ORACLE
PeopleSoft

PeopleTools 8.62: Analytic Calculation Engine

April 2025

ORACLE

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://docs.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://docs.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit https://docs.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

Preface

Understanding the PeopleSoft Online Help and PeopleBooks

The PeopleSoft Online Help is a website that enables you to view all help content for PeopleSoft applications and PeopleTools. The help provides standard navigation and full-text searching, as well as context-sensitive online help for PeopleSoft users.

Hosted PeopleSoft Online Help

You can access the hosted PeopleSoft Online Help on the Oracle Help Center. The hosted PeopleSoft Online Help is updated on a regular schedule, ensuring that you have access to the most current documentation. This reduces the need to view separate documentation posts for application maintenance on My Oracle Support. The hosted PeopleSoft Online Help is available in English only.

To configure the context-sensitive help for your PeopleSoft applications to use the Oracle Help Center, see Configuring Context-Sensitive Help Using the Hosted Online Help Website.

Locally Installed PeopleSoft Online Help

If you're setting up an on-premises PeopleSoft environment, and your organization has firewall restrictions that prevent you from using the hosted PeopleSoft Online Help, you can install the online help locally. Installable PeopleSoft Online Help is made available with selected PeopleSoft Update Images and with PeopleTools releases for on-premises installations, through the Oracle Software Delivery Cloud.

Your installation documentation includes a chapter with instructions for how to install the online help for your business environment, and the documentation zip file may contain a README.txt file with additional installation instructions. See *PeopleSoft 9.2 Application Installation* for your database platform, "Installing PeopleSoft Online Help."

To configure the context-sensitive help for your PeopleSoft applications to use a locally installed online help website, see Configuring Context-Sensitive Help Using a Locally Installed Online Help Website.

Downloadable PeopleBook PDF Files

You can access downloadable PDF versions of the help content in the traditional PeopleBook format on the Oracle Help Center. The content in the PeopleBook PDFs is the same as the content in the PeopleSoft Online Help, but it has a different structure and it does not include the interactive navigation features that are available in the online help.

Common Help Documentation

Common help documentation contains information that applies to multiple applications. The two main types of common help are:

- Application Fundamentals

- Using PeopleSoft Applications

Most product families provide a set of application fundamentals help topics that discuss essential information about the setup and design of your system. This information applies to many or all applications in the PeopleSoft product family. Whether you are implementing a single application, some combination of applications within the product family, or the entire product family, you should be familiar with the contents of the appropriate application fundamentals help. They provide the starting points for fundamental implementation tasks.

In addition, the *PeopleTools: Applications User's Guide* introduces you to the various elements of the PeopleSoft Pure Internet Architecture. It also explains how to use the navigational hierarchy, components, and pages to perform basic functions as you navigate through the system. While your application or implementation may differ, the topics in this user's guide provide general information about using PeopleSoft applications.

Field and Control Definitions

PeopleSoft documentation includes definitions for most fields and controls that appear on application pages. These definitions describe how to use a field or control, where populated values come from, the effects of selecting certain values, and so on. If a field or control is not defined, then it either requires no additional explanation or is documented in a common elements section earlier in the documentation. For example, the Date field rarely requires additional explanation and may not be defined in the documentation for some pages.

Typographical Conventions

The following table describes the typographical conventions that are used in the online help.

Typographical Convention	***Description***
Key+Key	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For **Alt+W,** hold down the **Alt** key while you press the **W** key.
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object. Ampersands also precede all PeopleCode variables.

Typographical Convention	*Description*
⇒	This continuation character has been inserted at the end of a line of code that has been wrapped at the page margin. The code should be viewed or entered as a single, continuous line of code without the continuation character.

ISO Country and Currency Codes

PeopleSoft Online Help topics use International Organization for Standardization (ISO) country and currency codes to identify country-specific information and monetary amounts.

ISO country codes may appear as country identifiers, and ISO currency codes may appear as currency identifiers in your PeopleSoft documentation. Reference to an ISO country code in your documentation does not imply that your application includes every ISO country code. The following example is a country-specific heading: "(FRA) Hiring an Employee."

The PeopleSoft Currency Code table (CURRENCY_CD_TBL) contains sample currency code data. The Currency Code table is based on ISO Standard 4217, "Codes for the representation of currencies," and also relies on ISO country codes in the Country table (COUNTRY_TBL). The navigation to the pages where you maintain currency code and country information depends on which PeopleSoft applications you are using. To access the pages for maintaining the Currency Code and Country tables, consult the online help for your applications for more information.

Region and Industry Identifiers

Information that applies only to a specific region or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a region-specific heading: "(Latin America) Setting Up Depreciation"

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in the PeopleSoft Online Help:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in the PeopleSoft Online Help:

- USF (U.S. Federal)

- E&G (Education and Government)

Translations and Embedded Help

PeopleSoft 9.2 software applications include translated embedded help. With the 9.2 release, PeopleSoft aligns with the other Oracle applications by focusing our translation efforts on embedded help. We are not planning to translate our traditional online help and PeopleBooks documentation. Instead we offer very direct translated help at crucial spots within our application through our embedded help widgets. Additionally, we have a one-to-one mapping of application and help translations, meaning that the software and embedded help translation footprint is identical—something we were never able to accomplish in the past.

Using and Managing the PeopleSoft Online Help

Select About This Help in the left navigation panel on any page in the PeopleSoft Online Help to see information on the following topics:

- Using the PeopleSoft Online Help.
- Managing hosted Online Help.
- Managing locally installed PeopleSoft Online Help.

PeopleTools Related Links

PeopleTools 8.62 Home Page

PeopleSoft Search and Insights Home Page

“PeopleTools Product/Feature PeopleBook Index” (Getting Started with PeopleTools)

PeopleSoft Online Help

PeopleSoft Information Portal

PeopleSoft Spotlight Series

PeopleSoft Training and Certification | Oracle University

My Oracle Support

Oracle Help Center

Contact Us

Send your suggestions to psoft-infodev_us@oracle.com.

Please include the applications update image or PeopleTools release that you’re using.

Follow Us

Icon	*Link*
	Watch PeopleSoft on YouTube
	Follow @PeopleSoft_Info on X.
	Read PeopleSoft Blogs
	Connect with PeopleSoft on LinkedIn

Chapter 1

Getting Started with PeopleSoft Analytic Calculation Engine

Understanding Analytic Calculation Engine

Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features that enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multidimensional reporting, data editing, and analysis.

Specifically, application developers create analytic models to define the rules that are used to calculate data. Application developers also create PeopleSoft Pure Internet Architecture pages with analytic grids to display the data within PeopleSoft applications. Within the application, a PeopleSoft Pure Internet Architecture page with an analytic grid may be referred to as an interactive report. End users view, analyze, and make changes to analytic model data. When end users save their changes, Analytic Calculation Engine recalculates the data and sends the calculated data to the application database.

Common Elements Used in This Document

Term	*Definition*
	Click the **New Cube Collection** icon to create a new cube collection.
	Click the **New Cube** icon to create a new data cube.
	Click the **Attach Cubes** icon to add existing data cubes to the selected cube collection.
	Click the **New Dimension** icon to create a new dimension.
	Click the **Attach Dimension** icon to attach one or more dimensions to the selected data cube.
	Click the **Detach Dimension** icon to detach one or more dimensions from the selected data cube.

Term	*Definition*
	Click the **Validate Model** icon to validate the analytic model.
	Click the **Find Part** icon to find all of the locations of the selected part in the analytic model.
	Click the **Causes and Effects Tool** icon to browse through the cube collections and data cubes of your analytic model to view the causes, effects, and inputs of data cubes.
	Click the **Direct Causes** icon to display the direct causes of the selected data cube.
	Click the **Direct Effects** icon to display the direct effects of the selected data cube.
	Click the **Accept Changes** icon to accept the changes you made to the rule.
	Click the **Cancel Changes** icon to cancel the changes you made to the rule.

Analytic Calculation Engine Implementation

The functionality to create your own analytic models is delivered as part of standard PeopleSoft PeopleTools that are provided with all PeopleSoft products. However, you must complete these activities before you can create analytic models:

Step	*Reference*
Use the PeopleSoft DPKs to deploy a full PeopleSoft application environment, including database, application server, Process Scheduler, and web server (PIA).	See *PeopleSoft 9.2 Application Installation* for your database platform.
Enable the appropriate number of analytic server instances.	See Configuring and Starting Analytic Servers.
Establish a user profile that gives you access to PeopleSoft Application Designer and any other processes that you will use.	See “Security Basics” (Security Administration).

Step	*Reference*
Configure the application for which you are creating or changing an analytic model.	The appropriate product documentation for your application.

Chapter 2

Understanding Oracle's PeopleSoft Analytic Calculation Engine

Analytic Calculation Engine Components

This topic discusses the components of Analytic Calculation Engine.

Analytic Model

Use PeopleSoft Application Designer to create an analytic model. When you create an analytic model, you:

- Define data cubes, dimensions, cube collections, and other parts that are used to organize and calculate individual fields of data.
- Map records to cube collections within the analytic model.
- Map individual record fields to data cubes and dimensions within the cube collections.

Analytic Type

Both Analytic Calculation Engine and PeopleSoft Optimization Framework use analytic types. Use PeopleSoft Application Designer to create an analytic type definition, which defines the caching behavior of the records that the analytic model uses, specifies the records that are accessible by all end users, and specifies the records that are only accessible by certain users for what-if scenario forecasting.

See Purpose of Analytic Type Definitions, “Understanding Analytic Type Definitions” (Optimization Framework), “Creating Analytic Type Definitions” (Optimization Framework).

Analytic Instances

Analytic instances are runtime instances of analytic types that are loaded into analytic server instances by means of the AnalyticInstance class Load method. The AnalyticInstance class is one of the Analytic Calculation Engine classes. To view and edit analytic model data, an end user selects an analytic instance ID within an application's PeopleSoft Pure Internet Architecture pages.

Analytic Calculation Engines

Analytic calculation engines run inside analytic server instances. They calculate analytic instance data by using the relationships and rules that are defined in the analytic model.

Analytic Server Instances

When used with Analytic Calculation Engine, analytic server instances are processes in the application server domain that contain and run analytic instances and analytic calculation engines. When a user

selects an analytic instance ID within an application, the system generates one analytic calculation engine and one analytic server instance. *One analytic server instance can contain one and only one analytic instance, and one and only one analytic calculation engine.* All three of these components, plus the application server, exist in *one* application server domain. An application server can communicate only with analytic server instances that exist in the same application server domain as the application server. For this reason, a PeopleSoft application that exists in one application server domain cannot communicate with an analytic model that exists in a different application server domain.

The PSANALYTICREG table contains a row that provides information about each analytic server instance that is running within an application server domain. You administer analytic server instances by using the Summary and Servers pages that display the data from the PSANALYTICREG table.

See Administering Analytic Servers.

Analytic Calculation Engine Classes

You use the Analytic Calculation Engine classes for all runtime operations between PeopleSoft applications and analytic calculation engines. Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) data cube values. The Analytic Calculation Engine classes run on the application server and use Tuxedo service requests to communicate with analytic server instances.

The Analytic Calculation Engine classes contain the AnalyticInstance classes, which are used by Analytic Calculation Engine and PeopleSoft Optimization Framework to manipulate analytic instance definitions with PeopleCode. Use the AnalyticInstance classes to manipulate analytic instance definitions at runtime.

See Understanding Analytic Calculation Engine.

Analytic Calculation Engine Metadata Classes

The Analytic Calculation Engine Metadata classes are application classes that PeopleSoft applications use to create and change analytic model metadata. For example, using the Analytic Calculation Engine Metadata classes you could modify a calculation rule. Applications can use the Analytic Calculation Engine Metadata classes to perform all of the analytic model-related actions that are available in PeopleSoft Application Designer.

See “Understanding the Analytic Calculation Engine Classes” (PeopleCode API Reference).

AnalyticType Classes

The AnalyticType classes are PeopleCode application classes that PeopleSoft applications use to manipulate analytic type definitions. Use the AnalyticType classes at runtime to perform all of the analytic type definition-related actions that are available in PeopleSoft Application Designer.

See “Understanding the PeopleSoft Analytic Calculation Engine Metadata Classes” (PeopleCode API Reference), “Analytic Model Metadata Classes Examples” (PeopleCode API Reference).

AnalyticGrid Classes

Analytic Calculation Engine uses the AnalyticGrid classes to manipulate analytic grids using PeopleCode. Use the AnalyticGrid classes to manipulate the display or data of analytic grids at runtime.

See “Understanding the Analytic Calculation Engine Classes” (PeopleCode API Reference).

PeopleSoft Pure Internet Architecture Pages with Analytic Grids

Use PeopleSoft Application Designer to create PeopleSoft Pure Internet Architecture pages with analytic grids. Pages with analytic grids display Analytic Calculation Engine data and application data within PeopleSoft applications. Within the application, a PeopleSoft Pure Internet Architecture page with an analytic grid may be referred to as an *interactive report.* Analytic grids provide drag-and-drop functionality so end users can view their data in different ways.

See Understanding Analytic Grid Design.

Analytic Model Viewer

The Analytic Model Viewer helps developers debug and analyze analytic models by enabling them to view an analytic model's parts and to view and edit an analytic model's application data.

See Understanding the Analytic Model Viewer, Viewing Analytic Model Properties, Using Analytic Model Viewer Alongside PeopleSoft Application Designer.

Analytic Instance Capture Utility

When experiencing problems with an application that uses an analytic model, customers can use the Analytic Instance Capture Utility to package analytic model data and metadata to send to PeopleSoft support for analysis.

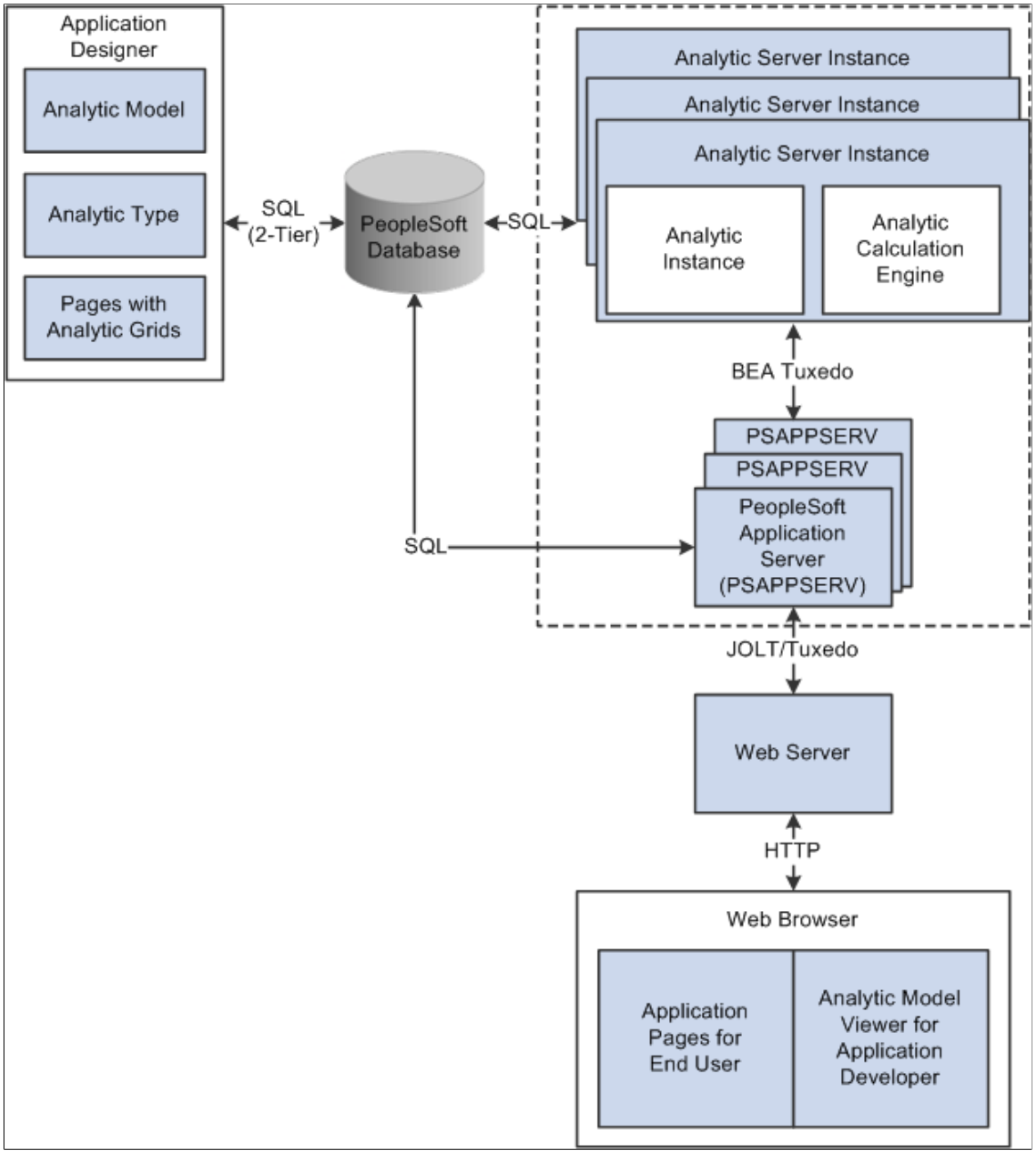
See Understanding the Analytic Instance Capture Utility, Capturing Analytic Instance Data, Importing Analytic Instance Data.

PeopleSoft Performance Monitor

PeopleSoft Performance Monitor enables you to monitor Analytic Calculation Engine and view real-time and historical performance data. PeopleSoft Performance Monitor provides the information that you need to solve immediate performance issues as well as to analyze trends in system performance.

Analytic Calculation Engine Architecture

This diagram illustrates how the various Analytic Calculation Engine architecture components work together.



Analytic Calculation Engine Development Process

You should follow the development process outlined here for your PeopleSoft application to correctly employ the features of Analytic Calculation Engine.

Development Process Without Existing Record Structures

If you *do not* have existing record structures on which to base your analytic model, perform this iterative process:

- Create the record definitions while creating the analytic model's cube collections.

  See “Viewing Record Definitions” (Application Designer Developer’s Guide), “Saving Record Definitions” (Application Designer Developer’s Guide), “Naming Record Definitions” (Application Designer Developer’s Guide), “Creating New Record Definitions” (Application Designer Developer’s Guide), Creating Cube Collections.

- Create the rest of the analytic model.

  See Creating a New Analytic Model Definition.

- Define application data security.

  See “Application Data Security” (Security Administration).

- Create an analytic type definition.

  At the appropriate step in this process, you must attach the analytic type to the analytic model.

- Attach the analytic type to the analytic model.

  See “Creating Analytic Type Definitions” (Optimization Framework), Creating a New Analytic Model Definition.

- Create PeopleSoft Pure Internet Architecture pages with analytic grids.

  Note: Within the application, a PeopleSoft Pure Internet Architecture page with an analytic grid may be referred to as an *interactive report*.

  See Understanding Analytic Grid Design, Setting Analytic Grid General Properties, Setting Column Properties for Analytic Grids.

- As needed, create pages and any required PeopleCode to administer analytic instances.

  See “Understanding Page Design and Development” (Application Designer Developer’s Guide), “Using Page Development Tools” (Application Designer Developer’s Guide), “Creating New Page Definitions” (Application Designer Developer’s Guide), “Understanding the Analytic Calculation Engine Classes” (PeopleCode API Reference).

- As needed, write PeopleCode programs using the appropriate classes (Analytic Calculation Engine, Analytic Calculation Engine Metadata, AnalyticGrid, AnalyticType) to manipulate the analytic model, its data, and its display as necessary for your application.

  See “Understanding the PeopleSoft Analytic Calculation Engine Metadata Classes” (PeopleCode API Reference).

- As needed, write Application Engine programs for batch calculations.

  See “Viewing Application Engine Programs” (Application Engine), “Creating, Opening, and Renaming Programs” (Application Engine).

- As needed, write application pages that enable end users to load analytic instances.

  Note: You can also embed analytic instance loading functionality into PeopleSoft Pure Internet Architecture pages with analytic grids.

Development Process Using Existing Record Structures

If you have existing record structures on which the analytic model should be based, perform this iterative process:

- Create a new analytic model.

  At the appropriate step in this process, create the analytic model's cube collections to retrieve data from the records.

  See Creating Cube Collections.

- Create an analytic type definition.
- Attach the analytic type to the analytic model.

  See “Creating Analytic Type Definitions” (Optimization Framework), Understanding the Analytic Model Definition Creation Process, Creating a New Analytic Model Definition.

- Create PeopleSoft Pure Internet Architecture pages with analytic grids.

  Note: Within the application, a PeopleSoft Pure Internet Architecture page with an analytic grid may be referred to as an *interactive report.*

  See Understanding Analytic Grid Design, Setting Analytic Grid Analytic Properties.

- As needed, create pages and any required PeopleCode to administer analytic instances.

  See “Understanding Page Design and Development” (Application Designer Developer’s Guide), “Creating New Page Definitions” (Application Designer Developer’s Guide), “Understanding the Analytic Calculation Engine Classes” (PeopleCode API Reference).

- As needed, write PeopleCode programs using the appropriate classes (Analytic Calculation Engine, Analytic Calculation Engine Metadata, AnalyticGrid, AnalyticType) to manipulate the analytic model, its data, and its display as necessary for the application.

  See “Using the Analytic Type Classes” (PeopleCode API Reference), “How to Create an Analytic Type Class Object” (PeopleCode API Reference).

- As needed, write Application Engine programs for batch calculations.

  See “Viewing Application Engine Programs” (Application Engine), “Creating, Opening, and Renaming Programs” (Application Engine).

- As needed, write application pages that enable end users to load analytic instances.

  Note: You can also embed this functionality into PeopleSoft Pure Internet Architecture pages with analytic grids.

Analytic Calculation Engine Security

Analytic Calculation Engine does not provide additional data security features beyond what is already available in PeopleTools. You define analytic model data security within the application that uses the analytic model by creating a view for each read/write cube collection. Additionally, you can:

- Create filter functions to restrict the data that appears in the analytic grid.

  See Filter User Functions.

- Create filter functions that filter data by user ID.

  See OPRID.

Chapter 3

Understanding Analytic Models

Analytic Models

An analytic model is an information workshop. Just like an ordinary workshop, it contains parts that you use to build your projects and tools to put the parts together. But instead of building a cabinet or a chair, you organize data by building analytic models of information. This analytic model imitates the structure and relationships of information in the real world.

You can think of an analytic model as a collection of various kinds of information that are held together by a common purpose. For example, you can create an analytic model of an entire business, with information about revenues, employee expenses, accounts receivable, assets, liabilities, equity, and so on. You can also create an analytic model of a particular part of a business—such as employee expenses—and include more detail than you would in a more general analytic model of a business. The focus can be wide or narrow, but all the information about the area of interest goes into a single analytic model.

Because of an analytic model's multidimensional capabilities, end users analyze data from different angles to gain insight into their data. This data can range from a small table of values to a very large table containing hundreds of kinds of data about thousands of people, places, or things.

Relationship of Parts

This topic discusses the relationship of parts in an analytic model.

Data Cubes and Dimensions

The primary parts in an analytic model are data cubes (cubes) and dimensions:

- A data cube is like a sheet of paper that contains one and only one kind of data.

  When you build an analytic model, you create a data cube for each kind of information in the analytic model. For example, an analytic model of a business might contain a data cube for sales, a data cube for rent, a data cube for salary, and so on.

- A dimension contains a list of one kind of data that can span various contexts.

  For example, an analytic model of a business might contain the PRODUCT_CODES dimension and the MONTHS dimension. These two dimensions can be used in both a SALES cube collection and a COST_OF_GOODS cube collection to track the products sales and costs over a period of months.

- A dimension member (member) is one list item within a dimension.

  Maximum Length of Dimension member is 30. Many different kinds of dimension members exist. For example, the 010 product code is a detail member of the PRODUCT_CODES dimension. Western Europe is an aggregate member of the REGIONS dimension.

See Types of Dimension Members.

Note: You do not create dimension members in the analytic model definition. Instead, dimension members are dynamically created during runtime.

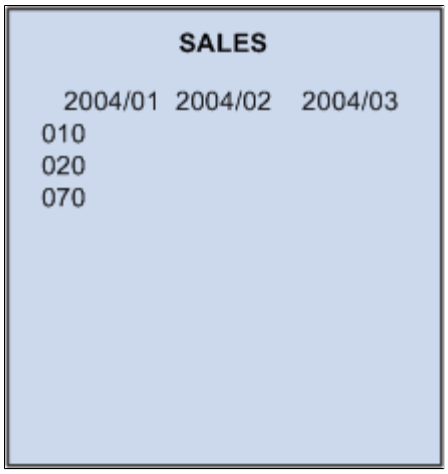
See Creation of New Members at Runtime.

Note: The maximum number of dimensions attached to a data cube is 31. The aggregation routines fail if the attached dimensions are greater than 31.

