.00	$.00	$.00	$.00	$.00	$3,878.96

Ready

- The Purchase Cost of trout in DaysOld batch 1 on April 30 is calculated by the statement

```
['1','Purchase
Cost']= N:DB('Purchase',!FishType,'Total Markets',!Date,'Purchase
Cost - USD').
```

- The Purchase Cost of trout in DaysOld batch 2 on May 1 is calculated by the statement

```
['Purchase Cost']=N:['Average Purchase
Price/Kg'] *['Quantity in Stock - Kgs'].
```

  All subsequent Purchase Cost values highlighted in the view are calculated by this statement.

  The Cost of Fish Used for this batch and day is calculated by the statement

```
['Cost of Fish Used']=N:DB('Inventory',!FishType,!Date,!DaysOld,'
Purchase Cost')-DB('Inventory',!FishType,DNEXT('Date', !Date),
DNEXT('DaysOld',!DaysOld),'Purchase Cost').
```

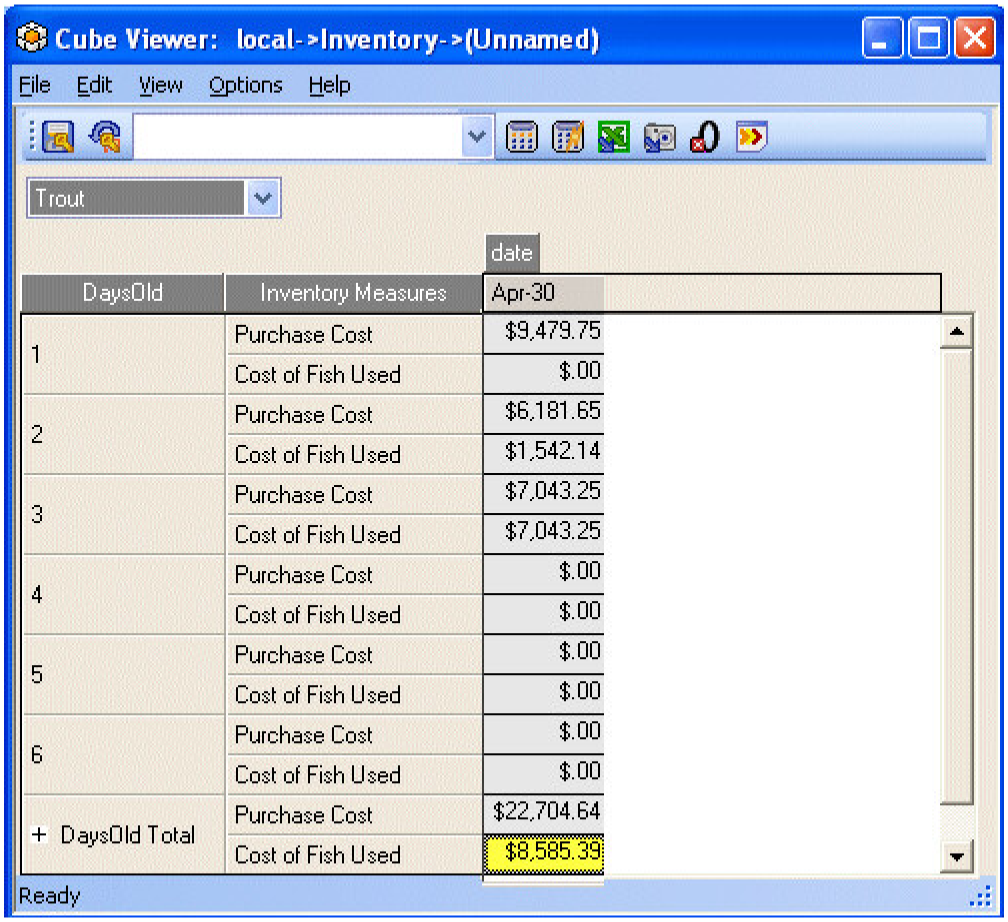
  This statement subtracts the value of Purchase Cost for DaysOld batch 3 on May 2 ($9,070.25) from the value for DaysOld batch 2 on May 1 ($9,479.75), returning $409.50.
- The Cost of Fish Used for DaysOld batch 3 on May 2 is calculated in the same manner. In this case, the statement subtracts the value of Purchase Cost for DaysOld batch 4 on May 3 ($3,878.96) from the value for DaysOld batch 3 on May 2 ($9,070.25), returning $5,191.29.
- Purchase Cost and Cost of Fish Used remain unchanged for batches 4 and 5 on May 3 and May 4, respectively.
- At the end of the day on May 6, any unused trout in DaysOld batch 6 is discarded and the entire purchase cost of the fish in the batch is charged to the cost of fish used. The statement ['6','Cost of Fish Used']=['Purchase Cost'] derives the Cost of Fish Used for batch 6 from the current Purchase Cost of the batch.

Allocating Costs in the FishRequired Cube

Now that you have calculated daily costs for each fish type, you can allocate these costs to the different types of fishcakes produced by Fishcakes International.

Costs for a given fish type need to be allocated to each type of fishcake that utilizes the fish. The allocation of costs is dependent upon the number of kilos of the fish used in the fishcakes on a given day.

For example, in the Inventory cube you can see that the total cost of trout used on April 30 is $8,585.39. This total encompasses all DaysOld batches.


Cube Viewer: local->Inventory->(Unnamed)
File Edit View Options Help
Trout
date
Ready

DaysOld	Inventory Measures	Apr-30
1	Purchase Cost	$9,479.75
	Cost of Fish Used	$.00
2	Purchase Cost	$6,181.65
	Cost of Fish Used	$1,542.14
3	Purchase Cost	$7,043.25
	Cost of Fish Used	$7,043.25
4	Purchase Cost	$.00
	Cost of Fish Used	$.00
5	Purchase Cost	$.00
	Cost of Fish Used	$.00
6	Purchase Cost	$.00
	Cost of Fish Used	$.00
+ DaysOld Total	Purchase Cost	$22,704.64
	Cost of Fish Used	$8,585.39

This total cost needs to be broken down and allocated to individual fishcake types depending on the quantity of trout used in each fishcake on April 30. You can accomplish this with a statement in the rule for the FishRequired cube:

```
['Cost']=DB('Inventory',!fishtype,!date,'DaysOld Total','Cost
of Fish Used') * ['Qty Required - Kgs'] \ ['Total Fish Cake Types','Qty
Required - Kgs'];
```

This statement says that for any given fish type on any given date, the allocated cost of that fish for any given fishcake type is calculated by multiplying the total cost of fish used in the Inventory cube by the quantity of the fish required in the FishRequired cube. Then, divide that product by the total quantity of the fish required by *all* fishcake types. A bit later in this section you will see a full illustration of how this statement works.

You can now add the statement to the rules for the FishRequired cube.

Procedure

1. Double-click the FishRequired rule in **Server Explorer**.
2. Add the statement immediately beneath the last calculation statement in the rule.

The rule for the FishRequired cube should now appear as follows:

```
SKIPCHECK;
```

```
['Qty Required - Kgs']=N:DB('Production',!CakeType,!Date,'Quantity
Produced - Kgs')*DB('Ingredients',!CakeType,!FishType);
```

```
['Cost']=DB('Inventory',!fishtype,!date,'DaysOld Total','Cost
of
Fish Used')*
```

```
    ['Qty Required - Kgs']\['Total Fish Cake Types','Qty
Required - Kgs'];
```

```
FEEDERS;
```

```
['Total fish Cake Types','Qty Required - Kgs']=>DB('Depletion',!fishtype,
!date,'6','Required');
```

3. Click **Save** to compile and save the rule.

Required Feeder

The preceding statement calculates the value of Cost in the Fish Required cube based on the value of Cost of Fish Used in the Inventory cube. Accordingly, the following feeder belongs in the Inventory cube.

```
['Cost of Fish Used']=>DB('FishRequired','Total Fish
Cake Types',!FishType,!Date,'Cost');
```