Data cubes and dimensions work together to create the structure of the analytic model. To see how this works, imagine writing *SALES* at the top of a blank sheet of paper. This is the equivalent of creating a new data cube.

You could write only a single value in the SALES cube, but a cube with only one value is not useful. So your next action is to write a list of months across the top of the cube and a list of product codes down the side of the cube.

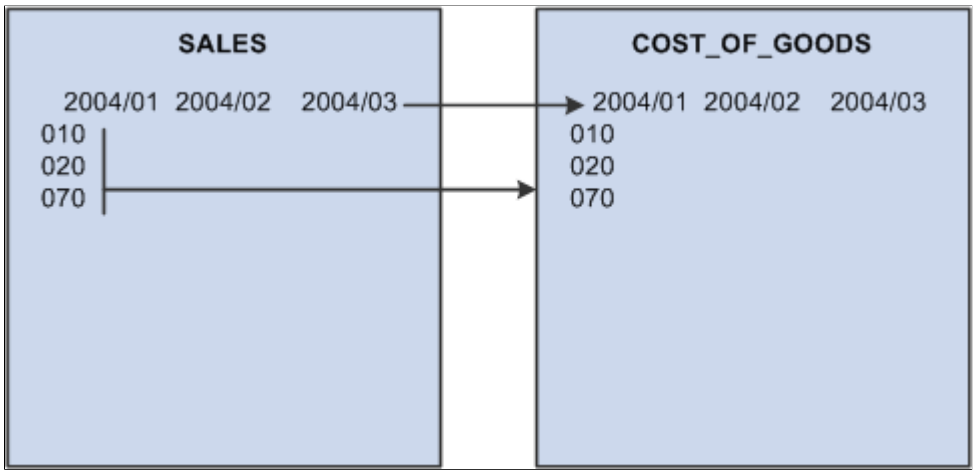
This is an example of SALES data cube with attached PRODUCT_CODES and MONTHS dimensions.



The SALES cube now contains a value for every month and product code because you attached two dimensions to the cube.

Dimensions are separate objects that can be used independently of data cubes. For this reason, even though you originally created the MONTHS and PRODUCTS dimensions for the SALES cube, you can reuse these dimensions with other data cubes. For example, imagine taking a new sheet of paper and writing *COST_OF_GOODS* at the top of the page, and then attaching the existing dimensions to the new sheet.

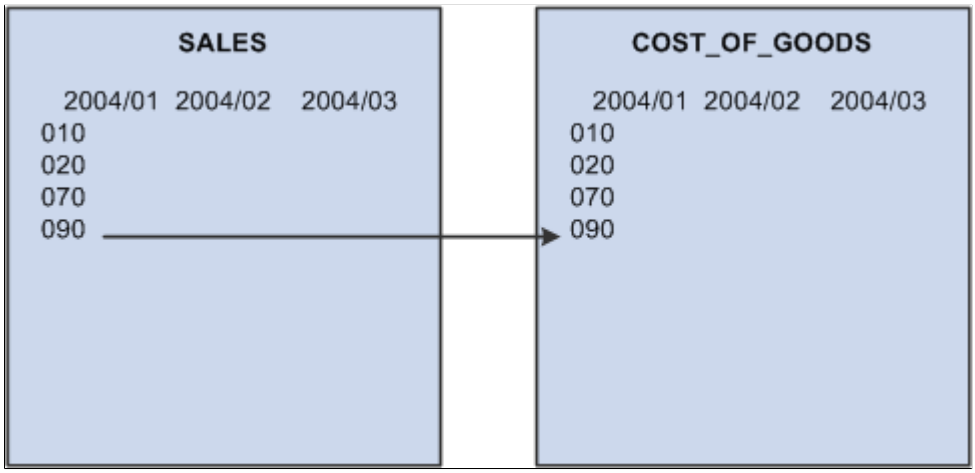
This is an example of attaching the PRODUCT_CODES and MONTHS dimensions to the COST_OF_GOODS data cube.



It is tempting to think that the dimensions attached to the COST_OF_GOODS cube are copies of the dimensions on the SALES cube. This might be true on paper, but an analytic model works differently. The dimensions are independent objects that you attach to the data cubes. For this reason, the MONTHS dimension that is attached to the COST_OF_GOODS cube is the same dimension as the MONTHS dimension that is attached to the SALES cube. Therefore, any change that an application makes in a dimension is reflected on all data cubes that use that dimension.

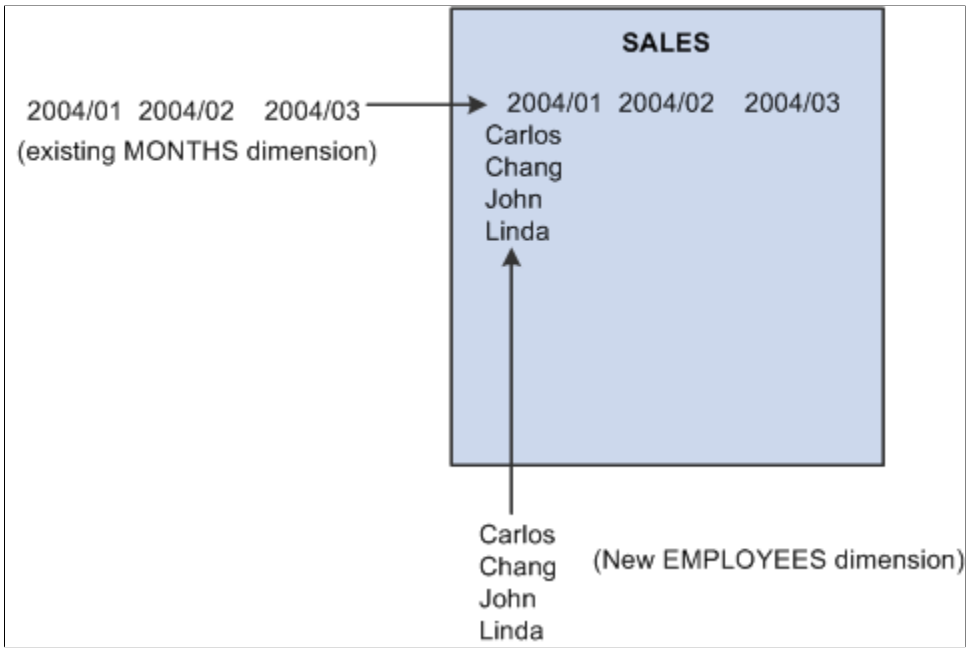
For example, suppose the application adds a product code called 090 to the PRODUCT_CODES dimension on the SALES cube. The analytic model adds 090 to the PRODUCTS dimension on the COST_OF_GOODS cube.

This is an example of adding the 090 product code to the PRODUCT_CODES dimension on the SALES cube.



When updating an analytic model, you can use a combination of existing and new dimensions when you define a data cube. For example, suppose you create a cube called SALARY. You want to track monthly data, so you attach the existing MONTHS dimension to the data cube. You also want to track the salary for each employee, so you create a new EMPLOYEES dimension and attach it to the SALARY cube.

This is an example of SALARY data cube with attached MONTHS dimension and new EMPLOYEES dimension.



Formulas and User Functions

You create formulas to define the relationships among the data cubes in an analytic model. For example, this formula for the GROSS_MARGIN data cube defines the relationship between GROSS_MARGIN and other data cubes called SALES and COST_OF_GOODS:

```
SALES - COST_OF_GOODS
```

The analytic calculation engine recalculates the values in the GROSS_MARGIN data cube whenever the end user changes the values in the SALES or COST_OF_GOODS data cubes and saves those changes.

Formulas refer to kinds of information as defined by data cubes. Formulas do not refer to specific values. The relationship between GROSS_MARGIN, SALES, and COST_OF_GOODS remains true regardless of the specific data contained in these data cubes. You can attach new products or months to the data cubes without changing or copying the formula because the relationships between the data cubes have not changed.

User functions serve several purposes. You can create a user function that contains all or part of a formula and apply this user function to calculate multiple data cubes. You can also create user functions to define filter functions and rules for aggregating dimension members.

See Understanding Rules, Formulas, and User Functions, Understanding Dimension Members.

Cube Collections

A cube collection is a collection of related data cubes. You create cube collections to load data from the database into the analytic model, save data back to the database, and display calculated data to the end user at runtime. Some cube collections contain data cubes that receive user input, and other cube collections calculate data cubes and display calculated data to the end user.

See Creating Cube Collections.

Organizers

You can use organizers to arrange an analytic model's parts for more convenient viewing and editing. Within an analytic model, you can place any of the following parts in as many different organizers as you want:

- Cube collections.
- Data cubes.
- Dimensions.
- User functions.
- Organizers.

  You can drag and drop parts into and between organizers, place suborganizers into organizers, and drag and drop organizers and suborganizers to arrange their positions in the Organizers branch of the part browser.

Related Links

Creating Organizers

Tools

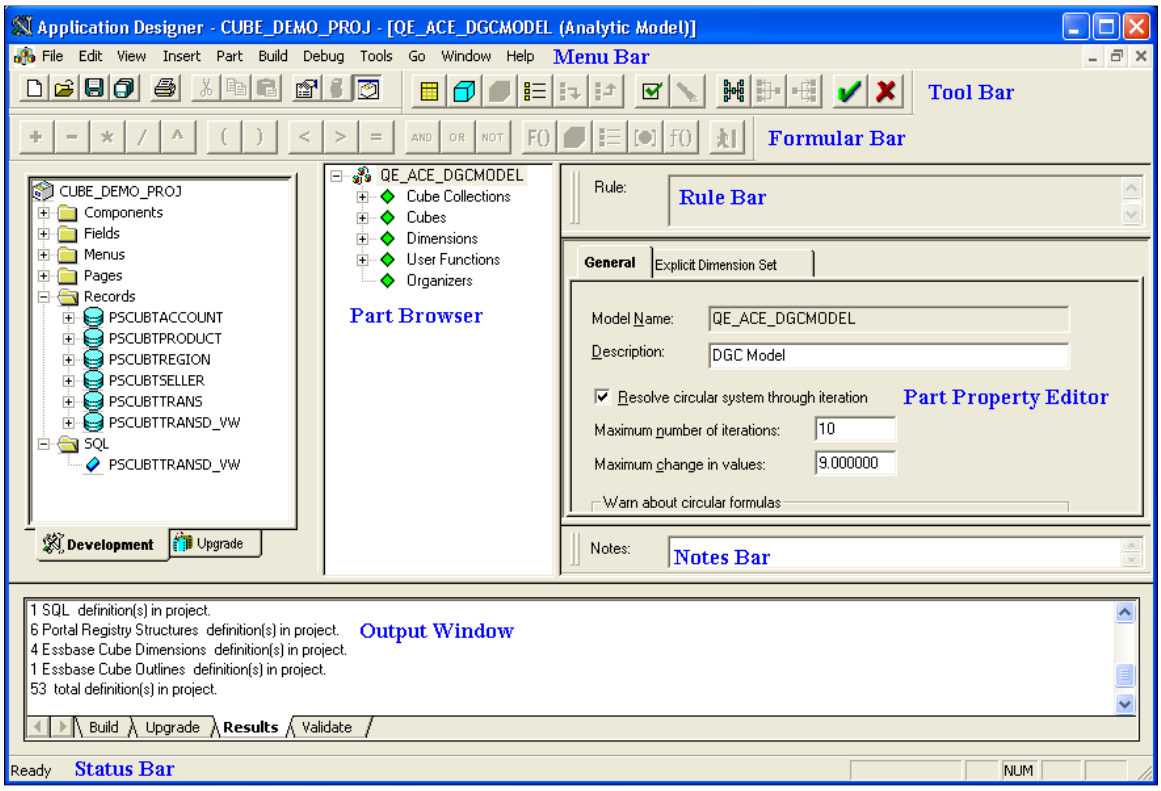
This topic discusses different tools available in PeopleSoft Application Designer for creating analytic models.

PeopleSoft Application Designer Window Components for Creating Analytic Model Definitions

Navigation:

Start > **Programs** > **PeopleTools 8.5x** > **Application Designer**

This is an example of PeopleSoft Application Designer interface that is used for creating analytic model definitions.



Term	***Definition***
Menu Bar	Provides access to commands and features that you use to create analytic model definitions. See the Menu Bar section below.
Tool Bar	Contains buttons that you use to perform common commands and edit the parts of an analytic model definition.
Formula Bar	Contains buttons that you use to define formulas for data cubes and user functions.
Rule Bar	Displays the rules for data cubes and user functions. **Note:** The type of information that appears in the rule bar depends on the part that is currently selected.
Part Browser	Contains hierarchies that you use to view, organize, and edit an analytic model definition's parts. See the Part Browser section below.

Term	*Definition*
Part Property Editor	Contains dialog boxes that you use to edit the parts of an analytic model definition.
Notes Bar	Enables you to enter notes about the different parts of the analytic model definition.
Output Window	Contains the output text from PeopleSoft Application Designer operations, such as Build (SQL Create and Alter), Find Definition References, Upgrade, Results, Validate, and PeopleCode Log.
Status Bar	Contains descriptions of buttons and menu commands.

Behavior of Bars

All of the bars—except for the menu bar and status bar—are dockable. You can drag the bars to the top, bottom, left, or right sides of the PeopleSoft Application Designer interface.

To float the bars, drag them away from the edges of the PeopleSoft Application Designer interface. You can then resize them vertically and horizontally.

Menu Bar

This topic reviews PeopleSoft Application Designer menu commands that you use to create analytic model definitions.

Note: This topic does *not* review all of PeopleSoft Application Designer menu commands.

See “Using the PeopleSoft Application Designer User Interface” (Application Designer Developer’s Guide).

The following table shows the analytic model definition specific commands in the Edit menu.

Edit Menu Commands	*Usage*	*Quick Keys*
Paste Function	Select to insert a built-in function and its arguments into a rule.	**Ctrl**+**Shift**+**F**
Paste Cube Name	Select to insert a reference to the data cube into a rule.	**Ctrl**+**Shift**+**N**

Edit Menu Commands	*Usage*	*Quick Keys*
Paste Member Ref...	Insert a reference to a dimension into a rule. After Analytic Calculation Engine inserts the dimension reference, complete the syntax for a member reference. See Inserting a Dimension Member Reference into a Rule.	**Ctrl**+**Shift**+**M**
Paste Dimension	Insert a reference to a dimension into a rule.	None
Paste User Function	Insert a reference to a user function into a rule.	None

The following table lists the analytic model definition specific commands in the View menu.

View Menu Commands	*Usage*
Notes Bar	Active and deactivate the notes bar.
Rule Bar	Active and deactivate the rule bar.

The following table lists the analytic model definition specific commands in the Part menu. You can access the same Part menu commands by right-clicking the part or subbranch that you want to add to or edit.

Part Menu Commands	*Secondary Menu Commands*	*Usage*	*Quick Keys*
New	Cube Collection	Create a new cube collection.	**Ctrl**+**L**
New	Data Cube	Create a new data cube.	**Ctrl**+**D**
New	Dimension	Create a new dimension.	**Ctrl**+**E**
New	User Function	Create a new user function.	**Ctrl**+**U**
New	Organizer	Create a new organizer.	None
Attach	Data Cubes...	Attach one or more existing data cubes to the selected cube collection.	**Ctrl**+**S**+**D**

Part Menu Commands	*Secondary Menu Commands*	*Usage*	*Quick Keys*
Attach	Dimensions...	Attach one or more existing dimensions to the selected data cube.	**Ctrl**+**Shift**+**E**
Move	Up	Move the selected part one position up in the part browser.	**Alt**+**Up**
Move	Down	Move the selected part one position down in the part browser.	**Alt**+**Down**
Clone Part	None	Make a copy of the selected part.	None
Detach	Data Cubes...	Detach one or more data cubes from the selected cube collection.	None
Detach	Dimensions...	Detach one or more dimensions from the selected data cube.	None
Delete Part	None	Delete the selected parts.	None

The following table lists the analytic model definition specific commands in the Tools menu.

Tools Menu Commands	*Secondary Menu Commands*	*Tertiary Menu Commands*	*Usage*	*Quick Keys*
Analytic Model	Validate	None	Validate the analytic model.	None
Analytic Model	Find Part...	None	Find where the current part is used by other parts in the analytic model.	None
Analytic Model	Causes and Effects Tool...	None	Launch the Causes and Effects Tool.	None
Analytic Model	Causes	Direct Causes	Display the direct causes of the selected data cubes.	**Alt** + **<**

Tools Menu Commands	*Secondary Menu Commands*	*Tertiary Menu Commands*	*Usage*	*Quick Keys*
Analytic Model	Causes	All Causes	Display all causes of the selected data cubes.	None
Analytic Model	Causes	All Inputs	Display all of the input data cubes that directly or indirectly affect the selected data cubes.	None
Analytic Model	Effects	Direct Effects	Display the direct effects of the selected data cubes.	**Alt** + **>**
Analytic Model	Effects	All Effects	Display all effects of the selected data cubes.	None
Analytic Model	Circular Formulas	Show Circular System	Show the data cubes involved in the circular system.	None

Part Browser

The part browser is a treelike structure whose main branch is the analytic model itself. The main branch contains several subbranches, as described in the following table.

Branch	*Description*
Cube Collections	Contains a subbranch for each cube collection in the analytic model. Double-click a cube collection to edit the cube collection's properties. Expand a cube collection to view all of the dimensions and data cubes in a cube collection. Click a dimension or data cube to edit its properties.
Cubes	Contains a subbranch for each data cube in the analytic model. Click a data cube to edit its properties. Expand a data cube view all of the dimensions that are attached to the data cube. Click a dimension to edit its properties.
Dimensions	Displays each dimension in the analytic model. Click a dimension to edit its properties.
User Functions	Displays each user function in the analytic model. Click a user function to edit its properties.
Organizers	Contains a subbranch for each organizer in the model. Expand the organizers to view the parts within the organizers.

Dragging and Dropping Parts in the Part Browser

You can drag and drop parts in the part browser to:

- Add new data cubes to cube collections.
- Attach data cubes to dimensions.
- Rearrange the order of dimensions.
- Rearrange the hierarchy of organizers.

Chapter 4

Creating Analytic Model Definitions

Understanding the Analytic Model Definition Creation Process

You create analytic model definitions to define the rules that are used to calculate application data in the Analytic Calculation Engine. This topic provides a high-level discussion of the steps for creating a new analytic model definition assuming that you already have record structures on which to base your analytic model.

To create an analytic model:

1. Create a new analytic model definition.

   See Creating a New Analytic Model Definition.

2. Perform these tasks in the order that is appropriate to your own development needs:

   - Create the analytic model definition's dimensions.

     See Understanding Dimensions, Creating a New Dimension.

   - Create the analytic model definition's data cubes and set the properties of the data cubes.

     Consider the kinds of information that the end user should analyze. These kinds of information should be your data cubes.

     See Understanding Data Cubes, Creating Input Data Cubes, Creating Calculation Data Cubes, Creating Association Data Cubes, Creating Virtual Data Cubes.

   - Define formulas and user functions to calculate the data cubes.

     Define a formula for each data cube that you want to calculate. If you want to reuse the formula in more than one data cube, create a user function and reference the user function in the formula for each of the data cubes.

     See Defining and Editing Data Cube Formulas.

   - Attach dimensions to the data cubes.

     Attach the dimensions to the data cubes after you have created the dimensions and data cubes.

     See Attaching a Dimension to a Data Cube.

   - Define the analytic model's cube collections.

     See Creating Cube Collections.

   - Define the analytic model definition's filter functions.

See Filter User Functions.

- Define the analytic model definition's organizers.

  See Creating Organizers.

3. Save the analytic model definition.

Understanding Conventions for Naming Analytic Models and Parts

You must adhere to these rules when naming analytic models and all analytic model parts:

- Names must consist only of letters, numbers, and underscores (_).

  Other than underscores, do not use nonalphanumeric characters.

- All letters must be uppercase.
- The first character in a name must consist of a letter.

  Do not use a number or underscore as the first character in a name.

- Blank spaces are not allowed in names.

  Use underscores instead of blank spaces.

- Names must not exceed 30 characters.

Related Links

Understanding Part Conversion Details
“AnalyticModelDefn Class” (PeopleCode API Reference)

Creating a New Analytic Model Definition

To open a new analytic model definition:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access Oracle's PeopleSoft Application Designer.
2. Select **File** > **Open** after signing in to the PeopleSoft Application Designer.

   The New Definition dialog box appears.

3. Select the *Analytic Model* option.
4. Click the **OK** button.

The new analytic model definition appears.

Opening an Analytic Model Definition

To open an analytic model definition:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. Select **File** > **Open** after signing in to the PeopleSoft Application Designer.

   The Open Definition dialog box appears.
3. Select the *Analytic Model* option in the **Definition** drop-down list box.
4. Provide selection criteria.

   Enter an analytic model definition name or description (or the beginning characters of either), or select a project.
5. Click the **Open** button or press **Enter** to display analytic model definitions matching the selection criteria that you entered.

   To clear the current selection criteria and start over, click the **New Search** button. To change how the search list appears, perform one of these actions:

   - Click the **List** button to view only the names of the analytic model definitions.
   - Click the **Details** button to view the names and descriptions.

     By default, both the names and descriptions appear in the search list.
6. Double-click the analytic model definition that you want to open in the definition workspace, or highlight the analytic model definition and click the **Open** button.

   You can also press **Shift**+**Left Click** to select more than one definition to open in a single action, or right-click to view a pop-up menu from which you can open, print, rename, or delete the selected analytic model definition.

Creating Organizers

To create an organizer:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, in the analytic model, select the Organizers branch in the part browser.
3. Select **Part** > **New** > **Organizer.**

The Edit Part Name dialog box appears.

4. Enter a name for the organizer.
5. Click the **OK** button.

Related Links

Cube Collections

Entering Notes for an Analytic Model Definition's Parts

Use the notes bar to create notes for the analytic model definition or its parts. To create a note:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, in the part browser, select the analytic model definition or the part for which you want to create a note.
3. Click the notes bar.
4. Enter the note.

   You can also press the **Enter** key to create multiple paragraphs and click the **Cancel Changes** button to cancel the changes you made to the note.
5. Click the **Accept Changes** button to accept the changes you made to the note.

Finding Parts

You can select one or more parts and find all of the locations in the analytic model where the parts are used.

Note: The Find Part feature does not operate on organizers.

To find a part:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, select one or more parts in the part browser.
3. Select **Tools** > **Analytic Model** > **Find Part.**

   The locations of the parts are listed in the output window.

Validating Analytic Models

An important part of the analytic model creation process involves periodically validating the analytic model. The validate utility runs a series of tests on the analytic model and sends its results to the Validate tab in the output window. If errors are found, they are listed on this tab.

To validate an analytic model:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. Select **Tools** > **Analytic Model** > **Validate** after signing in to the PeopleSoft Application Designer.

Chapter 5

Creating Data Cubes

Understanding Data Cubes

This topic provides an overview of data cubes in cube collections.

Definition of a Data Cube

A data cube is a container for one kind of data that you use in cube collections.

You can place the same data cube in more than one cube collection. For example, you can place the EMPLOYEE_EXPENSE data cube in both an EMPLOYEE_ANALYSIS cube collection and an INCOME_STATEMENT cube collection. To populate the data cubes with data from the database, you map fields to the data cubes within the cube collection's properties.

See Mapping Data Cubes and Dimensions to Fields.

Within PeopleSoft Pure Internet Architecture pages with analytic grids, end users view cube collections and drag and drop data cubes to view their relationships to other data cubes.

You create four different types of data cubes that you use within an analytic model:

- Input data cubes.
- Calculation data cubes.
- Association data cubes.
- Virtual data cubes.

The four types of data cubes are not mutually exclusive, but certain combinational restrictions apply. For example, consider that all calculation data cubes contain formulas, and association data cubes may or may not contain formulas. When an association data cube does contain a formula, it is considered to be a type of calculation data cube. Similarly, when an input data cube contains a formula, it is also considered to be a type of calculation data cube. Any of these data cubes may also be considered virtual data cubes if their values are not stored in the database.

The following table lists each type of data cube and specifies whether the data cube can contain a formula, whether the data cube can lack a formula, whether the data cube can be virtual, and whether the data cube can be nonvirtual.

Data Cube Type	*Formula Allowed?*	*No Formula Allowed?*	*Can Be Virtual?*	*Can Be nonvirtual?*
Input	Yes **Note:** When input data cubes contain formulas, they must use the INPUT built-in function.	Yes	No	Yes
Calculation	Yes	No	Yes	Yes
Association	Yes	Yes	Yes	Yes
Virtual	Yes	No	Yes	No

Example: Working with Data Cubes and Dimensions

To be useful, a data cube must work with one or more dimensions. For example, suppose that you want to track the sales of multiple products in multiple regions. First, create an input data cube called SALES and dimensions called PRODUCTS and REGIONS. Next, attach the PRODUCTS dimension and REGIONS dimension to the SALES data cube.