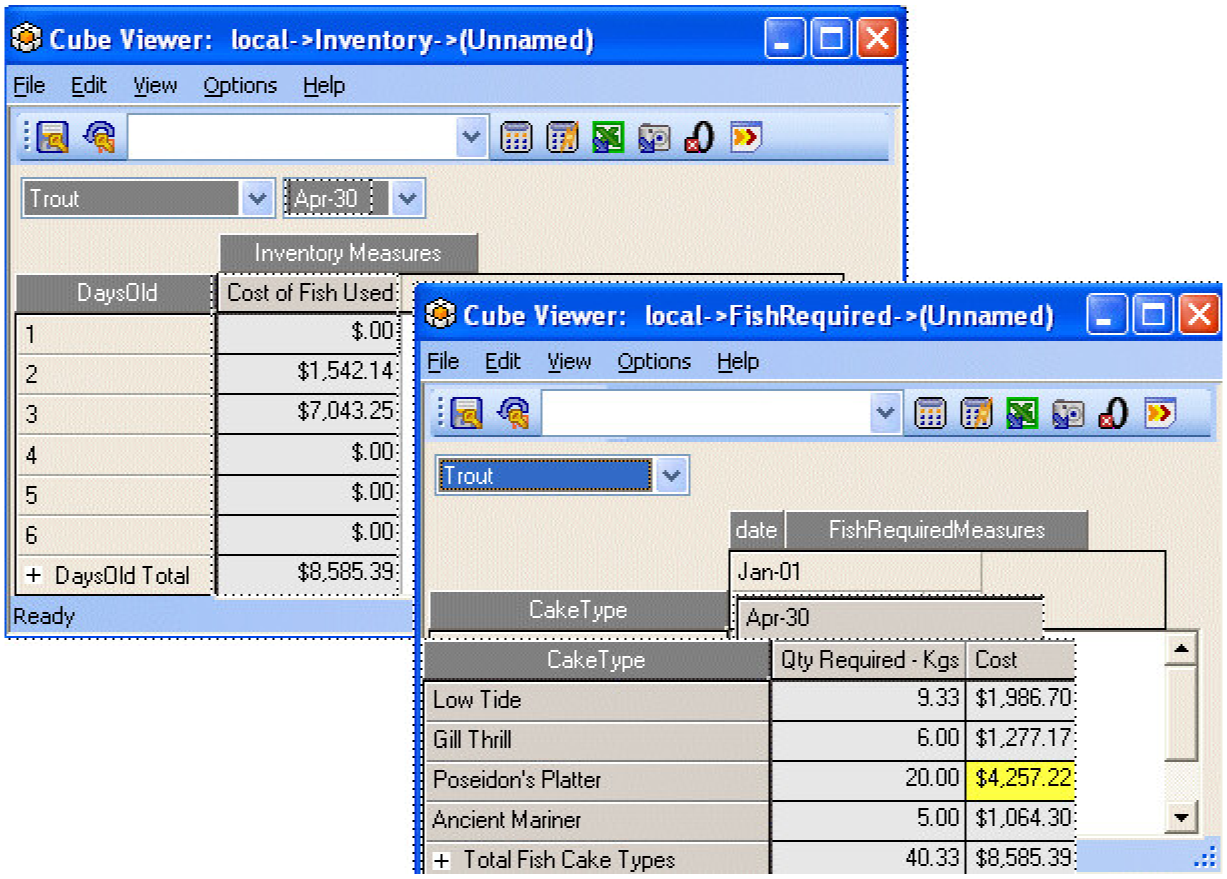
You can now add this feeder statement to the rules for the Inventory cube.

Procedure

1. Double-click the Inventory rule in **Server Explorer**.
2. Add the statement immediately beneath the last feeder statement in the rule.
3. Click **Save** to compile and save the rule.

Viewing the Results

You can see the effect of the new calculation statement by opening the Cost of trout used - Apr. 30 view of the Inventory cube and the Trout required - Apr 30 view of the FishRequired cube.


Cube Viewer: local->Inventory->(Unnamed)

File Edit View Options Help

Trout | Apr-30

Inventory Measures

DaysOld	Cost of Fish Used
1	$.00
2	$1,542.14
3	$7,043.25
4	$.00
5	$.00
6	$.00
+ DaysOld Total	$8,585.39

Ready

Cube Viewer: local->FishRequired->(Unnamed)

File Edit View Options Help

Trout

date | FishRequiredMeasures

Jan-01

CakeType | Apr-30

CakeType	Qty Required - Kgs	Cost
Low Tide	9.33	$1,986.70
Gill Thrill	6.00	$1,277.17
Poseidon's Platter	20.00	$4,257.22
Ancient Mariner	5.00	$1,064.30
+ Total Fish Cake Types	40.33	$8,585.39

- Total cost of trout used on Apr. 30 -- $8,585.39 (DaysOldTotal from the Cost of trout used - Apr 30 view)
- Quantity of trout required for the Poseidon's Platter fishcake on Apr. 30 -- 20 Kgs
- Quantity of trout required for all fishcake types on Apr. 30 -- 40.33 Kgs

TM1 applies these values to the new calculation statement to determine the allocated cost of trout used in Poseidon's Platter fishcakes on Apr. 30:

```
['Cost'] = $8,585.39 x 20 / 40.33
```

This yields an allocated cost $4,257.22 for trout used in Poseidon's Platter fishcakes on Apr. 30, as shown in the highlighted cell of the Trout required - Apr 30 view.