Note: When a cube collection is mapped to either a Writable-only record or a record with the Readable and Writable attributes, all data cubes in the cube collection should share the same set of dimensions. The combined string of all the attached dimension names should not exceed 256 characters.

Related Links

Understanding Dimensions
Creating a New Dimension
Attaching a Dimension to a Data Cube
Data Cubes and Dimensions

Input Data Cubes

Input data cubes receive their data from either the end user in the application or tables and views in the database. Input data cubes can exist in all types of cube collections, although they do not serve a purpose in intermediate/calculation cube collections. Use the INPUT built-in function to work with input cube data.

Note: Even though an input cube that uses either the INPUT built-in function is considered to be a type of calculation data cube, it would not serve a purpose in an intermediate/calculation cube collection.

See INPUT.

Related Links

Creating Input Data Cubes
Understanding Types of Cube Collections

Calculation Data Cubes

Calculation data cubes contain formulas that calculate data based on the data of other cubes. Calculation data cubes can exist in all types of cube collections.

Note: Even though an input cube that uses the INPUT built-in function is considered to be a type of calculation data cube, it would not serve a purpose in an intermediate/calculation cube collection.

Related Links

Creating Calculation Data Cubes
Understanding Types of Cube Collections

Association Data Cubes

An association data cube is a data cube that is formatted as a member of a dimension and has one or more attached dimensions. An association data cube associates two dimensions, enabling the end user to group members of one dimension into categories that are defined by the members of a different dimension. When an association data cube receives its values from dimension members, it can be considered to be a type of input data cube. When an association data cube receives its values from a calculation formula, it can be considered to be a type of calculation data cube.

Association data cubes can exist in all types of cube collections.

Example: Creating the DEPT_TO_REGION Association Data Cube

This example associates the DEPTID dimension with the REGION dimension. The following table lists the members that are included in each dimension.

DEPTID Dimension Members	***REGION Dimension Members***
Note: In the application, the end users group or categorize these members by categories that are defined by the members of the REGION dimension.	**Note:** In the application, the end users select members from this dimension to group members of the DEPTID dimension.
AUS01	APAC
AUS02	LATAM
BRA01	NAMER
CAN01	EUROP

DEPTID Dimension Members	*REGION Dimension Members*
EUR01	N/A
GBR01	N/A
JAP01	N/A
JAP02	N/A
MEX01	N/A
USA01	N/A
USA02	N/A

This association enables the end user to group the members of the DEPTID dimension into categories that are defined by the members of the REGION dimension.

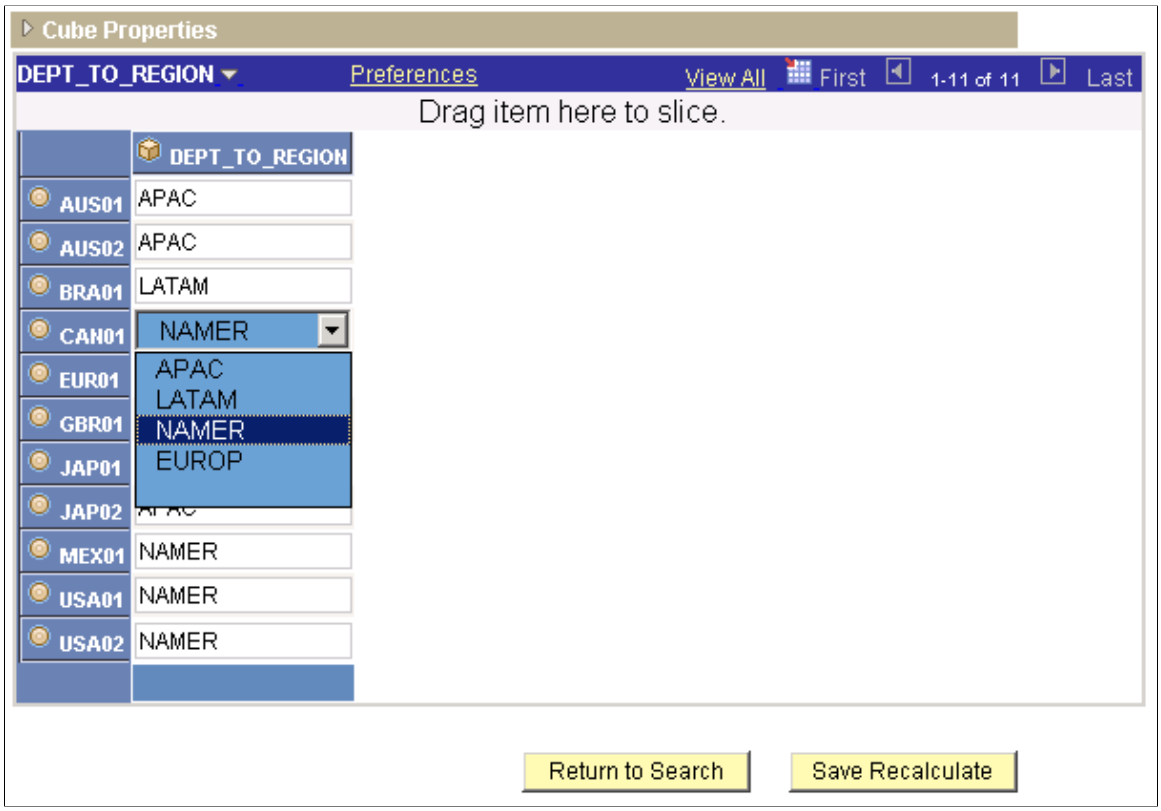
To create the DEPT_TO_REGION association data cube:

1. Create a new data cube named DEPT_TO_REGION.
2. Format the data cube as a member of the REGION dimension.

   This dimension contains the categories that the end user will use to group the members of the DEPTID dimension. These members appear in the right-hand column of the data cube's data. The end user can select these members from a drop-down list box.
3. Attach the DEPTID dimension to the DEPT_TO_REGION association data cube.

   This dimension contains the members that the end user will group or categorize. These members appear in the left-hand column of the data cube's data.

This example shows an association data cube and its drop-down list box in an analytic grid.



Related Links

Creating Association Data Cubes
Understanding Types of Cube Collections

Virtual Data Cubes

A virtual data cube is a type of calculation data cube whose values are not saved to the database. Virtual data cubes can exist in intermediate/calculation and presentation cube collections.

The following table describes the characteristics of virtual data cubes and the resulting benefits to the analytic model.

Characteristic	***Benefit***
Value data of virtual data cube is not stored in the database.	Reduces: • Size of the database. • Time to load data from the database.
The analytic calculation engine does not recalculate the virtual data cube unless the virtual data cube has nonvirtual dependents.	Reduces recalculation time.

Characteristic	*Benefit*
The analytic calculation engine neither allocates memory nor calculates virtual data cubes until it receives a request for recalculation of the virtual data cube.	Reduces memory consumption and recalculation time.

See Defining General Data Cube Properties.

When an end user loads an analytic instance, the underlying analytic model's virtual data cubes do not contain data. However, as soon as the analytic calculation engine receives a request for a virtual cube's data, the analytic calculation engine calculates the entire cube and places the totals and all nonzero values in a temporary storage area. After this point, if the application requires the data, the analytic calculation engine retrieves the data from the temporary storage area.

Virtual cube data is recalculated for these circumstances:

- The virtual data cube's data is displayed in an analytic grid.
- The virtual data cube is used during a step of a recalculation.
- The virtual data cube is accessed by a user function, even if the cube's data does not appear in the application.
- An application uses a PeopleCode program to request data from the virtual data cube.

Note: Whenever a circumstance requires a recalculation of all the data in an analytic model (for example, when the application adds a member to a dimension), the temporary storage for *all* virtual data cubes is discarded. This storage is created again as needed.

Virtual data cubes have the following two restrictions. Otherwise, you can use virtual data cubes in the way you use nonvirtual data cubes.

- Because a virtual data cube does not permanently store data, it must contain a formula to generate its data.

  Note: Deleting the formula for a virtual data cube results in an invalid analytic model.

- A virtual data cube cannot participate in recursive or circular systems because a virtual data cube's formula cannot refer to itself, either directly or indirectly.

  This restriction applies because the first time a virtual cube's data is requested, the analytic calculation engine calculates and stores the data for the entire virtual data cube. In recursive or circular systems, the analytic calculation engine cannot calculate all of the data at the same time for any given data cube.

  Note: If a virtual cube is part of a recursive or circular system, the analytic calculation engine generates an error value for all of the cube's values. Use the Recalculate function in the Analytic Calculation Engine classes to determine whether you violated this restriction. The Recalculate function returns a VIRTUAL error for the data cube cells that are affected.

  PeopleSoft recommends that you create virtual data cubes when you expect the cubes to be large, sparse, and output-only, especially when a relatively small slice of the ordinary cubes is used in any

given analytic instance ID. The analytic calculation engine takes a long time to recalculate nonvirtual cubes that are large, sparse, and output-only. When you make these cubes into virtual cubes, you eliminate them from the recalculation process and drastically reduce memory requirements. If an analytic instance uses only a small slice of the cube, the cube calculates on demand quickly and requires less memory because of the sparsity compression.

Virtual cubes are also useful for intermediate calculations that do not require permanent storage permanently, especially if these cubes would normally be large and sparse.

Note: You cannot use virtual cubes for intermediate calculations that are part of a recursive or circular system.
Do not create virtual cubes out of large, dense cubes that are displayed frequently and take a long time to recalculate. Such virtual cubes cause delays when an application requests data. To be certain of recalculation time, PeopleSoft recommends that you test whether using a virtual cube causes a significant delay in the generation of data.

See Understanding Circular Systems and Recursive Systems.

Intermediate virtual cubes can count as output-only cubes, as long as they do not have nonvirtual dependents. For example, you can create formulas such as the following for output-only virtual cubes:

- This formula is for the SALARY_BY_EMPLOYEE data cube:

```
GROUPSUM(RCD JOB, SALARY, BUDGET_PERIOD, BUS_UNIT, EMPID, LEDGER, VERSION)
```

- This formula is for the BENEFITS_BY_EMPLOYEE data cube:

```
GROUPSUM(RCD JOB, BENEFITS, BUDGET_PERIOD, BUS_UNIT, EMPID, LEDGER, VERSION)
```

- This formula is for the SALARY_AND_BENEFITS_BY_EMPLOYEE data cube:

```
SALARY_BY_EMPLOYEE + BENEFITS_BY_EMPLOYEE
```

Even though SALARY_BY_EMPLOYEE and BENEFITS_BY_EMPLOYEE are used by another virtual cube, they are not recalculated by the analytic calculation engine if there are no nonvirtual dependents. For this reason, you must write the final formula for the SALARY_AND_BENEFITS_BY_EMPLOYEE data cube in this way:

```
GROUPSUM(RCD_JOB, SALARY, BUDGET_PERIOD, BUS_UNIT, EMPID, LEDGER, VERSION) + GROUPS⇒
UM(RCD JOB, Benefits, Budget Period, Bus Unit, EmpID, Ledger, Version)
```

Related Links

Creating Virtual Data Cubes
Understanding Types of Cube Collections

Understanding the Relationship Between Field Definition Attributes and Data Cube Formats

Because data cubes receive data from fields, it is important to correctly set both the attributes of field definitions and the formats of data cubes to ensure compatibility.

The following table describes compatibilities between field definition attributes and data cube formats. Cells marked Yes indicate compatibility. Cells marked No indicate incompatibility. Cells marked Warn indicate potential compatibility and yield a warning during design time. During runtime, the analytic calculation engine generates an error if it determines that the mapping is not compatible.

Field Definition Attributes	***Data Cube Format: Number***	***Data Cube Format: Date***	***Data Cube Format: Member***	***Data Cube Format: Text***
Char	Warn	Warn	Yes	Yes
Number	Yes	No	Yes	Yes
Signed Number	Yes	No	Yes	Yes
Date	No	Yes	Yes	Yes
Time	No	No	No	No
Date Time	No	Warn **Note:** When a date-formatted data cube is mapped to a field with a Date Time attribute, time-specific data is truncated in the data cube data.	Yes	Yes
Image	No	No	No	No
Long Char	No	No	No	No

Related Links

Defining General Data Cube Properties

“Understanding Field Definitions” (Application Designer Developer’s Guide)

“Creating New Field Definitions” (Application Designer Developer’s Guide)

Creating Input Data Cubes

To create an input data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.

2. After signing in to the PeopleSoft Application Designer, with an analytic model definition opens, select **Part** > **New** > **Data Cube.**

   The Edit Part Name dialog box appears.

3. Enter the data cube name.

4. Click the **OK** button.

Note: Do not create formulas for input data cubes.

Related Links

Input Data Cubes

Creating Calculation Data Cubes

To create a calculation data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.

2. After signing in to the PeopleSoft Application Designer, with an analytic model definition opens, select **Part** > **New** > **Data Cube.**

   The Edit Part Name dialog box appears.

3. Enter the data cube name.

4. Click the **OK** button.

5. Create a formula for the calculation data cube.

   See Defining and Editing Data Cube Formulas.

Related Links

Calculation Data Cubes

Creating Association Data Cubes

To create an association data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.

2. After signing in to the PeopleSoft Application Designer, with an analytic model definition opens, select **Part** > **New** > **Data Cube.**

   The Edit Part Name dialog box appears.

3. Enter the data cube name.
4. Click the **OK** button.
5. Format the data cube as a member of a dimension.

   This dimension contains the members that the end user will group or categorize. In the application, these members appear in the left-hand column of the data cube's data.

   See Defining General Data Cube Properties.
6. Attach a different dimension to the data cube.

   This dimension contains the categories by which the end user will group the members of the X dimension. These members appear in the right-hand column of the data cube's data. The end user can select these members from a drop-down list box.

   See Attaching a Dimension to a Data Cube.

Related Links

Association Data Cubes

Creating Virtual Data Cubes

To create a virtual data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, with an analytic model definition opens, select **Part** > **New** > **Data Cube.**

   The Edit Part Name dialog box appears.
3. Enter the name of the data cube.
4. Click the **OK** button.
5. On the General tab of the data cube's properties, select the **Virtual Cube (doesn't store data)** check box.

Related Links

Virtual Data Cubes

Defining Data Cube Properties

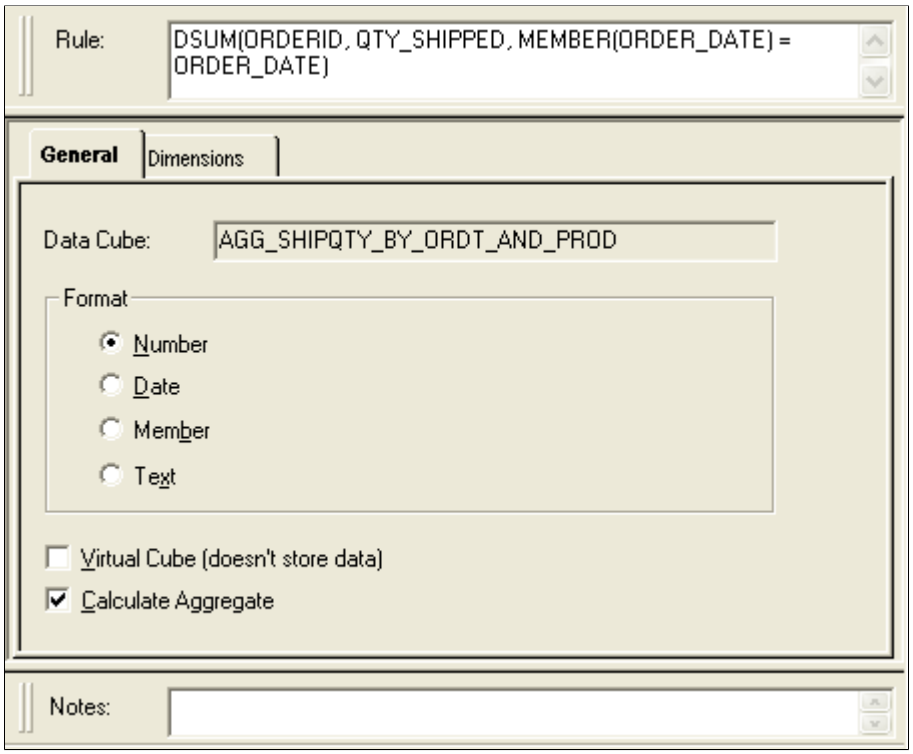
This topic provides an overview of defining data cube properties.

Defining General Data Cube Properties

To define general data cube properties:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose properties you want to define, and then select the General tab.

This is an example of the General tab in PeopleSoft Application Designer–Analytic Model.



Field or Control	*Description*
Data Cube	Displays the name of the data cube.

Field or Control	*Description*
Format	*Number:* Select to format the data cube's values as numbers. *Date:* Select to format the data cube's values as a date in the format YYYY-MM-DD. For example, 2004/03/18 for March 18, 2004. **Note:** Although the values are saved in the database using this date format, end users can use My Personalizations to select a different display format in PeopleSoft Pure Internet Architecture. *Member:* Select to format the data cube's values as members of a specified dimension, as part of creating an association data cube. See Association Data Cubes. When you select the **Member** check box, the **Dimension** drop-down list box appears. Select a dimension for which you want to format the data cube's values as members. For example, you can format a CUSTOMER_ID data cube as a member of the CUSTID dimension. **Note:** In the analytic grid, data cubes formatted as members should have a field type of *Edit Box.* *Text:* Select to format the data cube's values as text. This option is useful for entering names, addresses, and other textual data.
Virtual Cube (doesn't store data)	Select to set the data cube as a virtual data cube. Clear to set the data cube as a nonvirtual data cube. See Virtual Data Cubes. **Note:** A virtual data cube must contain a formula. Selecting this option without entering and accepting a formula for a virtual data cube results in an invalid analytic model.

Field or Control	*Description*
Calculate Aggregate	Select to enable calculation of the data cube's aggregates. **Note:** If **Calculate Aggregate** is selected for the data cube, the analytic calculation engine initially retrieves the aggregate data from the aggregate record when the analytic instance is loaded, but overwrites this data upon recalculation. If this check box is cleared, values from the aggregate record still load when the analytic instance is loaded; however, these values are not recalculated. Clear this check box to disable calculation of all of the data cube's aggregates, regardless of specified overrides. **Note:** Disabling aggregate calculation for data cubes disables all aggregate calculations, including the default sum aggregation. See Understanding Override Order of Precedence.

Related Links

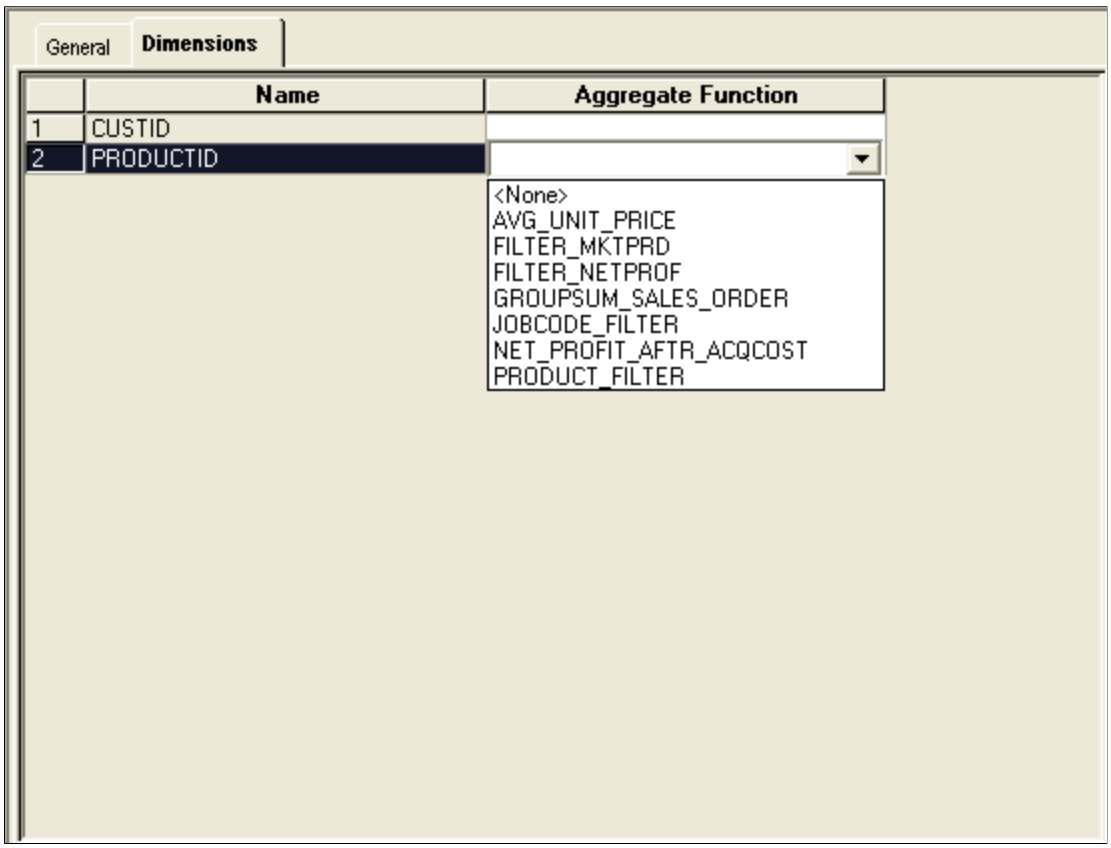
Understanding the Relationship Between Field Definition Attributes and Data Cube Formats

Selecting Aggregate Functions for Attached Dimensions

To select an aggregate function for attached dimensions:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube for which you want to select an aggregate function, and then select the Dimensions tab.

This is an example of the Dimensions tab in PeopleSoft Application Designer–Analytic Model.



Field or Control	*Description*
Name	Displays the names of the dimensions that are attached to the data cube. See Attaching a Dimension to a Data Cube.
Aggregate Function	Select a cube dimension override user function to calculate the aggregates for the dimension as it is attached to the data cube. See Understanding Override Order of Precedence.

Auditing Data Cubes at Design Time

This topic provides an overview of auditing data cubes at design time.

Note: Use the Analytic Model Viewer to audit cube collections and data cubes in runtime.

See Viewing and Debugging Cube Collection Properties.

Understanding Causes and Inputs

Any data cube that affects another data cube is a cause or precedent of that data cube. A data cube can be a direct cause or an indirect cause of another data cube. A direct cause is used in the data cube's formula. An indirect cause is not used in the formula, but it appears somewhere in the chain of formulas that ultimately affect the data cube.

For example, suppose the GROSS_MARGIN and NET_INCOME data cubes contain these formulas:

- Formula for the GROSS_MARGIN data cube:

  ```
  SALES - COST_OF_GOODS
  ```

- Formula for the NET_INCOME data cube:

  ```
  GROSS_MARGIN - TOTAL_EXPENSE
  ```

In this example, SALES is a direct cause of GROSS_MARGIN because it is used in GROSS_MARGIN's formula. SALES is an indirect cause of NET_INCOME because it affects GROSS_MARGIN, which in turn affects NET_INCOME.

You can display the causes of a data cube to view the assumptions behind a result or to find a formula that is not working properly.

Using the **All Inputs** option, you can also display all of the input data cubes that affect a data cube, either directly or indirectly.

Understanding Effects

Any data cube that is affected by another data cube is an effect or dependent of that data cube. A data cube can be a direct effect or an indirect effect of another data cube. A direct effect uses the data cube in its formula. An indirect effect does not use the data cube in its formula, but it is part of the chain of calculations that are affected by the data cube.

Again, suppose the GROSS_MARGIN and NET_INCOME data cubes contain these formulas:

- Formula for the GROSS_MARGIN data cube:

  ```
  SALES - COST_OF_GOODS
  ```

- Formula for the NET_INCOME data cube:

  ```
  GROSS_MARGIN - TOTAL_EXPENSE
  ```

GROSS_MARGIN is a direct effect of SALES because it uses SALES in its formula. NET_INCOME is an indirect effect of SALES because it is affected by GROSS_MARGIN, which in turn is affected by SALES.

You can display either the direct or direct plus indirect effects of a data cube to view the consequences of a data cube's values.

Displaying Causes and Inputs

To display the causes or inputs of a data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select a data cube whose causes or inputs you want to display.
   - To select several consecutive data cubes, hold down the **Shift** key and select the data cubes.
   - To select a series of nonconsecutive data cubes, hold down the **Ctrl** key and select the data cubes.
4. Select **Tools** > **Analytic Model** > **Causes.**
5. Select one of these options:
   - **Direct Causes.**
   - **All Causes.**
   - **All Inputs.**

     When applicable, the Causes and Effects dialog box displays the causes or inputs of the data cube. You expand any of the data cubes in the dialog box to view their attached dimensions.

     Note: The **All Inputs** option does not display the INPUT built-in function.
6. Click the **Close** button when you have finished viewing the causes or inputs.

Note: You can also display causes and inputs by selecting **Tools** > **Analytic Model** > **Causes and Effects Tool.**

Displaying Effects

To display the effects of a data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select a data cube whose effects you want to display.

   To select several data cubes, hold down the **Ctrl** key and select the data cubes.
4. Select **Tools** > **Analytic Model** > **Effects.**
5. Select one of these options:
   - **Direct Effects.**
   - **All Effects.**

The Causes and Effects dialog box displays either the direct effects or all (direct and indirect) effects of the data cube. You expand any of the data cubes in the dialog box to view their attached dimensions.

6. Click the **OK** button when you have finished viewing the effects.

Note: You can also display effects by selecting **Tools** > **Analytic Model** > **Causes and Effects Tool.**

Related Links

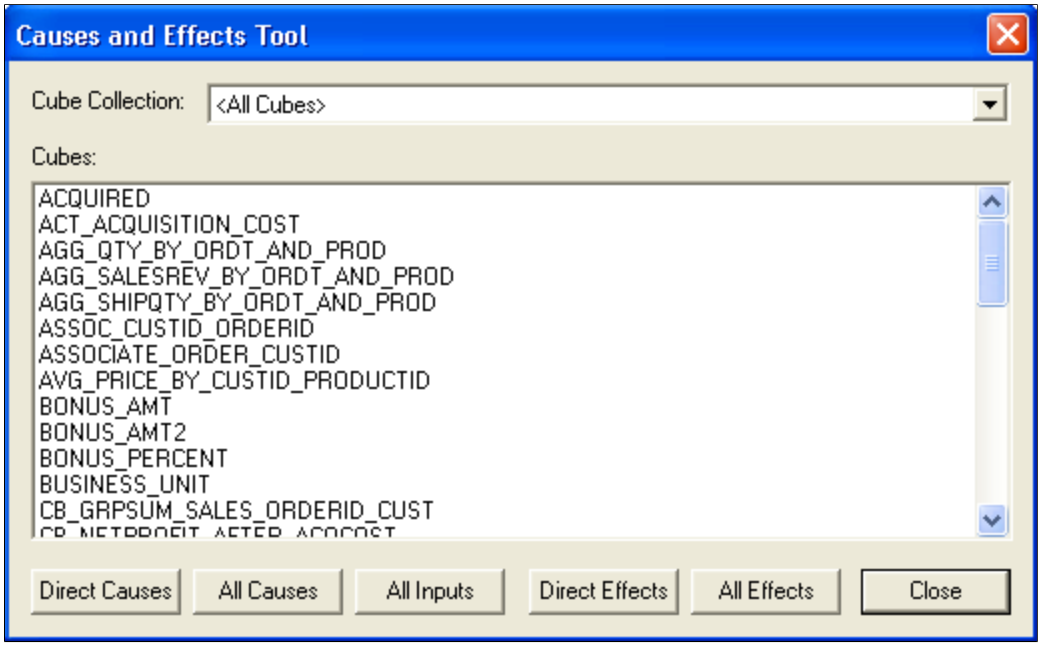
Using the Causes and Effects Tool

Using the Causes and Effects Tool

To use the Causes and Effects Tool option:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select **Tools** > **Analytic Model** > **Causes and Effects Tool** to browse through the cube collections and data cubes of your analytic model to view the causes, effects, and inputs of data cubes.

This is an example of the Causes and Effects Tool dialog box.



Field or Control	*Description*
Cube Collection	Select the cube collection to display a list of its data cubes. **Note:** You can also select <*All Cubes*> to display a list of all data cubes in the analytic model.
Cubes	Displays the names of the data cubes in the selected cube collection or the analytic model. Select a data cube to view its causes, inputs, or effects. To select several data cubes, hold down the **Ctrl** key and select the data cubes.

Note: You can also audit individual data cubes by selecting the data cube, and then selecting **Tools** > **Analytic Model** and the desired audit option from the menu bar.

See Displaying Causes and Inputs, Displaying Effects.

Chapter 6

Creating Dimensions

Understanding Dimensions

A dimension is a collection of people, places, events, or things for which you want to keep data. Each member of the dimension is called a dimension member.

See Understanding Dimension Members.

To keep data for each member of the dimension, attach the dimension to one or more data cubes. For example, to keep sales data for multiple products, attach a PRODUCTS dimension to a SALES data cube. To track the cost of goods for each product, attach the PRODUCTS dimension to a COST_OF_GOODS data cube.

Note: When a cube collection is mapped to either a Writable-only record or a record with the Readable and Writable attributes, all data cubes in the cube collection should share the same set of dimensions.

Because dimensions receive data from fields, it is important to correctly set the attributes of field definitions to ensure compatibility. You can map fields with the following attributes to dimensions:

- Char
- Number
- Signed Number
- Date
- Date Time

Note: The limit on total number of dimensions in a model is 99.

Related Links

Data Cubes and Dimensions

Creating a New Dimension

To create a new dimension:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select **Part** > **New** > **Dimension.**

The Edit Part Name dialog box appears.

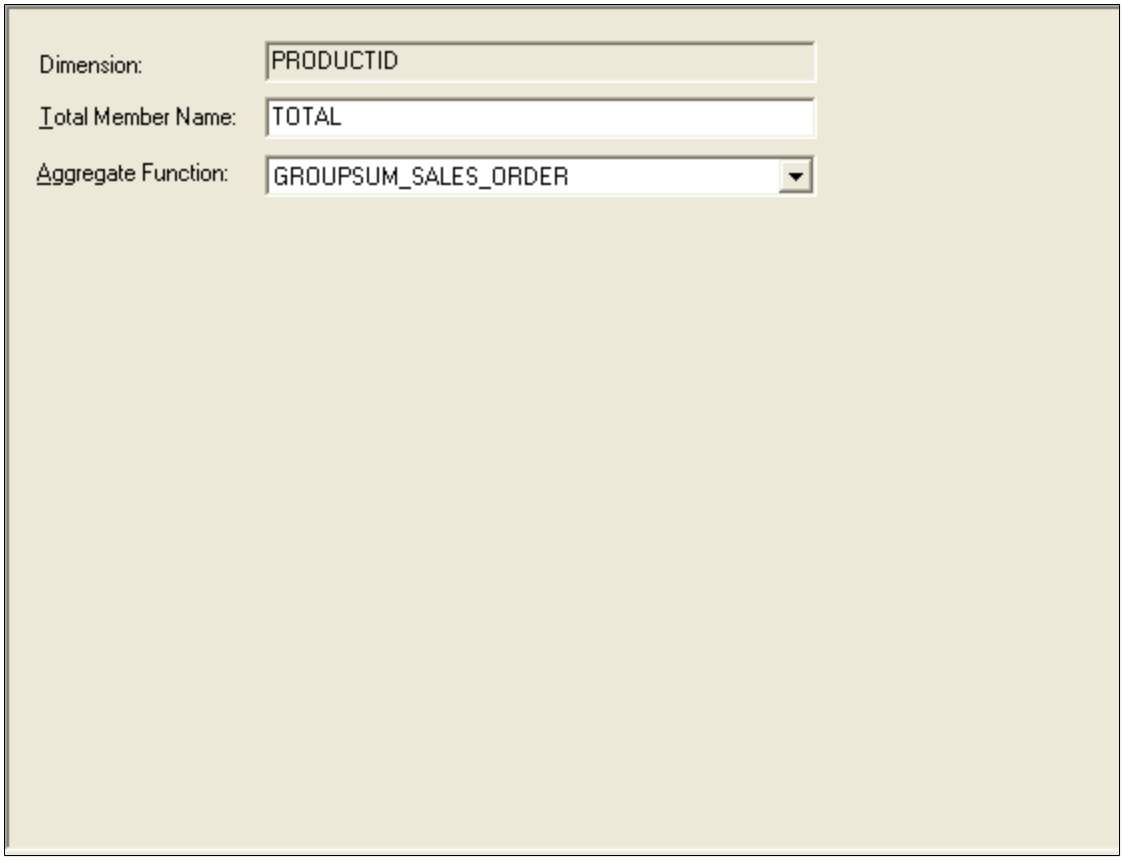
4. Enter the dimension name.
5. Click the **OK** button.

Defining Dimension Properties

To define dimension properties:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the dimension whose properties you want to define.

This is an example of the dimension properties.



Field or Control	*Description*
Total Member Name	This field performs different functions depending upon whether you have entered a value into the **Total Member Name** field, whether the dimension belongs to an analytic model that was converted from a BAM 8.8 model, and whether a tree is attached to the dimension.
Aggregate Function	Select a dimension override function to calculate the dimension's aggregate fields. The analytic calculation engine uses this aggregate function to calculate all of a dimension's aggregates. **Note:** This aggregate function does not apply to leaf members or detail members. See Understanding Override Order of Precedence, Example: Creating a Hierarchy with Mixed Aggregate and Detail Members.

Interpreting the Total Member Name Field

The following table describes the state of the dimension based upon whether:

- A tree is attached to the dimension.
- The **Total Member Name** field has a value.

  Values for the **Total Member Name** field come from either the value that you entered into the **Total Member Name** field of the dimension's properties, a converted BAM 8.8 model, or a PeopleCode command using the Analytic Calculation Engine Metadata Classes.

Tree Attached to Dimension?	*Has Value?*	*Resulting State of Dimension*
Yes	No	If you select the **Calculate Aggregates** check box, the value of hierarchy root member appears to the end user and to the analytic calculation engine.

Tree Attached to Dimension?	*Has Value?*	*Resulting State of Dimension*
No	Yes	Analytic Calculation Engine creates a basic, one-node hierarchy for the dimension. By default, the Show Hierarchy method is used on the analytic model, and both the one-node hierarchy and the name that you enter into the **Total Member Name** field appear to the end user. **Note:** For the actual value of the hierarchy to appear to the end user, you must select the **Calculate Aggregates** check box. If you do not select this check box, the value of *0* appears to the end user.
Yes	Yes	One of these resulting states applies: • If you apply an aggregate function to this dimension, the value that you enter into the **Total Member Name** field serves as an alias to the hierarchy root member. This alias only appears to the user functions within the analytic model; the actual value of the hierarchy root member appears to the end user. • If you do not apply an aggregate function to this dimension and select the **Calculate Aggregates** check box, the aggregate value of the hierarchy root member appears to the end user.
No	No	Analytic Calculation Engine does not create a hierarchy for the dimension.

Attaching a Dimension to a Data Cube

To be useful, a dimension must work with one or more data cubes.

Note: When a cube collection is mapped to either a Writable-only record or a record with the Readable and Writable attributes, all data cubes in the cube collection should share the same set of dimensions.

To attach a dimension to a data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Right-click the data cube to which you want to attach the dimension.
4. Select the **Attach Dimensions** option.

   The Attach Dimension(s) to Selected Cube(s) dialog box appears.
5. Select one or more dimensions.
   - Press the **Ctrl** key and click the left mouse button to select multiple dimensions.
   - Click the **Select All** button to select all of the dimensions.
   - Click the **Unselect All** button to clear all of the dimensions.
   - Click the **OK** button.

Related Links

Data Cubes and Dimensions

Changing the Order of Dimensions in the Part Browser

To change the order of dimensions in the part browser:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select one dimension that you want to move up or down in relationship to the other dimensions in the part browser.
4. Perform one of these steps:
   - Right-click the dimension member and select the *Move Up* or the *Move Down* option.
   - Hold down the left mouse button, drag the dimension to the desired location, and release the left mouse button.
5. Perform steps 1 and 2 until all of the dimensions appear in the desired order in the part browser.

Related Links

Dimension Order Impact on Calculation

Chapter 7

Creating Cube Collections

Understanding Cube Collections

A cube collection is a collection of related data cubes. You create cube collections to load data from the database into the analytic model, receive user input, persist data back to the database, and display calculated data to the end user at runtime.

When the end user loads an analytic instance, Analytic Calculation Engine loads the data from the database into the data cubes of the analytic model. These data cubes exist within cube collections. You map the main record from the database to the cube collection, and the main record's fields to the data cubes and dimensions within the cube collection.

After loading an analytic instance, the end user has access to one or more cube collections within the application. These cube collections which are displayed in PeopleSoft Pure Internet Architecture pages with analytic grids contain the data that end users can view or edit. You create these pages in PeopleSoft Application Designer.

See Understanding Analytic Grid Design.

Note: The analytic model may contain cube collections that are not visible to the end user.

For each record that you want to work with in the analytic model, you generally create read/write cube collection to load data into the analytic model, and a presentation cube collection for end user input, reporting and forecasting. The presentation cube collection calculates the data from the read/write cube collection's data cubes and displays the calculated data to the end user. Presentation cube collections may also receive end user input.

You can also create an intermediate/calculation cube collection to organize data cubes and create calculations whose results are not displayed to the end user.

In the General tab of the cube collection's properties, you map the main record, which stores the fact data that you want to load and persist. For a read/write cube collection, select one of the records that you selected in the analytic type definition that corresponds to the analytic model. For a presentation cube collection, select a derived/work record from the list of available records.

Note: Do not map intermediate/calculation cube collections to any records.
In the analytic type definition, you do not have to select the derived/work records that you want to map to presentation cube collections. However, the analytic type definition must include all derived/work records that are mapped to cube collections on which you use the NetChanges parameter of the GetData CubeCollection class method.
When a cube collection is mapped to either a Writable-only record or a record with the Readable and Writable attributes, all data cubes in the cube collection should share the same set of dimensions.

If desired, you can also use the General tab (in PeopleSoft Application Designer–Cube Collections) to map the cube collection to an aggregate record to persist the cube collection's aggregate data.

After you map the cube collection to the main and aggregate records, use the Field Map tab (in PeopleSoft Application Designer–Cube Collections) to map the cube collection's data cubes and dimensions to the fields of the main and aggregate records.

Note: You can map a data cube or dimension to one field within one record. After you have mapped a data cube or dimension to a particular record field, you cannot use that record field in another mapping. You can, however, reuse the same field if that field is from a different record.

Use the Dimensions tab (in PeopleSoft Application Designer–Cube Collections) to define these additional attributes for the dimensions in the cube collection:

- How much aggregate data is saved.
- Sort order.
- Filter user function.

Understanding Types of Cube Collections

You create three different types of cube collections in an analytic model. This topic discusses:

- Read/write cube collections.
- Intermediate/calculation cube collections.

  Note: Intermediate/calculation cube collections are optional.

- Presentation cube collections.

Read/Write Cube Collections

Use read/write cube collections to load data from the database, receive user input, and persist data back to the database. For the main record, you can select any record type except derived/work records.

Note: The analytic type that you use with the analytic model must contain the records that you map to read/write cube collections.

See Purpose of Analytic Type Definitions, Relationship of Record Attributes to Data Caching Behavior.

The following table indicates which data cube types are allowed in a read/write cube collection.

Data Cube Type	***Allowed in Read/Write Cube Collection?***
Input data cubes	Yes

Data Cube Type	*Allowed in Read/Write Cube Collection?*
Calculation data cubes	Yes **Note:** Initial data values for calculation data cubes are loaded from the database. Calculated values are written back to the database.
Association data cubes	Yes
Virtual data cubes	No

See Input Data Cubes.

Intermediate/Calculation Cube Collections

Use intermediate/calculation cube collections to organize data cubes and create intermediate calculations in an analytic model. These intermediate values are neither displayed to the end user nor persisted to the database. For this reason, do not map a main record to an intermediate/calculation cube collection. You can view intermediate/calculation cube collections in the Analytic Model Viewer.

Note: Intermediate/calculation cube collections are optional.

The following table indicates which data cube types are allowed in an intermediate/calculation cube collection.

Data Cube Type	*Allowed in Intermediate/Calculation Cube Collection?*
Input data cubes	Yes **Note:** Although input data cubes are allowed in calculation cube collections, their data is not updated. For this reason, input data cubes serve no purpose in intermediate/calculation cube collections.
Calculation data cubes	Yes
Association data cubes	Yes
Virtual data cubes	Yes

See Input Data Cubes.

Presentation Cube Collections

Use presentation cube collections to present data to the end user for the purposes of reporting and forecasting. For forecasting purposes, end users may also enter data into presentation cube collections. You can select only a derived/work record as the main record of a presentation cube collection. If you select any other type of record, you will not be able to select the cube collection on the Analytics tab of the analytic grid.

Note: You must create the derived/work record before selecting it as the main record.

See “Creating New Record Definitions” (Application Designer Developer’s Guide).

Use the GetData and SetData methods to transfer data between presentation cube collections and the application server.

The following table indicates which data cube types are allowed in a presentation cube collection.

Data Cube Type	***Allowed in Presentation Cube Collection?***
Input data cubes	Yes
Calculation data cubes	Yes
Association data cubes	Yes
Virtual data cubes	Yes

See Input Data Cubes.

Example: Creating Two Cube Collections

Suppose you want to work with sales data in an analytic model. Create these cube collections:

- SALES_RW read/write cube collection.

  On the General tab, map this cube collection to the SALES main record to receive and persist raw sales data. This record must exist in the analytic type definition. This cube collection contains these data cubes:

  - UNIT_COST data cube.

    Map this data cube to the UNIT_COST field.

  - UNIT_SOLD data cube.

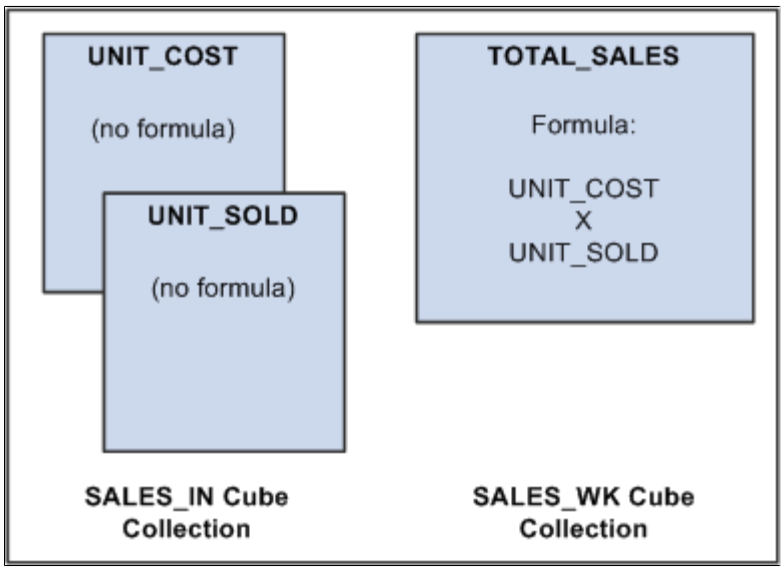
    Map this data cube to the UNIT_SOLD field.

- SALES_PRES presentation cube collection.

Map this cube collection to the SALES_WK derived/work record to calculate sales data and display the calculated data to the end user at runtime. This cube collection contains the TOTAL_SALES data cube, which is mapped to the TOTAL_SALES_WK field. The TOTAL_SALES data cube contains this formula:

```
UNIT_COST * UNIT_SOLD
```

This diagram provides a visual representation of these cube collections.



Creating Cube Collections

To create a cube collection:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select **Part** > **New** > **Cube Collection.**

   The Edit Part Name dialog box appears.
4. Enter the name of the cube collection.
5. Click the **OK** button.

   You can now drag and drop the desired data cubes and dimensions into the cube collection.

Related Links

Understanding Types of Cube Collections

Defining Cube Collection Properties

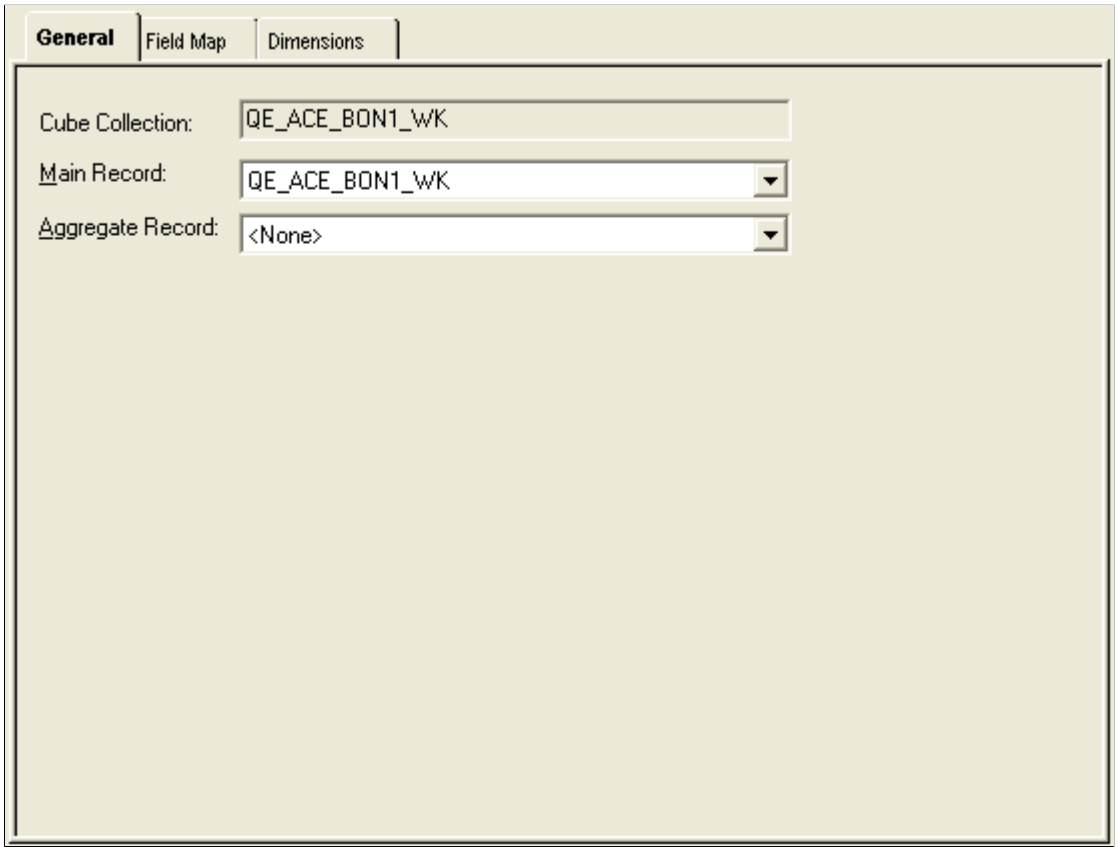
This topic discusses how to define cube collection properties.

Mapping a Cube Collection to Main and Aggregate Records

To map a cube collection to Main and Aggregate records:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the cube collection that contains the main and aggregate records that you want to map, and then select the General tab.

This is an example of the Cube Collections - General tab.



Field or Control	*Description*
Cube Collection	Displays the name of the cube collection.
Description	Enter a more detailed description of the cube collection.

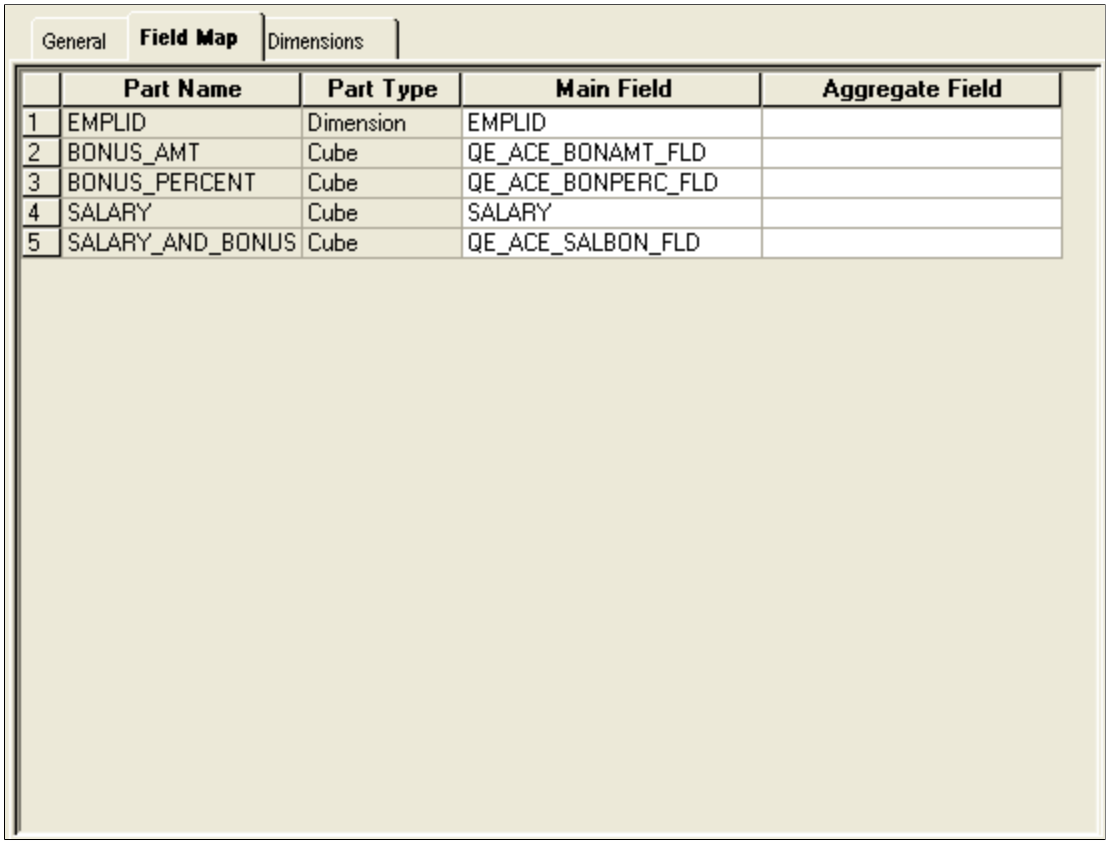
Field or Control	*Description*
Main Record	Select a main record to map to the cube collection. For a cube collection that is used for user input and data retrieval, select one of the records that you selected in the analytic type definition to use in the analytic model. For a cube collection that is used to calculate data and display the calculated data to the end user at runtime, select a derived/work record.
Aggregate Record	Select a record to store the cube collection's aggregate data. Records that are used as aggregate records should be read after records that are used as main records. See Synchronization Order.