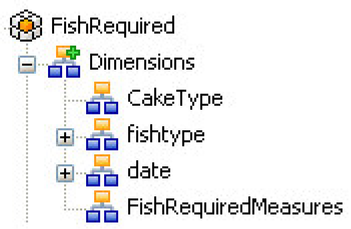
If you take the time to manually perform the calculation above you'll find that $8,585.39 x 20 / 40.33 = $4,257.57. So why does the Trout required - Apr 30 view report a value of $4,257.22? Because the views above use formatting that only shows two decimal places for values, while the complete values use up to four decimal places.

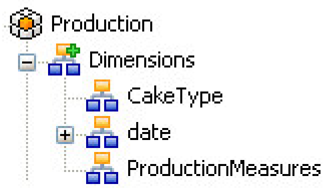
Specifically, the complete value of total cost of trout used on Apr. 30 (from the Cost of trout used - Apr 30 view) is $8,585.3875, while the complete value for the quantity of trout required for all fishcake types on Apr. 30 is 40.3333. If you plug these complete values into the calculation, you'll find that $8,585.3875 x 20 / 40.3333 = $4,257.2204. Applying two-decimal formatting yields $4,257.22, the value shown in the view above. Regardless of the formatting applied to a view or a dimension element, TM1 always uses the full underlying value when calculating rules.

Moving Costs to the Produce Cube

To complete the Fishcakes International model, you must now move production costs for each fishcake from the FishRequired cube to the Production cube.

Unlike the FishRequired cube, the Production cube does not include the FishType dimension; it only keeps information about fishcakes, as defined in the CakeType dimension.


FishRequired
Dimensions
CakeType
fishtype
date
FishRequiredMeasures


Production
Dimensions
CakeType
date
ProductionMeasures

Even though the cubes are of differing dimensionality, you can create a rule statement for the Production cube that calculates production costs based on the value of Total Fish Types in the FishRequired cube.

```
['Production Cost']=DB('FishRequired',!CakeType,'Total
Fish Types',!date,'Cost');
```

This statements says that for a given fishcake type on a given date, the value of Production Cost in the Production cube is equal to the value of Cost for Total Fish Types in the FishRequired cube.

In the Production cube, you also want to calculate the cost per kilogram of fishcakes produced. This is accomplished with a rule statement that divides the value of Production Cost by the value of Quantity Produced - Kgs'.

```
['Production Cost/Kg']=['Production Cost']\['Quantity
Produced - Kgs'];
```

You can now add both statements to the rule for the Production cube.

Procedure

1. Double-click the Production rule in **Server Explorer**.
2. Insert a SKIPCHECK; statement at the top of the rule.
3. Add the rules statements immediately beneath the SKIPCHECK; statement.
4. Click **Save** to save and compile the rule.

Required Feeders

The new calculation statements in the Production cube require two new feeder statements, one in the rule for the FishRequired cube and on in the rule for the Production cube.

Adding the Feeder Statement to the Rule for the FishRequired Cube

You must add a feeder in the rule for the FishRequired cube.

```
['Production Cost']=DB('FishRequired',!CakeType,'Total
Fish Types',!date,'Cost');
```

The calculation statement requires the following feeder in the rule for the FishRequired cube.

```
['Total Fish Types','Cost']=>DB('Production',!CakeType,!date,'Production Cost');
```

Procedure

1. Double-click the FishRequired rule in **Server Explorer**.
2. Add the feeder statement immediately beneath the existing feeder statement in the rule.