Mapping Data Cubes and Dimensions to Fields

To map data cubes and dimensions to fields:

1. Access PeopleSoft Application Designer by selecting **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer**.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the cube collection that contains the data cubes and dimensions that you want to map, and then select the Field Map tab.

This is an example of the Cube Collections - Field Map tab.



Field or Control	*Description*
Part Name	Displays the name of the data cube or dimension to which you map fields. **Note:** You can map a field to only one data cube or dimension.
Part Type	Displays whether the part to which you map fields is a data cube or dimension.

Field or Control	*Description*
Main Field	Select a main field to map to the data cube or dimension. **Note:** You can map a data cube or dimension to one field within one record. After you have mapped a data cube or dimension to a particular record field, you cannot use that record field in another mapping. You can, however, reuse the same field if that field is from a different record. When mapping dimensions and data cubes, you may want to map dimensions to the key fields in the main record and data cubes to the non-key fields in the main record. The PeopleSoft system, however, does enable you to map dimensions to non-key and data cubes to key fields. To perform the most appropriate mapping, you must have a deeper understanding of the relationship between data cubes and dimensions. See Data Cubes and Dimensions.
Aggregate Field	Select a field to store the cube collection's aggregate data.

Defining Additional Cube Collection Dimension Properties

This topic discusses how to:

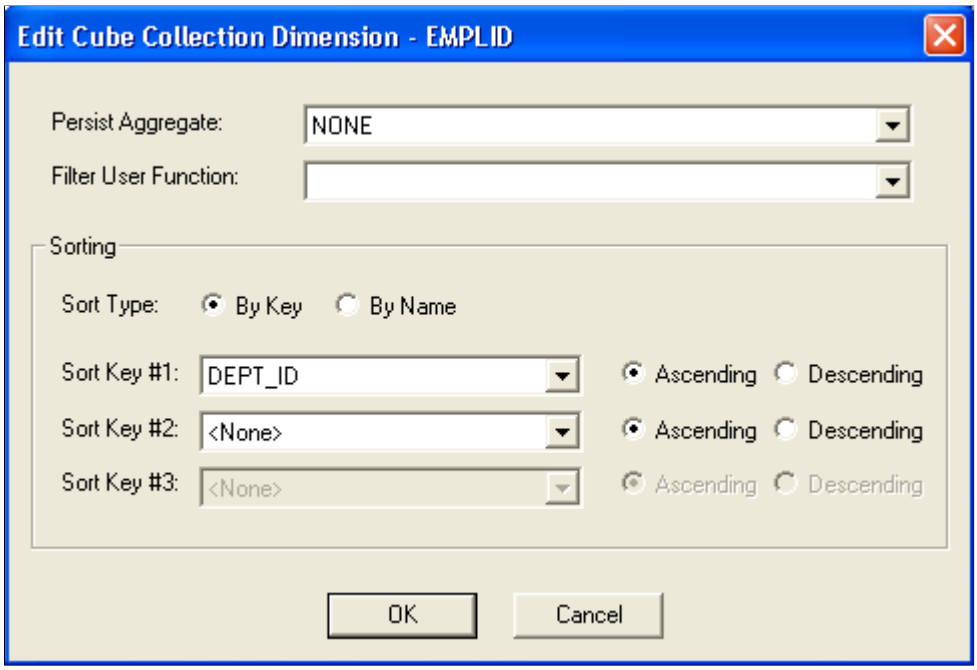
- Set additional cube collection dimension properties.
- View additional cube collection dimension properties.

Setting Additional Cube Collection Dimension Properties

To set additional cube collection dimension properties:

1. Access PeopleSoft Application Designer by selecting **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer**.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the cube collection that contains the dimensions whose properties you want to set, and then select the Dimensions tab.
4. Double-click any of the cells in the row of a dimension to access the Edit Cube Collection Dimension dialog box.

This is an example of the Edit Cube Collection Dimension dialog box.



Field or Control	***Description***
Persist Aggregate	Select whether to persist the dimension's aggregate values to the database. Aggregate data is persisted to the aggregate record that you select on the General tab. • *ALL:* Select to persist all of the dimension member aggregate values to the database. • *NONE:* Select to persist none of the dimension member aggregate values to the database. • *ROOT:* Select to persist only the value of the hierarchy root member to the database. See Understanding the Persistence of Aggregate Data.
Filter User Function	Select a filter user function to apply to the dimension. Select *None* if you do not want to apply a filter user function to the dimension. See Rules, Formulas, and User Functions.

Field or Control	*Description*
Sort Type	**By Name:** Select to sort the dimension member values by name. **By Key:** Select to sort the dimension member values by data cube values. **Note:** You can only select from the fields that are mapped to data cubes.
Sort Key #1	If the **By Key** option is selected, select the first data cube name by which you would like to sort the dimension member values. Select to sort the dimension's key values in ascending or descending order.
Sort Key #2	If the **By Key** option is selected, select the second data cube name by which you would like to sort the dimension member values. Select to sort the dimension's key values in ascending or descending order.
Sort Key #3	If the **By Key** option is selected, select the third data cube name by which you would like to sort the dimension member values. Select to sort the dimension's key values in ascending or descending order.

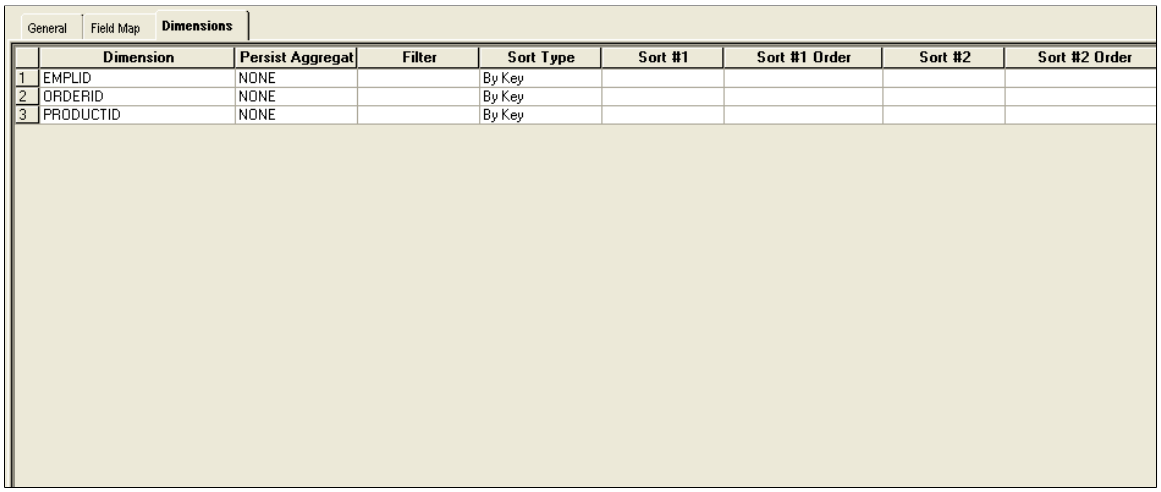
Viewing Additional Cube Collection Dimension Properties

The Dimensions tab displays additional properties that you have set for the dimensions in a cube collection.

To view additional cube collection dimension properties:

1. Access PeopleSoft Application Designer by selecting **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer**.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the cube collection that contains the dimensions whose properties you want to view, and then select the Dimensions tab.

This is an example of the Cube Collections - Dimensions tab.



Field or Control	*Description*
Dimension	Displays the name of the dimension.
Persist Aggregate	Displays any aggregates that are persisted for the dimension.
Filter	Displays the filter formula that is applied to the dimension.
Sort Type	Displays the sort type that is applied to the dimension's values.
Sort #1	If the dimension values are sorted by key, displays the first key field by which the dimension member values are sorted.
Sort #1 Order	If the dimension member values are sorted by the first key, displays whether the sort is by ascending or descending order.
Sort #2	If the dimension member values are sorted by key, displays the second key field by which the dimension member values are sorted.
Sort #2 Order	If the dimension member values are sorted by a second key, displays whether the sort is by ascending or descending order.
Sort #3	If the dimension member values are sorted by key, displays the third key field by which the dimension member values are sorted.

Field or Control	*Description*
Sort #3 Order	If the dimension member values are sorted by key, displays whether the sort is by ascending or descending order.

Chapter 8

Creating Explicit Dimension Sets

Understanding Explicit Dimension Sets

You create an explicit dimension set to form a distinct group of an analytic model's dimensions. Explicit dimension sets improve the calculation efficiency of multidimensional data cubes. You can create one or more explicit dimension sets within an analytic model. Explicit dimension sets may contain completely different or some of the same dimensions as one another. The analytic calculation set instantiates valid combinations of members called explicit tuples from explicit dimension sets.

See Understanding Implicit Tuples and Explicit Tuples.

A model can contain explicit dimension supersets and explicit dimension subsets. An explicit dimension superset is a set of dimensions that contains the same dimensions as its subset; however, the superset contains one or more dimensions than its subset. A subset is the inverse of a superset: it contains the same dimensions as its superset; however, the subset contains one or more fewer dimensions than its superset. An analytic model can contain multiple supersets and subsets.

Explicit dimension sets are applied to individual data cubes when calculating data, and are also used for exporting data. To determine which explicit dimension set it uses on a data cube, the analytic calculation engine first reads which dimensions are attached to the data cube, and then analyzes and iterates through the explicit dimension sets in priority order. The set at the lowest numbered row in the Explicit Dimension Sets dialog box receives the highest priority.

See Editing or Adding New Explicit Dimension Sets.

Note: Explicit dimension subsets must exist at a lower priority than their supersets. However, it is not necessary for an explicit dimension subset to exist in the row directly beneath its superset. For example, suppose that a superset exists in row 1. Its subset can be in row 3; it does not have to be in row 2. The explicit dimension set in row 2 can contain dimensions that are not included in other explicit dimension sets.

Using the priority order, the analytic calculation engine matches the first explicit dimension set that has either the same or fewer dimensions as are attached to the data cube, and then continues down the priority order for the remaining dimensions. The analytic calculation engine uses single dimensions if any of these dimensions remain unmatched after it has iterated through all explicit dimension sets.

For example, suppose these dimensions are attached to the OVERHEAD_COSTS data cube:

- CHANNELS
- CUSTOMERS
- TAXES
- EXPENSES

- MONTHS
- PRODUCTS
- REGIONS

The analytic model contains these explicit dimension sets:

- SET_1: MONTHS, REGIONS (first priority)
- SET_2: TAXES, PRODUCTS (second priority)
- SET_3: CHANNELS, CUSTOMERS, PRODUCTS (third priority)
- SET_4: CHANNELS, CUSTOMERS (fourth priority)
- SET_5: TAXES, EXPENSES (fifth priority)

The analytic calculation engine iterates through the explicit dimension sets, beginning with SET_1, and then continues down the priority order for the remaining dimensions, matching the following:

- SET_1: MONTHS, REGIONS (first priority)
- SET_2: TAXES, PRODUCTS (second priority)
- SET_4: CHANNELS, CUSTOMERS (fourth priority)

Example 1: Explicit Dimension Sets

The following table lists an analytic model's data cubes and their attached dimensions.

Data Cubes	***Data Cube Formula (if any)***	***Attached Dimensions***
SALES	UNIT_SALES * UNIT_PRICE	• PRODUCTS • REGIONS • MONTHS
UNIT_SALES	None (input cube)	• PRODUCTS • REGIONS • MONTHS
UNIT_PRICE	None (input cube)	• PRODUCTS • REGIONS • MONTHS

The company:

- Sells the hamburgers product in both the Africa and Asia regions during all months of the year.

- Sells the french fries product in the Africa region only during the summer months.
- Never sells the french fries product in the Asia region.

If you do not create an explicit dimension set to calculate these data cubes, the analytic calculation engine calculates the sales for all products in all regions during all months, even if some of these combinations are not valid. In other words, the analytic calculation engine calculates the sales for french fries for all months in Asia, even though the company does not sell french fries in Asia. Additionally, the analytic calculation engine calculates the sales for french fries during all months in Africa, even though the company only sells french fries in Africa during the summer.

In total, without using an explicit dimension set, the analytic calculation engine performs 48 calculations for the SALES data cube:

```
(2 PRODUCTS * 2 REGIONS * 12 MONTHS)
```

The analytic calculation engine generates the value of *0* for each invalid member combination, taking valuable time to do so. These invalid member values are:

- Not saved to the main record.
- Not displayed to end users in the application.

  End users view invalid member combinations as blank cells.

To prevent this unneeded calculation of data, you should create an explicit dimension set consisting of the PRODUCTS, REGIONS, and MONTHS dimensions. The analytic calculation engine uses this explicit dimension set, plus the UNIT_SALES and UNIT_PRICE input data cubes, to determine the explicit tuples. Using these tuples, the analytic calculation engine calculates only the necessary values.

Note: The analytic calculation engine uses the input cubes that comprise the relevant data cube's formula to instantiate the explicit tuples that it uses to calculate the data cube.

In total, the analytic calculation engine performs 27 calculations for the SALES data cube:

```
(1 PRODUCT * 2 REGIONS * 12 MONTHS) + (1 PRODUCT * 1 REGIONS * 3 MONTHS)
```

Example 2: Explicit Dimension Supersets and Subsets

An analytic model contains the SALES data cube. These dimensions are attached to this data cube:

- PRODUCTS
- REGIONS
- ORDER_ID
- BUSINESS_UNIT
- DEPT_ID
- EMPL_ID

You have defined these explicit dimension sets:

- SET_1: PRODUCTS, REGIONS, ORDER_ID (Priority 1).

- SET_2: PRODUCTS, REGIONS (Priority 2).
- SET_3: REGIONS, ORDER_ID (Priority 3).
- SET_4: DEPT_ID, EMPL_ID (Priority 4).

The explicit dimension subset (PRODUCTS, REGIONS) exists at a lower priority than its superset (PRODUCTS, REGIONS, ORDER_ID). The subset is in row 2; the superset is in row 1. The analytic calculation engine takes the cross product of the following to instantiate the valid combinations of dimension members for the SALES data cube:

- SET_1: PRODUCTS, REGIONS, ORDER_ID.
- SET_4: DEPT_ID, EMPL_ID.
- BUSINESS_UNIT dimension.

Related Links

Understanding the Calculation of Aggregate Data

Understanding Implicit Tuples and Explicit Tuples

Implicit tuples are the combinations of members that are used to calculate a single data cube but do not comprise an explicit dimension set.

Explicit tuples are the valid combinations of members that are instantiated from an explicit dimension set and are instantiated from these sources:

- Data loaded from the database.
- Data loaded by using the SetData method.
- Other explicit tuples in explicit dimension supersets.

Example: Implicit Tuples

This example illustrates the field to which the BUSINESS_UNIT dimension is mapped. Empty cells indicate null values.

BUSINESS_UNIT field
US001
US002

This example illustrates the field to which the DEPARTMENT dimension is mapped.

DEPARTMENT field
DEPT1000
DEPT2000
DEPT3000
DEPT4000

In this example, departments 1000 and 2000 exist only in business unit US001, and departments 3000 and 4000 exist only in business unit US002.

If the BUSINESS_UNIT and DEPARTMENT dimensions do not comprise an explicit dimension set and both dimensions are attached to the SALES data cube, the analytic calculation engine uses these implicit tuples to calculate the SALES data cube:

```
(US001, DEPT1000),
(US001, DEPT2000),
(US001, DEPT3000),
(US001, DEPT4000),
(US002, DEPT1000),
(US002, DEPT2000),
(US002, DEPT3000),
(US002, DEPT4000)
```

Example: Explicit Tuples

This example uses the same fields as the implicit tuples example.

If you create an explicit dimension set that includes the BUSINESS_UNIT and DEPARTMENT dimensions, the analytic calculation engine uses these explicit tuples to calculate the SALES data cube:

```
(US001, DEPT1000),
(US001, DEPT2000),
(US002, DEPT3000),
(US002, DEPT4000)
```

Note: The analytic calculation engine also uses the input cubes that comprise the relevant data cube's formula to instantiate the explicit tuples that it uses to calculate the data cube.

Defining Explicit Dimension Sets

This topic discusses how to define explicit dimension sets.

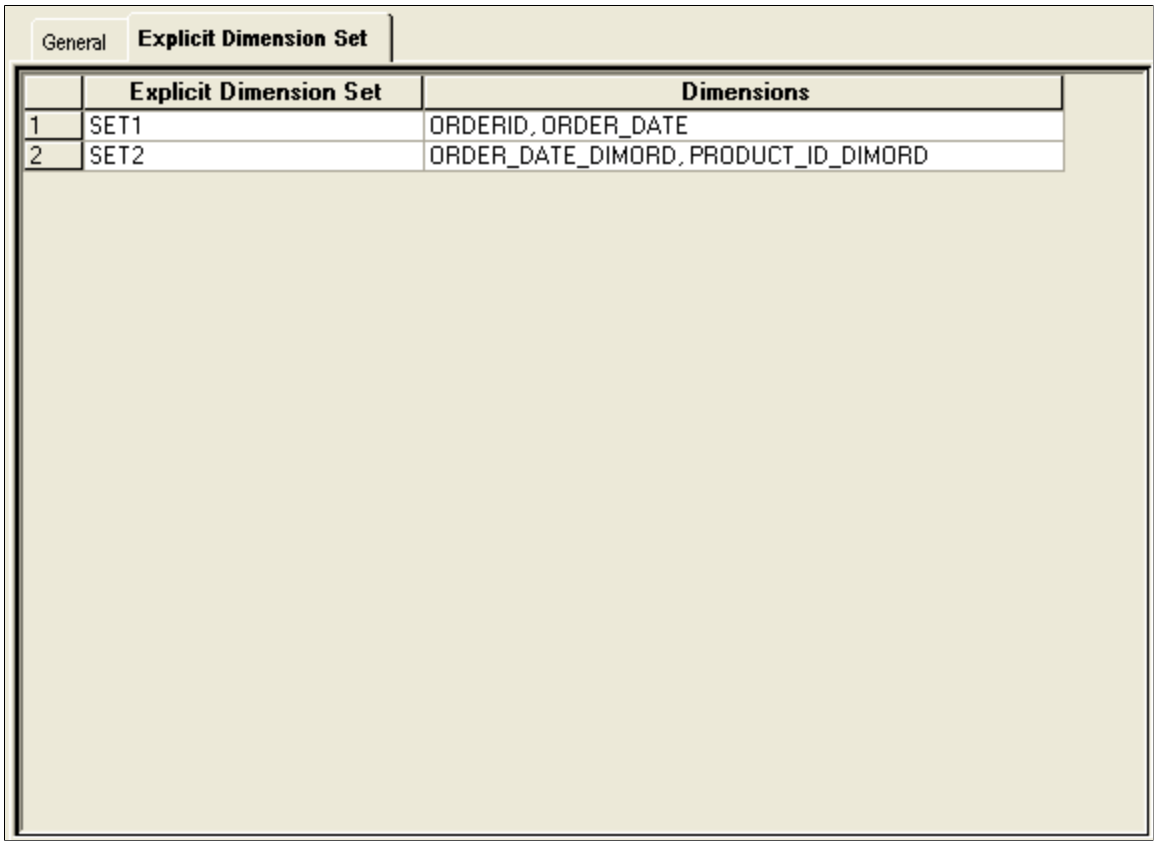
Viewing Explicit Dimension Set Properties

To view explicit dimension set properties:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.

2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Double-click the Parts branch in the part browser, and then select the Explicit Dimension Set tab.

This is an example of the Explicit Dimension Set tab.



Field or Control	*Description*
Explicit Dimension Set	Displays the name of the explicit dimension set.
Dimensions	Displays the dimensions that are included in the explicit dimension set.

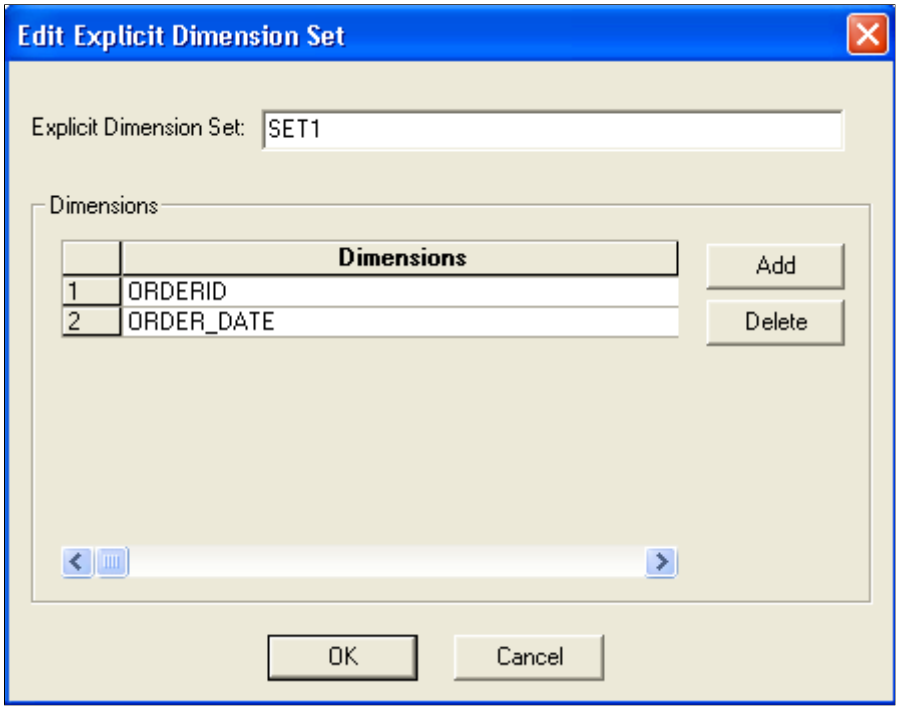
Editing or Adding New Explicit Dimension Sets

To edit or add new explicit dimension sets, perform one of these actions:

- To edit a preexisting explicit dimension set, double-click the name of the explicit dimension set on the Explicit Dimension Set tab.
- To add a new explicit dimension set to an analytic model that does not yet contain any explicit dimension sets, right-click the area at the bottom portion of the Explicit Dimension Set tab and select the **Add Dimension Set** option.

In either case, the Edit Explicit Dimension Set dialog box appears.

This example illustrates the Edit Explicit Dimension Set dialog box.



Field or Control	***Description***
Explicit Dimension Set	Enter or edit the name of the explicit dimension set.
Dimensions	Displays the dimensions in the explicit dimension set.
Add	If a row is currently selected, click this button to add a blank row beneath the currently selected row. You can then click the blank row to select a new dimension to add to the set. If a row is not currently selected, click this button to add a blank row to the bottom of the set. You can then click the blank row to add a new dimension from the resulting drop-down list box. **Note:** Explicit dimension subsets must exist at a lower priority than their supersets. However, it is not necessary for an explicit dimension subset to exist in the row directly beneath its superset. For example, if a superset exists in row 1, its subset can be in row 3 (it does not have to be in row 2). The explicit dimension set in row 2 can contain dimensions that are not included in other explicit dimension sets.
Delete	Delete a blank row or dimension from the explicit dimension set. **Note:** You must select a row before clicking the **Delete** button.

Chapter 9

Creating Hierarchies

Understanding the Relationship of PeopleSoft Trees to Analytic Models

This topic discusses the relationship of PeopleSoft trees to analytic models.

Note: This topic provides an overview of the relationship of PeopleSoft trees to analytic models, Business Analysis Modeler (BAM) total members, dimension members, the calculation of aggregate data, and the persistence of aggregate data, and discuss how to work with overrides.

Purpose of PeopleSoft Trees and Analytic Model Hierarchies

Analytic Calculation Engine uses trees to establish hierarchies of a dimension's parent-child relationships. Analytic Calculation Engine uses these hierarchies to:

- Calculate and display aggregated data to end users.
- Enable end users to navigate through data by performing such actions as expanding and collapsing nodes.
- Enable end users to drill down and drill up through data.

It is important to understand that PeopleSoft trees and hierarchies differ in the following manner: You create one tree for each dimension that requires a hierarchy; the analytic model uses that tree to create one hierarchy for one dimension.

Before loading the analytic model into the analytic server, the application uses the AttachTree method to attach the tree to its corresponding dimension. Next, the analytic model creates its own hierarchy by reading the parent-child relationships that are defined by that tree. During the remainder of the user session, the analytic model uses its own hierarchy, and no longer uses the original tree. For this reason, when the application adds a new dimension member during runtime, the member is actually added to the analytic model's hierarchy; the original tree is not modified.

Note: If a tree is not attached to a dimension, you can create a basic hierarchy for that dimension by specifying a total member name for the dimension.

See Defining Dimension Properties.

You can use the Analytic Model Viewer to view the properties of the trees that you are using with your analytic model.

See Viewing Dimension Properties.

PeopleCode Usage with PeopleSoft Trees and Analytic Models

Use the AttachTree and DetachTree methods to work with PeopleSoft trees and analytic models.

AttachTree Method

Use the AttachTree method to:

- Attach a tree to its corresponding dimension.

  Analytic Calculation Engine attaches the tree to the dimension and then creates and displays the hierarchy.

- Make changes to the tree.

- Create a record that uses PSACETREEOVRD as a subrecord, then attach the new record to the dimension members to associate the member override function with the hierarchy.

Be aware of these restrictions:

- Because the AttachTree method attaches a specific tree to an analytic model, the system throws an error if the tree's name, setID, or effective date is incorrect.

- You can attach only one tree to a dimension.

- If the analytic model is already loaded into an analytic server, the tree is not attached until the next time that the analytic model is reloaded.

DetachTree Method

Use the DetachTree method to detach the tree from the dimension.

Note: If the application loads the analytic model after the tree is detached, the analytic model does not create a hierarchy for the dimension.
If the analytic model is already loaded into an analytic server, the DetachTree method is not applied to the tree until the next time the application loads the analytic model.

Updating a Tree at Runtime

To update a tree at runtime, perform these steps:

1. Unload the analytic model.

   See “Unload” (PeopleCode API Reference).

2. Use the DetachTree method to detach the tree from the analytic model.

3. Use the AttachTree method's parameters to update the tree with the changes.

   Note: Be aware of the details start level and tree discard level before making any changes to the tree.

   See Purpose of Node Levels in Creating Hierarchies.

4. Reload the analytic model.

Related Links

"AttachTree" (PeopleCode API Reference)
"DetachTree" (PeopleCode API Reference)
"PeopleSoft Tree Manager Overview" (Tree Manager)

Understanding BAM Model Total Members

PeopleSoft BAM models often contain total members. A dimension in a BAM model may contain a total member to provide a simple aggregation of the other members of that dimension.

If you want to convert a BAM 8.8 model into an analytic model, you must understand how Analytic Calculation Engine handles the total members from BAM models, and the relationship between BAM total members and the hierarchies and dimension members of analytic models.

See Defining Dimension Properties.

Understanding Dimension Members

This topic provides an overview of dimension members present in analytic calculation engine.

Types of Dimension Members

Analytic Calculation Engine contains many different types of dimension members. The type of member that exists within a dimension is determined by:

- Whether a tree is attached to the dimension.
- The types of leaves or nodes that are mapped to the dimension members.

Note: Two dimension members should never share the same name unless one member is a detail member and one member is an aggregate member.

Detail Members and Leaf Members

If a tree is not attached to a dimension, Analytic Calculation Engine creates detail members for each value of the field to which the dimension is mapped.

If a tree is attached to a dimension, Analytic Calculation Engine creates detail members out of the tree's detail values to establish a dimension's parent and child relationships (in a tree, detail values can serve as children and parents).

See "Using Detail Values (Leaves)" (Tree Manager).

Note: When detail values serve as parents, they are also referred to as detail nodes because they do not display aggregated data. Instead, detail nodes usually display the key values of regular transactional tables.

When detail members serve as parents, they do not display aggregated data. Rather, they enable end users to navigate through the hierarchy.

Be aware of these characteristics of detail members' relationship to the main record:

- Detail member names are either read from the main record or generated from the tree's data.
- Navigation related functions such as PREV, NEXT, and PREVSELF operate on detail member names that are persisted in the main record.

  These functions do not use trees to determine the order of members.

  See PREV, NEXT, PREVSELF.

A leaf member is a special type of detail member that does not have children.

For example, suppose an end user enters 20040101 as a new detail value. Analytic Calculation Engine generates a new 20040101 leaf member. This is a leaf member because its corresponding detail value does not have any children.

Aggregate Members

Aggregate members are mapped to the nodes of a tree that have either children or leaf ranges. Aggregate members display a grouping of data, rather than a specific discrete value.

For example, suppose an analytic model's DATE dimension is mapped to a tree that contains 20040101 as a leaf node and Q12004 as a branch node. Analytic Calculation Engine generates the Q12004 aggregate member out of the branch node.

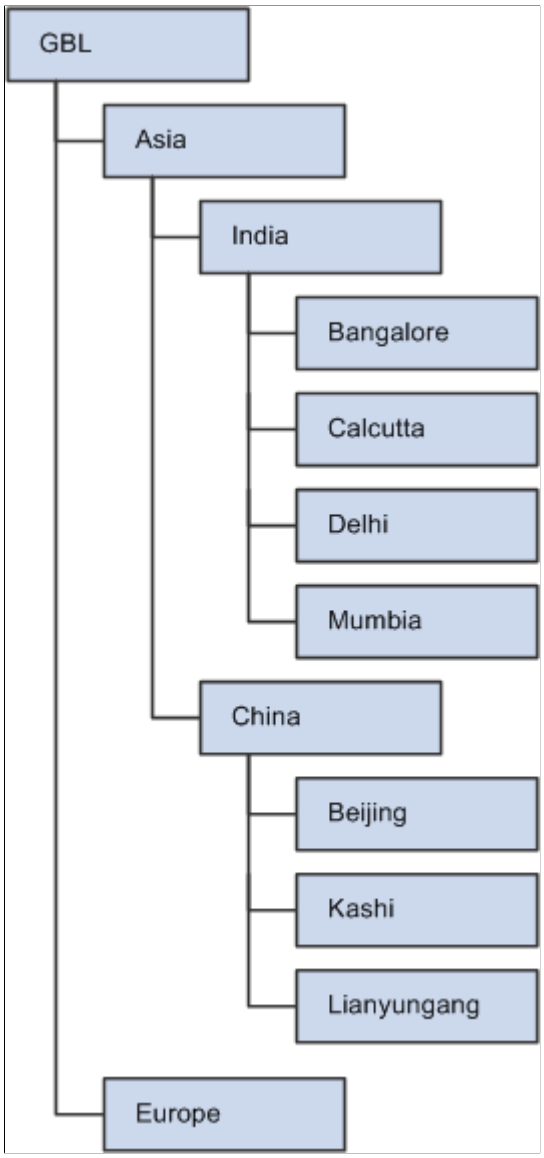
The analytic calculation engine creates aggregate members out of any tree elements that remain after it creates the hierarchy's detail members.

Hierarchy Root Member

You can map the hierarchy root member to any node that you want to serve as the root of the hierarchy. All sibling nodes or nodes at a higher level of the tree are not used to create the hierarchy. You map the hierarchy root member by using the NodeName parameter of the AttachTree method.

Note: Only one hierarchy root member can exist per dimension.

Consider this example of a tree's parent-child relationships.



Even though the highest level node is GBL, which is the root node of the tree, you can select the India node to serve as the hierarchy root member for this dimension. When you create the hierarchy root member out of the India node, only the children of India exist in the hierarchy.

If you have not attached a tree to the dimension, a hierarchy root member still exists for that dimension if you specified a root member name for that dimension. If you have not attached a tree to the dimension and you have not entered a value in the **Total Member Name** field, neither a hierarchy root member nor a hierarchy exists for that dimension.

See Defining Dimension Properties.

Orphan Members

An orphan member is any member that does not map to a child of a parent node in the tree.

For each orphan member, Analytic Calculation Engine:

- Adds each orphan member to the hierarchy root member.
- Adds each orphan member's value to the hierarchy root value's member.
- Generates a message with the ID of 123 and stores it in the Messages property for the analytic instance.

  You must write PeopleCode to iterate over the messages in the analytic instance and search for message 123, and then take any necessary further action.

  See “Error Handling” (PeopleCode API Reference).

Blank Members

A blank member is a member that has no value. Blank members are created out of either an empty detail in a tree or a null cell in the main record. To create blank members:

- When the analytic model learns of a new empty detail in the tree, it adds the blank member to the appropriate parent member.
- When the analytic model learns of a null cell in the main record, it adds the blank member as a child of the hierarchy root.
- When the AddMember method adds a member with a blank member name ( ), a blank member is added as a child of the hierarchy root.

Note: When blank members are mapped to date fields, they are written to the database as values of *1/1/1900.*

Related Links

Understanding the Elements of Rules

Purpose of Node Levels in Creating Hierarchies

Use node levels to create leaf, detail, and aggregate members out of tree nodes and leaves. Use the parameters of the AttachTree method to set the node levels.

Details Start Level

The details start level determines the type of dimension members that Analytic Calculation Engine creates out of the nodes and leaves of a tree. Use either the parameters of the AttachTree method or the Analytic Instance Load/Unload page to set the details start level. The details start level is a required parameter. The default value is 0. The root level is 1.

See Loading and Unloading Analytic Instances.

Note: If you specify a nonzero details start level, you must specify the strictly enforced method to the tree in PeopleSoft Tree Manager. The strictly enforced method ensures that all members that are created out of one level are created as the same member type.

See “Defining Basic Attributes” (Tree Manager).

The following table describes the members that Analytic Calculation Engine creates, depending on whether the details start level is specified.

Details Start Level Specified?	***Leaf Members***	***Detail Members***	***Aggregate Members***
Value > 0	Analytic Calculation Engine creates leaf members out of any detail values that are at the far right of the tree.	Analytic Calculation Engine creates detail members out of any detail values or nodes that are located either within the specified details start level, or within a level that is lower (higher number) than the details start level. If you specify the root level as the details start level, Analytic Calculation Engine creates detail members out of all nodes in the tree. **Note:** Analytic Calculation Engine cannot create detail members out of detail values that are at a higher level than the details start level. Do not specify a details start level that is equal to lower than the tree discard level.	Analytic Calculation Engine creates aggregate members out of any aggregate nodes that are located within a level that is higher than the details start level. **Note:** Analytic Calculation Engine cannot create aggregate members out of nodes that are at a lower level than the details start level.
Value = 0 **Note:** When the value = 0, the details start level is not specified.	Analytic Calculation Engine creates leaf members out of the detail values that are located at the far right of the tree.	Analytic Calculation Engine creates detail members out of all leaf members.	Analytic Calculation Engine creates aggregate members out of any nodes from which it has not created leaf members.

Tree Discard Level

The tree discard level determines the level from which Analytic Calculation Engine does not attach any more of the tree to the dimension. Use either the parameters of the AttachTree method or the Analytic Instance Load/Unload page to set the tree discard level.

See Loading and Unloading Analytic Instances.

Analytic Calculation Engine does not create members out of nodes or leaves that are either at this level or lower than this level. You must specify a details start level to every tree for which you want to specify a tree discard level. The default value is 0. If the tree discard level is anything other than Level 0, then the tree discard level must be at a lower level than the details start level.

The analytic calculation engine ignores the tree discard level if:

- The details start level is 0.
- The tree discard level is either equal to or higher than the details start level.

Creation of New Members at Runtime

Analytic Calculation Engine can create new dimension members during runtime by using:

- Data from the main record.
- Application data that is added at runtime.

Relationship of Leaf Ranges to New Members

If you map a dimension to a tree that includes leaf ranges, Analytic Calculation Engine adds a new dimension member to the appropriate parent in the hierarchy when the application adds a new leaf that falls within a leaf range of the tree. Use the AddMember method to add new members to the dimension.

Note: Analytic Calculation Engine ignores any new leaves that do not fall within the values of a leaf range.

For example, suppose a tree contains a node called 2003Q1 that includes a leaf range of 2003-01-01 to 2003-03-31. During runtime, Analytic Calculation Engine reads the main record data and recognizes that the application has added 20030204 as a new leaf that exists within the 2003Q1 leaf range. Analytic Calculation Engine creates the 20030204 member and adds it as a child to the 2003Q1 member.

Understanding the Calculation of Aggregate Data

This topic discusses the calculation of aggregate data in analytic calculation engine.

Related Links

Understanding Override Order of Precedence

Dimension Order Impact on Calculation

The order of dimensions in the analytic model determines which member the analytic calculation engine uses to calculate the data cube aggregate value that exists at an intersection of two or more aggregate members. When evaluating the data cube's value at this intersection, the analytic calculation engine uses the aggregate member of the dimension that appears as first in the order of dimensions in the part browser.

See Example: Creating Overrides.

Related Links

Changing the Order of Dimensions in the Part Browser

Logic for Determining the Order of Members

User functions often refer to dimension members to calculate data cubes. For this reason, you must understand the factors that affect the way in which the analytic calculation engine orders dimension members:

Note: This topic describes the member order as it exists within the analytic calculation engine. This internal order may differ from the member order that is ultimately displayed to the end user.

- If the dimension is mapped to a tree, the analytic calculation engine first determines the member order by the order of the detail values in the tree.

  Next, the analytic calculation engine determines the member order from the order of the values as they exist in the database.

  For example, suppose a dimension is attached to a GBL tree and uses the United Kingdom node as its hierarchy root member. This node contains detail values in this order: Cardiff, Liverpool, London, Manchester. This dimension is also mapped to the UNITED_KINGDOM field in the database, which contains the Edinburgh and Glasgow values. The analytic calculation engine creates this member order in the hierarchy:

  1. Cardiff
  2. Liverpool
  3. London
  4. Manchester
  5. Edinburgh
  6. Glasgow

  Note: The next time the analytic calculation engine creates these members (for example, when the application loads the analytic model), the analytic calculation engine employs the existing member order, even if it recognizes a new database value that matches the name of an existing member.
  For example, suppose the analytic model has already established the above hierarchy before the application adds the Manchester value to the UNITED_KINGDOM field. When the application reloads the analytic model, the member order in this hierarchy remains; for this reason, Manchester retains its fourth member position.

- If the dimension is not mapped to a tree, the member order is determined by the order of the members' values as they exist in the field that is mapped to the dimension.

Understanding the Persistence of Aggregate Data

This topic discusses the persistence of aggregate data in Analytic Calculation Engine.

Persistence of Aggregate and Detail Data

Assuming that you selected a main record and aggregate record in the cube collection, Analytic Calculation Engine uses these records to persist aggregate data and detail data:

- Tree data.

  Tree data includes:

  - Aggregate data:

    Analytic Calculation Engine persists aggregate data in the aggregate record.

    Note: Records that are used as aggregate records should be read after records that are used as main records.

    See Synchronization Order.

  - Pushed down data.

    Analytic Calculation Engine creates detail data out of pushed down aggregate data. For this reason, Analytic Calculation Engine persists this data in the main record.

    See Pushed Down Data.

- Detail data.

  Detail data is data that cannot be broken down any further. Analytic Calculation Engine persists detail data to the main record.

  A detail member is generated out of one value of detail data in the database.

Aggregate Record Properties

On a case-by-case basis, you must determine which aggregates you want to save for each dimension in the cube collection. You can select either *ALL, NONE,* or *ROOT* in the **Persist Aggregate** field of the Edit Cube Collection Dimension dialog box. Here are explanations for these selections:

- *ALL:* Select to persist all of the dimension member aggregate values to the database.
- *NONE:* Select to persist none of the dimension member aggregate values to the database.
- *ROOT:* Select to persist only the value of the hierarchy root member to the database.

See Defining Additional Cube Collection Dimension Properties.

Pushed Down Data

Sometimes a parent member in a hierarchy may contain aggregate data that is not derived by aggregating the parent member's children. When this is the case, you may want to break down the parent member's value to generate the detail data for the parent member's children. You create user functions that employ the PARENT, CHILDCOUNT, and FORCHILDREN built-in functions to push down aggregate data.

These user function examples push down parent member data:

```
AT (DIMENSION, Parent(DIMENSION), THISCUBE() * 0.2)
```

And:

```
AT (DIMENSION, Parent(DIMENSION), THISCUBE() / CHILDCOUNT(DIMENSION, #DIRECT))
```

Related Links

PARENT
CHILDCOUNT
FORCHILDREN

Data Type Considerations

Aggregate members can have different data types than their child detail members, even though they both display data for the same dimension. When this is the case, you must reconcile the situation if you want to persist aggregates.

For example, suppose the PRODUCTS dimension is mapped to a numeric field in the main record and contains members from this tree:

```
ALL_PRODUCTS
 Release Less than 8
  <Leaf Range (Low = 0, High= 799)
 Release 8
  <Leaf Range (Low = 800, High= 899)
```

Notice that the Release 8 member is not totally numeric; instead, it is a string that contains letters and a number. If the detail value 846 is added to the tree, the member 846 (which is numeric) is added to the parent member Release 8 (which is a string). To reconcile this situation, you should persist the aggregates for this dimension to a field with a data type of String and a length of at least 20.

Note: It is not necessary for the main record's fields to have the same data types as the aggregate record's fields.

Working with Overrides

This topic provides overviews of default aggregation, override order of precedence, and the PSACETREEOVRD subrecord.

Understanding Default Aggregation

By default, Analytic Calculation Engine sums all of the values of a parent member's direct children to calculate the value of the parent member. Analytic Calculation Engine executes this default aggregation by iterating over all child members and applying the plus operator. The default aggregation operates on all children, even if the child member itself is an aggregate value.

Note: This default aggregation is not used if you specified a cube dimension override user function, a member override user function, a dimension override user function, or do not have any aggregates in the relevant part of the analytic model.

This is an example of Analytic Calculation Engine's default aggregation:

```
2003  (170)
     Q1  (80)
        Jan  (10)
        Feb  (20)
        Mar  (50)
     Q2 (90)
        Apr   (20)
        May   (30)
        Jun   (40)
```

In this example, 2003, Q1, and Q2 are nonleaf members—that is, aggregates. The numbers in parentheses to the right of the aggregate members represent their aggregate values. The numbers in parentheses to the right of the nonaggregate members represent their nonaggregated values.

Understanding Override Order of Precedence

You must understand default aggregation and the order of precedence that the analytic calculation engine uses to override the default aggregation. When calculating aggregate members, Analytic Calculation Engine begins with the most specific override available, and then proceeds to more general overrides. The analytic calculation engine uses this order of precedence to evaluate aggregate members:

1. If the dimension does not contain any aggregate members, use the data cube's formula.

   If the dimension does contain aggregate members, perform one of these actions:

   - If aggregate calculation is not enabled for the data cube, do not perform any more calculation of aggregates.

     Note: The analytic calculation engine generates an error when the analytic model is loaded.

   - If the **Calculate Aggregates** option is selected for the data cube, perform step 2.

2. Perform one of these actions:

   Note: In either of these cases, the analytic calculation engine loads initial values from the aggregate record when the analytic model is loaded, but overwrites the initial values upon recalculation.

   - Use the cube dimension override user function if it exists.

     This override operates on all of a dimension's aggregate members for the dimension as it is attached to a specific data cube.

     For example, you can create one cube dimension user function to operate on the PRODUCTS dimension when it is attached to the SALES data cube, and another cube dimension user function to operate on the PRODUCTS dimension when it is attached to the COST_OF_GOODS data cube.

     To set a cube dimension override user function, create a user function in the analytic model, and then select the user function in the Aggregate Rule column in the Dimensions tab of the data cube's properties.

   - If the cube dimension user function does not exist, perform step 3.

3. Perform one of these actions:

   - Use the member override user function.

     This override operates on specific members of a dimension. You create the member override user function in the analytic model. However, because members are often instantiated at runtime, you use the PSACETREEOVRD subrecord to assign the member override user function to the dimension rather than assign the member override user function to specific members within the analytic model.

     For example, if the end user enters aggregate data, you can write a member override function that uses the INPUT built-in function to capture the user input, and use the PARENT and CHILDCOUNT built-in functions to push down the aggregate data and create new members. Then you can assign the member override user function to the appropriate dimension in the PSACETREEORRD subrecord.

     Note: If the member override user function does not contain a value, Analytic Calculation Engine applies the default aggregation (the plus operator) to the dimension members.

   - If the member override function does not exist, perform step 4.

4. Perform one of these actions:

   - Use the dimension override function.

     The analytic calculation engine uses this override user function to calculate all of a dimension's aggregates as they apply to all dimensions.

     For example, suppose you create a dimension override function for the PRODUCTS dimension. If this dimension is attached to both the SALES and COST_OF_GOODS dimensions, the dimension override function applies to the aggregates for both data cubes.

     To set a dimension override user function, create a user function and select the user function in the **Aggregate Function** field of the dimension's properties.

     Note: This override function does not apply to leaf members or detail members.

   - If the dimension override function does not exist, perform step 5.

5. Aggregate the values of the child members by using the operators that are attached to each child.

   The analytic calculation engine iterates over each value to evaluate the aggregate.

   The default operator for each member is the plus operator.

   You set the operators in the **OPERATION** field of the PSACETREEOVRD subrecord. When the default sum operator is used, the actual value in the **OPERATION** field is null.

   If you do not want to use the default sum aggregation, you can populate the **OPERATION** field with one of these values:

   - MIN.

Use this value for the analytic calculation engine to use subtraction aggregation. The analytic calculation engine iterates over each member and applies the subtraction operator to each iteration. The aggregate value is the resulting value after the final iteration.

- IGN.

  Use this value for the analytic calculation engine to ignore the **OPERATION** field.

Understanding the PSACETREEOVRD Subrecord

You use the PSACETREEOVRD subrecord to assign an override to a tree. To use the subrecord, you must first create a record definition. This record definition can contain additional fields. After you create the record definition, insert the PSACETREEOVRD subrecord into the record definition.

To attach the PSACETREEOVRD subrecord to a tree, specify the name of the override record as the RecordName parameter of the AttachTree method.

The following table describes the PSACETREEOVRD subrecord.

Field Name	***Description***	***Possible Values***
TREE_NAME	The name of the tree that contains the node to override.	DEPT_TREE
SET_ID	The setID of the tree.	123
EFFDT	The effective date of the tree.	12/1/03
TREE_NODE	The name of the dimension member on which the override should operate.	GBL
ACERULEID	The name of the user function to apply as an override. This field can be null because it can be reused.	USER_FUNCTION_NAME
OPERATION	Add, subtract, or ignore this entry when aggregating a parent member.	MIN, IGN **Note:** The default value in the OPERATION field is null, causing the analytic calculation engine to use the sum operator for aggregating members. Other possible values in the table are *MIN* for subtraction aggregation and *IGN* for ignore.

Note: The PSACETREEOVRD subrecord must contain a value in either or both of the OPERATION or ACERULEID fields. If both of these fields are null, the analytic calculation engine ignores the row.

Example: Using Default Aggregation

This topic provides an example of how to create an analytic model that uses default aggregation.

Requirements for Analytic Model

The following table provides an example of a table named MainData, which you specified as the main record of a cube collection.

DEPT field ***Data Type: Number***	***SOMEDATE field*** ***Data type: Date***	***NUM_SALES field*** ***Data Type: Number***	***PRICE_PER_UNIT field*** ***Data Type: Number***
101	20040101	1	10
102	20040102	2	14
103	20040101	4	15
201	20040101	8	20
202	20040201	16	23

You want the analytic model to:

- Calculate aggregates for the NUM_SALES field.
- Save the aggregates for the NUM_SALES field.
- Establish hierarchies for the DEPT and SOMEDATE dimensions.
- Refrain from calculating aggregates for the PRICE_PER_UNIT field.
- Save all aggregates for the DEPT dimension.
- Refrain from saving aggregates for the SOMEDATE dimension.

Creating the Aggregates

To fulfill the requirements of the analytic model, perform these steps:

1. Create these dimensions:
   - DEPARTMENT
   - DATE
2. Create these data cubes:
   - SALES

- PRICE_PER_UNIT

3. Enable the **Calculate Aggregates** field for the SALES data cube, because you want to calculate aggregates for this data cube.
4. Ensure that the **Calculate Aggregates** field is disabled for the PRICE_PER_UNIT data cube, because you do not want to calculate aggregates for this data cube.
5. Create a cube collection called SALES.
6. Select MainData as the main record for the SALES cube collection.
7. Map the data cubes and dimensions within the SALES cube collection to fields in the main record.

   The following table provides the mappings.

Data Cube or Dimension to Map	***Field in Main Record***
DEPARTMENT dimension	DEPARTMENT field Data type: Number
SOMEDATE dimension	SOMEDATE field Data type: Date
SALES data cube	NUM_SALES field Data type: Number
PRICE_PER_UNIT data cube	PRICE_PER_UNIT field Data type: Number

8. Select the AGGRDATE record as the aggregate record.

   The AGGRDATE database record currently contains no data. The following table describes the fields within the record.