The complete rule for the FishRequired cube should now look like this:

```
SKIPCHECK;
```

```
['Qty Required - Kgs']=N:DB('Production;,!CakeType,!Date,'Quantity
Produced - Kgs') * DB('Ingredients',!CakeType,!FishType);
```

```
['Cost']=DB('Inventory',!fishtype,!date,'DaysOld total','Cost
of
Fish Used')*
```

```
    ['Qty Required - Kgs']\['Total Fish Cake Types','Qty
Required - Kgs'];
```

```
FEEDERS;
```

```
['Total Fish Cake Types','Qty Required - Kgs']=>DB('Depletion',
```

```
!fishtype,!date,'6','Required');
```

```
[Total Fish Types','Cost']=>DB('Production',!CakeType,!date,
```

```
'Production Cost');
```

3. Click **Save** to compile and save the rule.

Adding the Feeder Statement to the Rule for the Production Cube

You must add a feeder in the rule for the Production cube.

```
['Production Cost/Kg']=['Production Cost']\['Quantity
Produced - Kgs'];
```

The calculation statement requires a feeder statement in the rule for the Production cube. You can use either Production Cost or Quantity Produced - Kgs to feed Production Cost/Kg. In this example, you'll use Quantity Produced - Kgs, a completely arbitrary choice.

```
['Quantity Produced - Kgs'] => ['Production Cost/Kg'];
```

Procedure

1. Double-click the Production rule in **Server Explorer**.
2. Add the statement immediately following the existing feeder statement in the rule.

The complete rule for the Production cube should now look like this:

```
SHIPCHECK;
```

```
['Production Cost']=DB('FishRequired',!CakeType,'Total
Fish Types',!date,'Cost');
```

```
['Production Cost/Kg']=['Production Cost']\['Quantity
Produced - Kgs'];
```

```
FEEDERS;
```

```
['Quantity Produced - Kgs']=>DB('Plan',!CakeType,!Date,'Planned
Production
Qty - Kgs');
```

```
['Quantity Produced - Kgs']=>DB('Fish Required',!CakeType,'Total
Fish Types',!Date,'Qty Required - Kgs');
```

```
['Quantity Produced - Kgs']=>['Production Cost/Kg'];
```

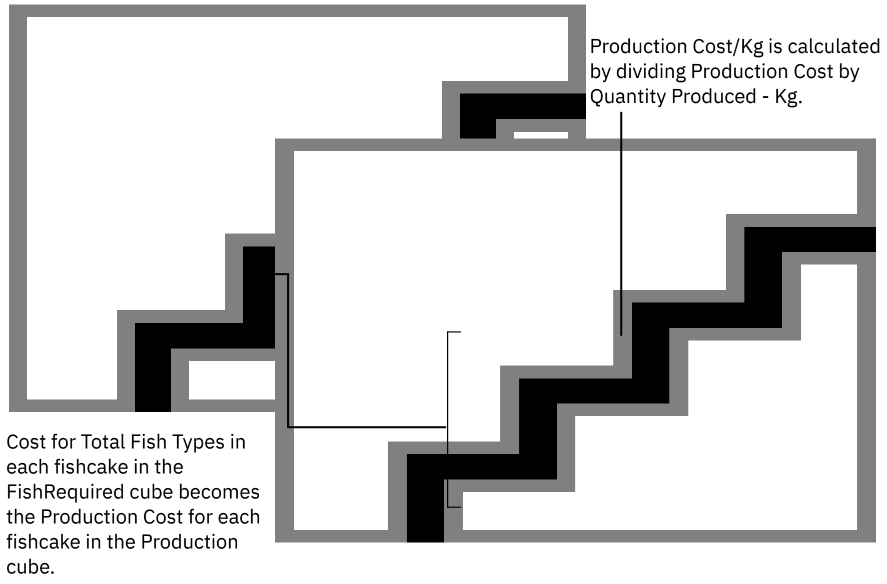
3. Click **Save** to compile and save the rule.

Viewing the Results

Use these views to view the results.

To view the result of the new calculation statements for the Production cube, open the following two views:

- Cost of fish required - Apr. 30 view of the FishRequired cube
- Production costs- Apr 30 view of the Production cube


Production Cost/Kg is calculated by dividing Production Cost by Quantity Produced - Kg.
Cost for Total Fish Types in each fishcake in the FishRequired cube becomes the Production Cost for each fishcake in the Production cube.

Here you can see that the Fishcakes International model reaches its conclusion, with final costs for fishcakes calculated in the Production cube.

Production Cost for each fishcake is calculated by retrieving the Cost of Total Fish Types for the corresponding fishcake in the FishRequired cube. (In other words, the production cost for a fishcake is equal to the total cost of all fish types that are used to make the fishcake.) Production Cost/Kg for each fishcake is calculated by dividing Production Cost by Quantity Produced - Kgs.