Field Name	***Data Type***
DEPARTMENT	String
TREE_DATE	String
NUM_SALES	Number

Notice that even though the DEPARTMENT and TREE_DATE fields are of the String data type, none of the main record's fields are of this same data type. The DEPARTMENT and TREE_DATE fields are of the String data type because the hierarchy's members display strings, not dates or numbers. The data types of the aggregate record's fields must match the data types and lengths of the hierarchy's aggregate members. However, the data types of the aggregate record's fields do not need to match the data types of the main record's fields.

Note: If you design a tree's nodes so that the fields of the nodes are of the same data type as the fields of the detail members, you can use the same data type for both the dimension's aggregate record fields and main record fields.

---

9. Map dimensions and data cubes to fields in the aggregate record.

   The following table lists the mappings.

Data Cube or Dimension to Map	***Field in Aggregate Record***
DEPARTMENT dimension	DEPARTMENT field
SOMEDATE dimension	TREE_DATE field
SALES data cube	NUM_SALES field

10. Use PeopleSoft Tree manager to create two trees:

Note: In the two tables, italicized children represent detail values. Remember, Analytic Calculation Engine creates detail members out of the tree's detail values to establish a dimension's parent-child relationships (in a tree, detail values can serve as children and parents).

- DEPT_TREE

Parents	***Children***
(no parent root)	*GBL*
GBL	*US*
GBL	*LAT AM*
US	101
US	102
US	103

Parents	*Children*
LAT AM	201
LAT AM	202

- DATE_TREE

Parents	*Children*
(no parent root)	*CAL2004*
CAL2004	*JAN*
CAL2004	*FEB*
JAN	20040101
JAN	20040102
FEB	20040201

11. Consider these scenarios for default sum aggregation:

    - If you select to persist all aggregates of both dimensions on the Dimensions tab of the cube collection's properties, the following rows are persisted in the aggregate record.

    Note: Italicized values are the actual persisted aggregate members. Zero (0) values in this table are not persisted. The **Fully Qualified Member Name** field is not a database value.

Fully Qualified Member Name	*DEPARTMENT*	*TREE_DATE*	*NUM_SALES*
GBL CAL2004.JAN. 20040101	*GBL*	20040101	13
GBL CAL2004.JAN.20040102	*GBL*	20040102	2

Fully Qualified Member Name	***DEPARTMENT***	***TREE_DATE***	***NUM_SALES***
GBL CAL2004.FEB.20040201	*GBL*	20040201	16
GBL.US CAL2004.JAN.20040101	*US*	20040101	5
GBL.US CAL2004.JAN.20040102	*US*	20040102	2
GBL.US CAL2004.FEB.20040201	*US*	20040201	0
GBL.LAT AM CAL2004.JAN.20040101	*LAT AM*	20040101	8
GBL.LAT AM CAL2004.JAN.20040102	*LAT AM*	20040102	0
GBL.LAT AM CAL2004.FEB.20040201	*LAT AM*	20040201	16
GBL.US.101 CAL2004	101	*CAL2004*	1
GBL.US.102 CAL2004	102	*CAL2004*	2
GBL.US.103 CAL2004	103	*CAL2004*	4
GBL LAT AM.201 CAL2004	201	*CAL2004*	8

Fully Qualified Member Name	***DEPARTMENT***	***TREE_DATE***	***NUM_SALES***
GBL LAT AM.202 CAL2004	202	*CAL2004*	16
GBL.US.101 CAL2004.JAN	101	*JAN*	1
GBL.US.102 CAL2004.JAN	102	*JAN*	2
GBL.US.103 CAL2004.JAN	103	*JAN*	4
GBL.LAT AM.201 CAL2004.JAN	201	*JAN*	8
GBL.LAT AM.202 CAL2004.JAN	202	*JAN*	0
GBL.US.101 CAL2004.FEB	101	*FEB*	0
GBL.US.102 CAL2004.FEB	102	*FEB*	0
GBL.US.103 CAL2004.FEB	103	*FEB*	0
GBL. LAT AM.202 CAL2004.FEB	201	*FEB*	0
GBL. LAT AM.203 CAL2004.FEB	202	*FEB*	16

Fully Qualified Member Name	*DEPARTMENT*	*TREE_DATE*	*NUM_SALES*
GBL CAL2004	*GBL*	*CAL2004*	31
GBL CAL2004.JAN	*GBL*	*JAN*	15
GBL CAL2004.FEB	*GBL*	*FEB*	16
GBL.US CAL2004	*US*	*CAL2004*	7
GBL.US CAL2004. JAN	*US*	*JAN*	7
GBL.US CAL2004.FEB	*US*	*FEB*	0
GBL.LAT AM CAL2004	*LAT AM*	*CAL2004*	24
GBL.LAT AM CAL2004.JAN	*LAT AM*	*JAN*	8
GBL. LAT AM CAL2004.FEB	*LAT AM*	*FEB*	16

- If you select to persist all aggregates of the DEPT_TREE dimension and to persist none of the aggregates of the DATE_TREE dimension on the Dimensions tab of the cube collection's properties, the following rows are persisted in the aggregate record.

Note: Italicized values are the actual persisted aggregate members. Zero (0) values in this table not persisted. The **Fully Qualified Member Name** field is not a database value.

Fully Qualified Member Name	*DEPARTMENT*	*TREE_DATE*	*NUM_SALES*
GBL CAL2004.JAN. 20040101	*GBL*	20040101	13
GBL CAL2004.JAN.20040102	*GBL*	20040102	2
GBL CAL2004.FEB.20040201	*GBL*	20040201	16
GBL.US CAL2004.JAN.20040101	*US*	20040101	5
GBL.US CAL2004.JAN.20040102	*US*	20040102	2
GBL.US CAL2004.FEB.20040201	*US*	20040201	0
GBL.LAT AM CAL2004.JAN.20040102	*LAT AM*	20040102	8
GBL.LAT AM CAL2004.JAN.20040102	*LAT AM*	20040102	0
GBL.LAT AM 20040201	*LAT AM*	20040201	16

- If you select to persist only the root aggregations of the DEPT_TREE dimension and to persist none of the aggregates of the DATE_TREE dimension on the Dimensions tab of the cube collection's properties, the following rows are persisted in the aggregate record.

Note: Italicized values are the actual persisted aggregate members. The **Fully Qualified Member Name** field is not a database value.

Fully Qualified Member Name	*DEPARTMENT*	*TREE_DATE*	*NUM_SALES*
GBL CAL2004.JAN. 20040101	*GBL*	20040101	13
GBL 20040102	*GBL*	20040102	2
GBL 20040201	*GBL*	20040201	16

Example: Creating Overrides

This topic provides an example of creating overrides and discusses the affect of dimension order on calculation.

The following table describes the three dimensions used in this example. The first column lists the names of the dimensions. The second column lists the dimension order, which determines calculation priority. The third column lists the dimension override functions that are used if member override functions do not exist for the children of the parents in the dimension.

Dimension	*Dimension Order/Priority*	*Dimension Override User Function*
ACCT	1	<ACCT_DIM_DEFAULT_FORMULA>
TRANS_DATE	2	<NONE>
DEPT	3	<DEPT_DIM_DEFAULT_FORMULA>

The following table describes the hierarchy of the ACCT dimension that is associated with the AcctTree tree. The first column lists the parents in the hierarchy. The second column lists the children of the parents. The third column lists the member override user functions that are performed on each child.

Note: Overrides are not performed on cells denoted (leaf) or <none>.

Parent	*Child*	*Member Override User Function*
100	110	<SALES_ACCT_SUM>

Parent	*Child*	*Member Override User Function*
100	110	<DIRECTOR_ACCT_SUM>
100	120	<none>
110	111	(leaf)
110	112	(leaf)
120	121	(leaf)

The following table describes the hierarchy of the TRANS_DATE dimension that is associated with the QrtrlyTree tree. The first column lists the parents in the hierarchy. The second column lists the children of the parents. The third column lists the member override user functions that are performed on each child.

Note: Overrides are not performed on cells denoted (leaf) or <none>.

Parent	*Child*	*Member Override User Function*
Q1	Q1	<none>
Q1	Jan	(leaf)
Q1	Feb	(leaf)
Q1	Mar	(leaf)

The following table describes the hierarchy of the DEPT dimension that is created from the DeptTree tree. The first column lists the parents in the hierarchy. The second column lists the children of the parents. The third column lists the member override user functions that are performed on each child.

Note: Overrides are not performed on cells denoted (leaf) or <none>.
This example uses the <RED_HERRING> child node override as incorrect data. Leaf nodes do not have aggregations.

Parent	*Child*	*Member Override User Function*
GBL	GBL	<SOME_DEPT_AVG>
GBL	USA	<none>
GBL	EUR	<none>

Parent	*Child*	*Member Override User Function*
USA	CA	(leaf)
USA	GA	(leaf)
USA	NY	(leaf)
USA	TX	<RED_HERRING>
USA	IL	(leaf)

Assume that a SALES data cube exists in the cube collection, and the three dimensions of this example are attached to this data cube.

Read the instructions carefully before analyzing the following table; the table describes two methods that the analytic calculation engine can use to calculate hierarchies.

- When you analyze only the first four columns of the table (ignore the fifth column), the basic analytic model does not contain any cube dimension overrides.

  The first column displays the row numbers. The second, third, and fourth columns list the members of each of the three dimensions.

- When you analyze all five rows of the table, the basic analytic model does contain cube dimension overrides.

  The first column displays the row numbers. The second, third, and fourth columns list the members of each of the three dimensions. The fifth column—where it applies—lists the override that the analytic calculation engine uses to calculate the row.

  For example, if the developer applies the SALES_CUBE_OVERRIDE cube dimension override to the TRANS_DATE dimension as it is attached to the SALES data cube, the Cube Dimension Override User Function column indicates where the override is applied.

In the following table, italicized values indicate aggregate members. A row may contain more than one aggregate member. If a row contains one or more aggregate members, the table denotes the winning aggregate member along with the method that the analytic calculation engine uses to calculate the member. If a row does not contain any aggregate members, the analytic calculation engine uses the data cube's rule to calculate values.

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
1	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	*Q1*	*USA*	N/A
2	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	*Q1*	CA	N/A
3	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	*Q1*	NY	N/A
4	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	*Q1*	TX	N/A
5	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Jan	*USA*	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
6	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Jan	CA	N/A
7	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Jan	NY	N/A
8	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Jan	TX	N/A
9	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Feb	*USA*	N/A
10	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Feb	CA	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
11	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Feb	NY	N/A
12	Winning aggregate: *100* Use member override user function: USA <SALES_ACCT _SUM>	Feb	TX	N/A
13	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_ SUM>	*Q1*	*USA*	N/A
14	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_ SUM>	*Q1*	CA	N/A
15	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_ SUM>	*Q1*	NY	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
16	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	*Q1*	TX	N/A
17	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Jan	*USA*	N/A
18	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Jan	CA	N/A
19	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Jan	NY	N/A
20	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Jan	TX	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
21	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Feb	*USA*	N/A
22	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Feb	CA	N/A
23	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Feb	NY	N/A
24	Winning aggregate: *110* Use member override user function: <DIRECTOR_ACCT_SUM>	Feb	TX	N/A
25	Winning aggregate: *120* Use the <ACCT_DIM_DEFAULT_FORMULA> dimension override user function because a member override user function does not exist for this member.	*Q1*	*USA*	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
26	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	*Q1*	CA	N/A
27	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	*Q1*	NY	N/A
28	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	*Q1*	TX	N/A
29	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Jan	*USA*	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
30	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Jan	CA	N/A
31	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Jan	NY	N/A
32	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Jan	TX	N/A
33	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Feb	*USA*	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
34	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Feb	CA	N/A
35	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Feb	NY	N/A
36	Winning aggregate: *120* Use the <ACCT_ DIM_DEFAULT _FORMULA> dimension override user function because a member override user function does not exist for this member.	Feb	TX	N/A
37	111	Winning aggregate: *Q1* Use default sum aggregation because neither a member override user function exists for this member, nor a dimension override user function exists for this dimension.	USA	<SALES_CUBE_ OVERRIDE>

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
38	111	Winning aggregate: *Q1* Use default sum aggregation because neither a member override user function exists for this member, nor a dimension override user function exists for this dimension.	CA	<SALES_CUBE_ OVERRIDE>
39	111	Winning aggregate: *Q1* Use default sum aggregation because neither a member override user function exists for this member, nor a dimension override user function exists for this dimension.	NY	<SALES_CUBE_ OVERRIDE>
40	111	Winning aggregate: *Q1* Use default sum aggregation because neither a member override user function exists for this member, nor a dimension override user function exists for this dimension.	TX	<SALES_CUBE_ OVERRIDE>

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
41	111	Jan	Winning aggregate: *USA* Use the dimension override user function <DEPT_ DIM_DEFAULT_ FORMULA> because a member override user function does not exist for this member.	N/A
42 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	111	Jan	CA	N/A
43 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	111	Jan	NY	N/A
44 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	111	Jan	TX	N/A

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
45	111	Feb	Winning aggregate: *USA* Use the dimension override user function <DEPT_ DIM_DEFAULT_ FORMULA> because a member override user function does not exist for this member.	N/A
46 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	111	Feb	CA	N/A
47 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	111	Feb	NY	N/A
48 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	111	Feb	TX	N/A
49	112	Winning aggregate: *Q1* Use default sum aggregation because a member override user function does not exist for this member.	*USA*	<SALES_CUBE_ OVERRIDE>

Row	***ACCT Dimension Member*** ***Priority 1***	***TRANS_DATE Dimension Members*** ***Priority 2***	***DEPT Dimension Members*** ***Priority 3***	***Cube Dimension Override User Function***
50	112	Winning aggregate: *Q1* Use default sum aggregation because a member override user function does not exist for this member.	CA	<SALES_CUBE_ OVERRIDE>
51	112	Winning aggregate: *Q1* Use default sum aggregation because a member override user function does not exist for this member.	NY	<SALES_CUBE_ OVERRIDE>
52	112	Winning aggregate: *Q1* Use default sum aggregation because a member override user function does not exist for this member.	TX	<SALES_CUBE_ OVERRIDE>
53	112	Jan	Winning aggregate: *USA* Use the dimension override user function <DEPT_ DIM_DEFAULT_ FORMULA> because a member override user function does not exist for this member.	N/A
54 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	112	Jan	CA	N/A

Row	*ACCT Dimension Member* *Priority 1*	*TRANS_DATE Dimension Members* *Priority 2*	*DEPT Dimension Members* *Priority 3*	*Cube Dimension Override User Function*
55 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	112	Jan	NY	N/A
56 *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	112	Jan	TX	N/A
57	112	Feb	Winning aggregate: *USA* Use the dimension override user function <DEPT_ DIM_DEFAULT_ FORMULA> because a member override user function does not exist for this member.	N/A
58 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	112	Feb	CA	N/A
59 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	112	Feb	NY	N/A

Row	*ACCT Dimension Member* *Priority 1*	*TRANS_DATE Dimension Members* *Priority 2*	*DEPT Dimension Members* *Priority 3*	*Cube Dimension Override User Function*
60 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	112	Feb	TX	N/A
61	121	Winning aggregate: *Q1* Use default sum aggregation.	USA	<SALES_CUBE_ OVERRIDE>
62	121	Winning aggregate: *Q1* Use default sum aggregation.	CA	<SALES_CUBE_ OVERRIDE>
63	121	Winning aggregate: *Q1* Use default sum aggregation.	NY	<SALES_CUBE_ OVERRIDE>
64	121	Winning aggregate: *Q1* Use default sum aggregation.	TX	<SALES_CUBE_ OVERRIDE>
65	121	Jan	Winning aggregate: *USA* Use the dimension override user function <DEPT_ DIM_DEFAULT_ FORMULA> because a member override user function does not exist for this member.	N/A

Row	*ACCT Dimension Member* *Priority 1*	*TRANS_DATE Dimension Members* *Priority 2*	*DEPT Dimension Members* *Priority 3*	*Cube Dimension Override User Function*
66 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	121	Jan	CA	N/A
67 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	121	Jan	NY	N/A
68 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	121	Jan	TX	N/A
69	121	Feb	Winning aggregate: *USA* Use the dimension override user function <DEPT_ DIM_DEFAULT_ FORMULA> because a member override user function does not exist for this member.	N/A
70 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	121	Feb	CA	N/A

Row	*ACCT Dimension Member* *Priority 1*	*TRANS_DATE Dimension Members* *Priority 2*	*DEPT Dimension Members* *Priority 3*	*Cube Dimension Override User Function*
71 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	121	Feb	NY	N/A
72 **Note:** *No winning aggregate.* Because this row does not contain any aggregates, use the data cube's rule for calculating values.	121	Feb	TX	N/A

The analytic calculation engine used this logic to determine which cell of the row it finally used to calculate the aggregate:

- Aggregation for row 25:

  The analytic calculation engine used the *120* value from the TRANS_DATE dimension because this dimension was the only dimension that contained an aggregate member. Understand that if either of the two other dimensions contained an aggregate member, the analytic calculation engine would still select the 120 value because the TRANS_DATE dimension is first in priority. The analytic calculation engine used the <ACCT_DIM_DEFAULT_FORMULA> dimension override because neither a cube dimension user function nor a member override user function existed for this member.

- Aggregation for row 37:

  The analytic calculation engine used the *Q1* value from the TRANS_DATE dimension because this was the only dimension that contained an aggregate member. The analytic calculation engine used the default sum aggregation because neither a member override user function existed for this member nor a dimension override user function existed for this dimension.

- Aggregation for row 41:

  The analytic calculation engine used the *USA* value from the DEPT dimension because this was the only dimension that contained an aggregate member. The analytic calculation engine used <DEPT_DIM_DEFAULT_FORMULA> because neither a cube dimension user function nor a member override user function existed for this member.

Example: Creating a Hierarchy with Mixed Aggregate and Detail Members

In this example, the analytic model contains a BONUS cube collection that calculates the bonus for a group of employees. The BONUS cube collection uses the main record, as described in the following table.

Employee	***Bonus (in thousands)***
VP	300
D1	200
D2	100
M1	40
M2	10
M3	20

This example uses the following tree, named BUS1:

```
VP - Vice president
   D1 - Director 1
      M1 - Manager 1
   D2 - Director 2
      M2 - Manger 2
      M3 - Manager 3
```

The hierarchical relationships in the BUS1 tree are:

- D1 and D2 are directors who report to VP.
- M1 is a manager who reports to D1.
- M2 and M3 are managers who report to D2.

The BONUS cube collection contains a data cube called EMPLOYEE_BONUS, to which the EMPLOYEE dimension is attached.

You do not want to calculate the vice president's bonus by summing the bonuses of all of the vice president's children. The VP node should not exist as an aggregate member of the hierarchy; it should exist instead as a detail member. For this reason, do not calculate aggregates for the EMPLOYEE_BONUS data cube.

To create the correct members to the nodes of this tree, specify the details start level as level two (because the root level is level one). With this specification, detail members are created out of every tree node at the VP level and any level lower than the VP level. End users use the hierarchy for navigating throughout the tree. Remember that you can create hierarchies that are only used for navigation.

Make this data available to the end user:

- The bonus for every employee.
- The total bonuses for each employee, plus the total bonuses for each employee who reports to him or her.

To achieve these results, perform these steps:

1. Create a data cube called TOTAL_BONUS.
2. Attach the EMPLOYEE_BONUS dimension to the TOTAL_BONUS data cube.
3. Add the TOTAL_BONUS data cube to the BONUS cube collection.

In this example's hierarchy, all the members are detail members except for the hierarchy root member. Because aggregate user functions are only used to calculate aggregate members, you should create a regular formula for the TOTAL_BONUS data cube to calculate its aggregates. Use the FORCHILDREN built-in function to sum the value of the EMPLOYEE_BONUS data cube plus all of the children of the member. For example:

```
FORCHILDREN(DIMENSION, MEMBER, EXPRESSION)
```

Because you specified level two as the details start level, the root member is calculated as an aggregate. The analytic calculation engine calculates both the TOTAL_BONUS and EMPLOYEE_BONUS data cubes by using the sum default, because you did not create and select an aggregate function for this purpose.

The following table displays the calculation results of all members that are attached to the EMPLOYEE_BONUS data cube.

Full Path to Employee in Hierarchy	***Employee (Dimension Member)***	***Employee Bonus (in thousands)***	***Total Bonus (in thousands)***
BUS1	BUS	300	670
BUS1.VP	VP	300	670
BUS1.VP.D1	D1	200	240
BUS1.VP.D2	D2	100	130
BUS1.VP.D1.M1	M1	40	40
BUS1.VP.D2.M2	M2	10	10
BUS1.VP.D2.M3	M3	20	20

Chapter 10

Creating Rules, Formulas, and User Functions

Understanding Rules, Formulas, and User Functions

This topic lists common elements and discusses rules, formulas, and user functions, filter user functions, and the rule bar display.

Common Elements Used in This Product Documentation

Field or Control	*Description*
+	Click the **Plus** icon to insert a plus symbol into the rule.
-	Click the **Minus** icon to insert a minus symbol into the rule.
*	Click the **Multiply** icon to insert a multiplication symbol into the rule.
/	Click the **Divide** icon insert a division symbol into the rule.
^	Click the **Exponent** icon to insert an exponent symbol into the rule.
(	Click the **Left Parenthesis** icon to insert a left parenthesis into the rule.
)	Click the **Right Parenthesis** icon to insert a right parenthesis into the rule.
<	Click the **Less Than** icon to insert a less than symbol into the rule.
>	Click the **Greater Than** icon to insert a greater than symbol into the rule.

Field or Control	*Description*
=	Click the **Equals** icon to insert an equal symbol into the rule.
AND	Click the **AND Operator** icon to insert an AND operator into the rule.
OR	Click the **OR Operator** icon to insert an OR operator into the rule.
NOT	Click the **NOT Operator** icon to insert a NOT operator into the rule.
F()	Click the **Paste Build-in Function** icon to paste a built-in function and its arguments into the rule.
	Click the **Paste Cube** icon to paste a data cube name into the rule.
	Click the **Paste Dimension** icon to paste a dimension name into the rule.
	Click the **Paste Member Reference** icon to paste a member reference into the rule.
f()	Click the **User Function** icon to paste a user function into the rule.
	Click the **Exit Formula Mode** icon to exit the formula without canceling the changes or validating the formula.

Rules, Formulas, and User Functions

In Analytic Calculation Engine, you use the rule bar to create rules that define the calculation of data. You use rules within formulas and user functions.

Formulas define the calculation of data cubes. You enter the formula within the rule bar of the data cube that you want to calculate.

You can create a formula and save it as a user function, which can be reused with various data cubes by entering the name of the user function in the rule bar of the relevant data cube. You also create user functions to create filters and to define the calculation of aggregates.

Analytic Calculation Engine enables you to create rules that contain references to other parts. When the values of these other parts change, the analytic calculation engine recalculates the rule and stores the results in the field mapped to the calculated part. These kinds of rules can be useful for working with

assumption data. When end users work with analytic instance data within an application, they can enter assumption values into one data cube, and then view the results of those assumptions in the values of other calculated data cubes.

For example, suppose an analytic model contains three data cubes called PROFIT, INCOME, and EXPENSE. The PROFIT data cube contains this formula:

```
INCOME - EXPENSE
```

When an end user changes a value that is tied to the INCOME or EXPENSE data cube, the analytic calculation engine recalculates the formula and stores the result in the field that is mapped to the PROFIT data cube.

Filter User Functions

You apply a filter user function to a specific dimension, on the Dimensions tab of the cube collection's properties.

See Defining Additional Cube Collection Dimension Properties.

This topic discusses:

- Data filters.
- Dimension member filters.

Data Filters

You can create filter user functions to display only the dimension members whose values meet a certain condition. For example, this is the formula for the FILTER_PROD_OVER_2000 filter user function, which is applied to the PRODUCTS dimension:

```
IF (SALES > 2000, RETURN(1), RETURN(0))
```

In the analytic model, only the PRODUCTS dimension is attached to the SALES data cube. In the analytic grid, the end user views the SALES data cube but has access only to the products that have sold over 2,000 units.

Here is the formula for the FILTER_RED_PRODUCTS filter user function, which is applied to the PRODUCTS dimension:

```
IF(PRODUCT_COLOR = "RED", RETURN(1), RETURN(0))
```

In this example, the end user has access only to the products whose members have the red attribute.

When a filter user function is applied to a dimension that is attached to a multidimensional data cube, the end user has access to a different set of members depending on whether the filtered dimension is in the column axis/row axis or slice bar.

Using the first filter user function example, the PRODUCTS, MONTHS, and REGIONS dimensions are attached to the SALES data cube. When only the PRODUCTS dimension is in the column or row axis—and the other dimensions are in the slice bar—the end user has access to only the PRODUCTS members that have sold over 2,000 units in the currently selected region and month in the slice bar. If the end user changes the region or month selection in the slice bar, the filter is reapplied and the analytic grid may display a different set of PRODUCTS members.