Chapter 11. Using the Rules Tracer

IBM Cognos TM1 provides a tool called the **Rules Tracer** to assist in the development and debugging of rules.

The Rules Tracer lets you:

- Trace calculations, to ensure that rules are being assigned to selected cells and calculated properly, or to trace the path of consolidated elements
- Trace feeders, to ensure that selected leaf cells are feeding other cells properly
- Check feeders, to ensure that the children of a selected consolidated cell are fed properly

The **Rules Tracer** is available only in the **Cube Viewer**. You cannot trace rules from the **In-Spreadsheet Browser** or slice worksheets.

Tracing Calculations

You can trace the calculation path of all cube cells that are derived by either rules or consolidation. These cells appear shaded in the **Cube Viewer**.

Tracing a calculation path can help you determine if the value for a cell location is being calculated properly.

Follow the steps below to trace a calculation.

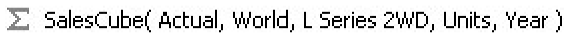
Procedure

1. In the **Cube Viewer**, right-click the cell containing the calculated value.
2. Select **Trace Calculation**.

   The **Rules Tracer** window opens. This window contains two panes.

   The top pane displays the definition of the current cell location, along with an icon indicating whether the value in the location is derived by consolidation or rules. This pane also displays the current value of the cell location. If the value is derived by rules, the rule displays in the lower part of the top pane.

   The bottom pane displays the components of the first consolidated element or first rule in the location definition. For example, the following example displays the definition of a location in the SalesCube.

   Σ SalesCube(Actual, World, L Series 2WD, Units, Year)

   In this definition, Actual is a leaf element while World is a consolidated element. The bottom pane would display the names and values of all the components of the World consolidation.
3. In the bottom pane, double-click the element for the calculation path you want to trace.

   The top pane now displays the definition of the cell location for the element you double clicked. The bottom pane now displays the components of the first consolidated element in this location definition.
4. Continue to double-click through the components for the calculation path you want to trace. If you encounter a rule in the path, you can double-click through to any of the components of the rule.

   When you arrive at the end of a calculation path, the elements in the bottom pane of the **Rules Editor** display a small dot icon. You can double-click one of these elements to view a final calculation path and value.

   As you navigate through a calculation path, the top pane of the **Rules Editor** records your progress as cell locations. If you click on any of these locations, the location becomes active and you can begin tracing a different path from the location.

Tracing Feeders

The **Rules Tracer** lets you trace the way a selected cell feeds other cells.

Since you can only feed from a leaf element, this option is not available for consolidated cells. The option is, however, available for cells defined by rules.

Follow the steps below to trace feeders for a cell.

Procedure

1. In the **Cube Viewer**, right-click the cell you want to trace.
2. Select **Trace Feeders**.

   The **Rules Tracer** window opens. This window contains two panes.

   The top pane displays the definition of the current cell location, as well as the feeder rules associated with the current cell.

   The bottom pane displays the locations fed by the current cell.
3. Double-click a location in the bottom pane.

   This location becomes the current cell location in the top pane, and the bottom pane displays any locations fed by the current cell.
4. Continue double-clicking locations in the bottom pane until you have traced the feeders to the level you require.

   As you trace feeders, the top pane of the **Rules Editor** records your progress as cell locations. If you click on any of these locations, the location becomes active and you can begin tracing new feeders.

Checking Feeders

If a cube has a rule attached that uses `SKIPCHECK` and `FEEDERS`, you can use the Rules Tracer to make sure that the components of consolidations are properly fed.

You can check feeders **ONLY** from a consolidated cell; the **Check Feeders** option is not available from leaf cells.

Follow the steps below to check feeders.

Procedure

1. In the **Cube Viewer**, right-click the consolidated cell you want to check.
2. Select **Check Feeders**.

   The **Rules Tracer** window opens. This window contains two panes.

   The top pane displays the definition of the current cell (consolidation) location.

   The bottom pane displays all components of the consolidation that are not properly fed.

   If the bottom pane is empty, the consolidation is fed properly and cubes values are accurate.

   If the bottom pane displays components of the consolidation, you must edit the rule associated with the current cube to contain FEEDERS statements that feed all the listed

Notices

Copyright and Trademarks

Index