However, when the PRODUCTS dimension plus one or more dimensions are in the slice bar, the end user has access to a different set of dimension members. For example:

- If the PRODUCTS dimension is in the slice bar, the end user has access to all PRODUCTS members that have sold over 2,000 units in all regions over the course of all months, regardless of where these dimensions are displayed in the grid.
- If the PRODUCTS dimension is in the row headings, the MONTHS dimension is in the column headings and the REGION dimension is in the slice bar, the end user has access to all PRODUCTS members that have sold over 2,000 units in the currently selected region in the slice bar, over the course of all months.
- If the PRODUCTS and MONTHS dimensions are in the row headings—and the MONTHS dimension is indented below the PRODUCTS dimension—plus the REGION dimension is in the slice bar, the end user has access to all PRODUCTS members that have sold over 2,000 units in the currently selected region in the slice bar, for the month under which the products are displayed.

  This means that the analytic grid may display a different set of products for each month.

Dimension Member Filters

You can create filter user functions to display only the dimension members that are referenced in the filter function. For example, this is the formula for the FILTER_DIGITAL_CAMERAS filter user function, which is applied to the PRODUCTS dimension:

```
IF(MEMBER(PRODUCTS) = [PRODUCTS:Digital Cameras], RETURN(1), RETURN(0))
```

In this example, the end user only has access to the Digital Cameras member of the PRODUCTS dimension.

You can also create filter functions that filter data by user ID by using the OPRID built-in function.

See OPRID.

Rule Bar Display

The information that is displayed in the rule bar depends on the selected part. The following table lists the parts and the resulting rule bar display.

Selected Part	***Rule Bar Display***
Data cube	The data cube's formula (if any).
User function	The user function's rule.
All other parts	Remains blank.
No selected part	Remains blank.

To edit a formula or user function, click in the rule bar. The rule bar displays buttons that enable you to edit rules.

Understanding Design Time Rule Error Messages

When creating an analytic model, it is important that you create rules that follow certain guidelines. For example, a multiplication symbol needs a value or expression on both sides of the symbol; therefore, if you create a rule such as `3 + 5 *`, the analytic calculation engine cannot interpret the rule.

When you either click the **Accept Changes** button to accept a rule or you select *Tools, Validate Project,* the analytic calculation engine examines the analytic model's rules for errors. All error messages for rules appear in the Output window. When you click an error message, the cursor moves to the part or rule in the analytic model definition that caused the error message. At this time, you can edit the rule in question and fix the error.

The following table describes Analytic Calculation Engine's rule error messages and how to resolve them.

Note: When %1 or %2 appears in this table, it denotes that the actual error message includes context-specific information. For example, the Invalid Dimension %1 error message would yield the error Invalid dimension PRODUCTS in the Output window if a rule referred to a nonexistent PRODUCTS dimension.

Error Message	***Description***
A dimension argument cannot be used here.	An invalid argument was passed to the function. The function does not take a dimension as an argument. Please check the number and argument types for the function in question.
Analytic model with name %1 not found.	The analytic model was not found in the PeopleSoft database. Please make sure that the model is saved before the validate is called.

All dimension arguments must be declared before any expression arguments are declared.	All the dimension arguments must be declared before expression arguments are declared. Dimension arguments are declared with a prefix of $, and expression arguments are declared with a prefix of @. For example:
	
	`ARGUMENTS($DIM, @ExprToLookup,`
	`@Condition,`
	`@Direction := #FORWARD);`

All required arguments must be declared before any optional arguments are declared.	Optional arguments should be placed at the end of the declaration. If there are two or more optional arguments, place the most optional argument last. For example:
	
	`ARGUMENTS($Dim, @ExprToLookup,`
	`@Condition,`
	`@Direction := #FORWARD);`
	
	In this example, @Direction is an optional argument and is placed after the non-optional @Condition argument.
	
	Note: Optional arguments should have a default value.

Circular reference.	See the Circular Reference section below.

Error Message	*Description*
Comment is not terminated.	The comment in a rule was not terminated with the symbols >>. See Inserting a Blank Line into a Rule.
Data cube name is not terminated by a single quote.	The data cube reference in a rule contained a starting single quote but was not terminated with a single quote. Valid syntax for a data cube reference is either of these: • `DATA_CUBE` • `'Data Cube'`
Duplicate argument name %1.	The argument mentioned in the error is a duplicate. Another argument with the same name is used in the context. Please check the formula in question.
Duplicate dimensions in member references.	Two or more member references in a data cube slice use the same dimension. A data cube slice can refer to only one member from each dimension. For this reason, you must remove one of the clashing member references. See “Slicing Analytic Grid Data” (Applications User’s Guide).
Error in ARGUMENTS of user function %1.	The analytic calculation engine encountered an error while parsing the ARGUMENTS section of the user function. Please check this section to make sure that it conforms to the following syntax: `ARGUMENTS(argument1, argument2...argumen⇒tN)` Dimension arguments should be declared with a prefix of $, and expression arguments should be declared with a prefix of @.
The following error occurred while preprocessing user function %1 %2.	This error occurred while processing the user function, which was referred to in another rule or user function.
Function not allowed in this context.	You used a function that is not allowed in the current context.
Functions are nested too deeply.	Functions are nested when one function is used inside another function. For example, the SIN function is nested inside the ABS function in the expression `ABS(SIN(A))`. The nesting depth refers to the number of levels of functions within functions. For example, the expression `ABS(SIN(MAX(A, B)))` has a depth of three, while `MAX(ABS(A), SIN(B))` has a depth of two, because the SIN function is not used inside the ABS function. Functions can be nested up to 32 levels deep. This error message appears when you try to nest functions beyond this limit.

Error Message	*Description*
Internal error. Uncompiled user function reference made in the formula.	The analytic calculation engine encountered an internal error while compiling rules. Please analyze and correct the user function in question.
Invalid constant.	The rule contains an invalid constant reference. Please refer to the valid constants that are supported by Analytic Calculation Engine. See Understanding the Elements of Rules.
Invalid dimension %1.	A function is referring to a dimension that does not exist. Make sure that all of the dimension names in the rule are spelled correctly.
Invalid function %1.	The rule contains a user function or built-in function name that the analytic calculation engine does not recognize. The analytic calculation engine reads a name as a function when it is followed by an opening parenthesis. For example, the expression `A + BLOOPER(X)` generates this error because Analytic Calculation Engine does not contain a function called BLOOPER.
Invalid member reference.	You incorrectly entered a member reference. When this error message occurs, check for one of these problems: • The dimension name in the member reference is not spelled correctly. • The member name in the member reference is not spelled correctly.
Invalid member reference syntax. Valid syntax is [DIMENSION:Member].	A member reference uses invalid syntax. When referring to members in rules, please make sure that the member is fully qualified with a dimension name. The valid syntax is `[DIMENSION_NAME: Member Name]`. The brackets ([ ]) are required.
Invalid number.	The current rule contains an invalid number. When this is the case, verify that: • The number does not contain any commas. • The number does not contain more than one decimal point. • If the number is negative, the minus sign precedes the number.
Syntax error.	See the Syntax Error section below.

Error Message	*Description*
Text not terminated by quote.	The rule contains a text value that does not have a closing quote. Text values must be enclosed within quotes.
The ARGUMENTS declaration must appear at the beginning of a user function.	The ARGUMENTS function should be placed at the starting block of the body of the user function. Please check the syntax of the ARGUMENTS function. See ARGUMENTS Declaration.
This argument has not been declared in the ARGUMENTS section.	You used an argument to a user function, in the body of the user function, before declaring it in the ARGUMENTS section. Please check the syntax of the ARGUMENTS function and ensure that all arguments are declared before they are used.
Unbalanced parentheses.	The rule does not contain a closing parenthesis for every opening parenthesis. For example, the expression `A + (B * C` generates this error because there should be a closing parenthesis following *C*.
Could not find the user function with the name %1.	The user function was referred to in a rule but was not found in the analytic model.
Could not find the user function rule with the name %1.	The user function that is referenced in the rule was not found in the analytic model. Please check the body of the user function.
Undefined data cube %1.	You referred to a nonexistent data cube in a formula or user function. You must create the data cube before referring to it.

Circular Reference

If a data cube's formula refers directly or indirectly to a current value of the same data cube, the analytic calculation engine generates a circular reference error. Following are some examples of circular references:

Case 1:

```
A = A + B
```

When the analytic calculation engine evaluates the formula A + B, the analytic calculation engine changes the value of A. Then, the analytic calculation engine must evaluate the formula again, using the new value of A, consequently changing the value of A again. For this reason, the analytic calculation engine must evaluate the formula again, and so on. Because the analytic calculation engine does not contain a method to exit this cycle, it refuses to accept a formula that contains a circular reference.

This formula contains a direct circular reference because A refers to itself in its own formula.

Case 2:

```
A = B + C
```

```
B = A + D
```

This case is slightly more complex, but is a result of the same issue presented in Case 1. When the analytic calculation engine evaluates the formula B + C, the analytic calculation engine changes the value of A. The analytic calculation engine must then evaluate the formula A + D, using the new value of A, consequently changing the value of B. For this reason, the analytic calculation engine must reevaluate B + C, consequently changing the value of A. For this reason, the analytic calculation engine must reevaluate A + D, and so on. These two formulas create an endless circle.

These formulas contain an indirect circular reference because neither A nor B refers to itself in its own formula. Instead, the circularity is created by the two formulas working together. The following statement describes this circularity: A depends on B, which depends on A.

Case 3:

```
A = B + C
B = D + E
D = F + G
F = A + H
```

In this case, A depends on B, which depends on D, which depends on F, which depends on A.

The analytic calculation engine traps all circular errors and does not allow you to inadvertently create circular references. Though this is the case, you may have to rethink the logic of the analytic model to ensure proper calculation. A circular reference is often the result of a logical error, which is an attempt to define something in terms of itself. If you encounter a circular reference error, you may need to step through the formulas in the analytic model definition to discover where the thinking is circular. After you find this answer, you must rework the logic to remove the circularity.

A circular reference error occurs when a data cube directly or indirectly refers to a current value of itself. On the other hand, if a data cube refers to a previous value of itself, the formula is not only valid but useful.

See PREVSELF.

See Working with Circular Formulas and Circular Systems.

Syntax Error

When you receive a syntax error, the current rule does not follow the basic guidelines for a rule. This is often the result of a typographical error. Possible violations of the rule guidelines include:

- An arithmetic operator does not have a value on both sides.

  For example, the expression `A+B+` generates a syntax error because the second plus operator does not have a value on both sides.

- Two values exist without an operator between the values.

  For example, the expression `Profit 0.50` generates a syntax error because there is no operator between the data cube reference and the number.

- The rule contains a symbol that the analytic calculation engine does not recognize.

  For example, a dollar sign ($) generates a syntax error.

- Either a function does not contain the correct number of arguments, or it contains an argument of the wrong type.

  If the syntax error occurs within a function, you may need to check the entry for that function in the built-in function reference.

  See Built-in Function Reference.

- The rule contains too many closing parentheses.

  For example, the expression `A * (B + C))` generates a syntax error because there is an extra closing parenthesis.

  Note: If there are too few closing parentheses, you receive the error *Unbalanced parentheses.*

Defining and Editing Data Cube Formulas

To define or edit a data cube formula:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. Click inside the rule bar.
5. Enter a new rule or edit the existing rule.

   See Working with the Elements of Rules.
6. Perform one of these actions:
   - Click the **Accept Changes** button to accept the changes.
   - Click the **Exit Formula Mode** button to keep the changes without validating the formula.
   - Click the **Cancel Changes** button to cancel the changes.

Defining and Editing User Functions

To define or edit a user function:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Perform one of these actions:

- To define a new user function, select **Part** > **New** > **User Function.**

  The Edit Part Name dialog box appears. Enter the name for the user function and click the **OK** button.

- To edit an existing user function, select the user function whose formula you want to edit.

4. Click inside the rule bar.
5. Enter a new rule or edit the existing rule.

   See Working with the Elements of Rules.
6. Perform one of these actions:
   - Click the **Accept Changes** button to accept the changes.
   - Click the **Exit Formula Mode** button to keep the changes without validating the formula.
   - Click the **Cancel Changes** button to cancel the changes.

Working with the Elements of Rules

This topic provides an overview of the elements of rules and discusses how to work with these rules.

Understanding the Elements of Rules

This topic discusses the various elements that are included in rules.

Built-in Functions

Many useful calculations are difficult or impossible to perform with simple arithmetic. You can perform many such calculations by using Analytic Calculation Engine's built-in functions.

Most functions have one or more arguments that supply the information that the function needs to perform the calculation. Arguments are enclosed within parentheses after a function name.

When a function contains more than one argument, the arguments are always separated by commas. For example, the following formula uses the MIN function to calculate the minimum of CASH_NEEDED and CREDIT_AVAILABLE to determine the values of the CASH_ADVANCE data cube:

```
MIN(CASH_NEEDED, CREDIT_AVAILABLE)
```

Some functions do not take any arguments because they do not require additional information to calculate a result. For example, the PI function returns the mathematical constant *pi.* Because this function does not require any information, it does not take any arguments. Nevertheless, you must still follow the function name with parentheses. For example, the following formula calculates the circumference of a circle using the PI function:

```
PI( ) * DIAMTER_OF_CIRCLE
```


The parentheses following PI indicate that the name is a function rather than a data cube reference.

Many functions have one or more optional arguments. If you leave out an optional argument, the analytic calculation engine supplies a default value for the argument. For example, the CHANGE function calculates the change between members of a dimension and takes these arguments in order:

1. The dimension for which you want to calculate the change.
2. The information for which you want to calculate the change.
3. How many members back to look.

Note: The third argument is optional; if you do not include it, the analytic calculation engine assumes you want to calculate the change from only the previous member.

For example, suppose you want to calculate the monthly change in sales. You can use the CHANGE function and leave out the third argument, as shown in the following formula:

```
CHANGE(MONTHS, SALES)
```

For each month, the analytic calculation engine calculates the change in sales from the previous month.

Now suppose you want to calculate the yearly change in sales. You can use the CHANGE function and supply 12 as the third argument, as shown in the following formula:

```
CHANGE(MONTHS, SALES, 12)
```

For each month, the analytic calculation engine calculates the change in sales from 12 previous months. To summarize:

- You must always place parentheses after a function name.
- If a function contains arguments, place the arguments inside the parentheses.
- If a function contains more than one argument, separate the arguments with commas.
- You can leave out an optional argument if the default value for the argument is satisfactory.

See CHANGE.

Conditions and Conditional Formulas

A condition is an expression that evaluates as true or false. A conditional formula returns different values for different conditions. The most simple conditional formula returns one value if a specified condition is true, and a different value if the condition is false. A complex conditional formula may return many different values based on many different conditions. These are types of conditions and conditional formulas:

- Comparison operators.
- Truth functions.
- Compound conditions.

See the Comparison Operators, Order of Precedence, and Compound Condition sections for more information.

Comparison Operators

You can compare the values of two expressions using one of Analytic Calculation Engine's comparison operators.

A comparison returns either a True value (1) or a False value (0), depending on the values of the two expressions.

Note: The analytic calculation engine always interprets a nonzero value as True and a zero value as False.

The expressions in a comparison can contain mathematical operators, parentheses, and functions, as well as data cubes and numbers. The analytic calculation engine evaluates the expressions on both sides of the comparison operator before it evaluates the truth of the comparison. Following are some examples of comparisons:

```
ADVERTISING >= 10000
```

```
ADVERTISING + PROMOTION < 0.5 * (MARKETING_EXPENSE - MARKETING_SALARIES)
```

The following table describes Analytic Calculation Engine's comparison operators.

Comparison Operator	***Example of Comparison***	***Meaning of Comparison***
=	A = B	A is equal to B.
<>	A <> B	A is not equal to B.
>	A > B	A is greater than B.
<	A < B	A is less than B.
>=	A >= B	A is greater than or equal to B.
<=	A <= B	A is less than or equal to B.

Truth Functions

A truth function is a function that returns *1* (True) or *0* (False), depending on whether the arguments of the function satisfy a condition. The analytic calculation engine uses truth functions to evaluate conditions that are too complex to express easily with comparison operators.

For example:

```
IF(FIRST(MONTH),  0, SET(&RunningTotal , &RunningTotal + THISCUBE())
```

In this example, if the current month that is calculated is the first month, the function returns 0. If the current month that is calculated is not the first month, the function returns the running total.

See FIRST, MATCH.

Logical Operators

A logical operator determines whether a condition is true. The following table describes the logical operators.

Logical Operator	*Meaning*	*Syntax*
.NOT.	Condition is not True.	`.NOT.` *Condition*
.AND.	Condition1 is True and Condition2 is True.	*Condition1* `.AND.` *Condition2*
.OR.	Condition1 is True or Condition2 is True.	*Condition1* `.OR.` *Condition2*

Compound Conditions

A compound condition tests whether some combination of conditions is true by combining two or more comparisons or truth functions using logical operators.

The analytic calculation engine evaluates the `.NOT.` operator before the `.AND.` and `.OR.` operators, and evaluates the `.AND.` and `.OR.` operators from left to right. You can override the precedence of the logical operators with parentheses, just as you can with the mathematical operators. The following table provides some examples of compound conditions.

Example of Compound Condition	*Meaning of Compound Condition*
`SALES > 50000 .AND. ADVERTISING < 10000`	Returns *True* if SALES is greater than 50000 and ADVERTISING is less than 10000.
`CASH_REMAINING < 1000 .OR. PROJECT_DONE`	Returns *True* if CASH_REMAINING is less than 1000 or if PROJECT_DONE is *True.*
`.NOT. IS_FIRST .AND. .NOT. IS_LAST`	Returns *True* if IS_FIRST is *not True* and IS_LAST is *not True.*
`.NOT. (IS_FIRST .OR. IS_LAST)`	Returns *True* if the condition (IS_FIRST or IS_LAST) is *not True.* **Note:** This condition has the same effect as the previous condition.

Predefined Constants

Analytic Calculation Engine provides several predefined constants that you can use in rules. You can use constants in the same way that you use numbers in rules. For example, you can test whether a data cube equals the constant, or you can return the constant as a result.

The following table describes predefined constants.

Predefined Constant	***Definition***
`#ALL`	Use this predefined constant as the last argument of the CHILDCOUNT or FORCHILDREN functions to return all of a dimension member's children, including grandchildren. If you do not specify a dimension member, this constant returns all of the children and grandchildren of the dimension member that is attached to the data cube that is currently being calculated. --- **Note:** You can also use the #DETAILS or #DIRECT predefined constants as the last argument for the CHILDCOUNT or FORCHILDREN functions. --- See CHILDCOUNT, FORCHILDREN.
`#BLANK`	A blank value. Use this constant to test whether a value in a data cube is blank or to return a blank value as a result.
`#DETAILS`	Use this predefined constant with trees as the last argument of the CHILDCOUNT or FORCHILDREN functions to return only the dimension members that are details. If you do not specify a dimension member, this constant returns only the details of the dimension member that is attached to the data cube that is currently being calculated. --- **Note:** You can also use the #ALL or #DIRECT predefined constants as the last argument for the CHILDCOUNT or FORCHILDREN functions. --- See CHILDCOUNT, FORCHILDREN.

`#DEFAULT`	Use this predefined constant as the last condition in a CASE function to return a default result when all other conditions are false. For example:
	
	`CASE(Condition 1 : Result 1, Condition 2⇒`
	
	`: Result 2,`
	`#DEFAULT : Default Result)`
	
	See CASE.

Predefined Constant	*Definition*
#DIRECT	Use this predefined constant with trees as the last argument of the CHILDCOUNT or FORCHILDREN functions to return a dimension member's direct children only. If you do not specify a dimension member, this constant returns only the direct children of the dimension member that is attached to the data cube that is currently being calculated. **Note:** You can also use the #ALL or #DETAILS predefined constants as the last argument for the CHILDCOUNT or FORCHILDREN functions. See CHILDCOUNT, FORCHILDREN.
#E	The value of *e* (2.7182818285), which is the base of natural logarithms.
#FALSE	A false value. Use this constant to test whether a data cube is false or to return a false value as a result.
#FORWARD	Use this predefined constant as the second argument in the FORMEMBERS function to loop through the dimension members in a forward direction. See FORMEMBERS.
#N/A	Use this predefined constant to test whether a value in a data cube is not available, or to return *N/A* as a result.
#PI	The value of (3.1415926536), which is the ratio of a circle's circumference to its diameter.
#REVERSE	Use this predefined constant as the second argument in the FORMEMBERS function to loop through the dimension members in a reverse direction. See FORMEMBERS.
#TRUE	A true value. Use this predefined constant to test whether a data cube is true or to return a true value as a result.

Mathematical Operators

This table describes Analytic Calculation Engine's operators and their order of execution.

Symbol	*Mathematical Operation*	*Order of Execution*
^	Exponentiation	1
*	Multiplication	2
/	Division	3
+	Addition	4
-	Subtraction	5

Order of Precedence

If you use more than one kind of operator in a rule, you must understand the precedence that the analytic calculation engine follows with the operators. Precedence refers to the order in which the different operators are evaluated.

For an example of precedence, the rule `3 + 2 * 4` evaluates as 11, not as 20. The analytic calculation engine performs the multiplication of 2 and 4 before it adds the number 3 because multiplication has a higher precedence than addition.

You can use parentheses to override the precedence of operators. For example, the rule `(3 + 2) * 4` evaluates as 20, because the analytic calculation engine first evaluates the operation within parentheses. You can nest parentheses to exercise more control of precedence; the operations within the inner sets of parentheses are evaluated first. For example, the analytic calculation engine calculates the rule `(8 + (3 + 2) * 4) * (6 + 7)` in the order described in this table.

Order of Execution	*Operation*	*Resulting Value*
1	3 + 2	= 5
2	5 * 4	= 20
3	8 + 20	= 28
4	6 + 7	= 13
5	28 * 13	= 364

The analytic calculation engine performs the multiplication of 5 * 4 before the addition of 8. The analytic calculation engine performs multiplication before addition unless you override this order of execution with parentheses.

Note: If you use parentheses, you must balance each opening parenthesis with a closing parenthesis. If you do not balance the parentheses, the analytic calculation engine generates an Unbalanced parentheses error. When this situation occurs, you must correct the rule.

See Understanding Design Time Rule Error Messages.

Values

A value is a number or a text string. For example, the NET_PRESENT_VALUE data cube contains this rule: NPV(MONTHS, ANNUAL_DISCOUNT_RATE / 12, NET_REVENUE_BY_PRODUCT). In this rule, the value is 12.

Data Cube References

Use a data cube reference to refer to a specific data cube. For example, you can use data cube references to multiply the values of two data cubes and place the calculation totals in a result data cube. Using this example, the PROD_SALES data cube contains the following rule: *UNIT_COST * UNITS_SOLD*.

Member References

Use a member reference to refer to a dimension member to access its data or to perform a calculation. Use the following syntax to refer to a member:

```
[DIMENSION_NAME:Member]
```

For example, you could use this member reference to refer to the Hard Drives member from the PRODUCTS dimension:

```
[PRODUCTS:Hard Drives]
```

If an aggregate member and detail or leaf member share the same name, use the following syntax to reference the desired member:

- `[DIMENSION_NAME:NODE.Member]`

  Access an aggregate member.

- `[DIMENSION_NAME:DETAIL.Member]`

  Access a detail or leaf member.

Note: Navigation related functions such as PREV, NEXT, and PREVSELF operate on detail member names that are persisted in the main record. These functions do not use trees to determine the order of members.

See Understanding Dimension Members, Logic for Determining the Order of Members.

Blank Member References

You reference blank members in user functions by using the MBR2TEXT or TEXT2MBR built-in function with this string: ″ ″ (two quotation marks). Do not include spaces between the quotation marks. For example:

```
AT(Product, TXT2MBR(""), SALES)
```

Or

```
IF(MBR2TXT(Product) = "", X, Y)
```

Note: When blank members are mapped to date fields, they are written to the database as values of 1/1/1900.

See Types of Dimension Members.

Variables

When referencing variables in rules, you must always begin the variable reference with the *&* symbol, followed by the variable name.

Note: The variable name can only contain letters or numbers; it cannot contain spaces.

To set a value to a variable, use the following:

```
&Index := 1
```

The following formula sets the first character of an account number to a variable, and then uses that variable to set the account category:

```
&AcctCode := LEFT(MBR2TEXT(ACCOUNTS), 1);
 CASE(&AcctCode = "1" : [ACCT_CAT:Assets],
      &AcctCode = "2" : [ACCT_CAT:Liabilities],
      &AcctCode = "3" : [ACCT_CAT:Income],
      #DEFAULT : [ACCT_CAT: Expense]
 )
```

Setting the value to a variable makes it unnecessary to repeat the expression for each condition of the CASE function, or to create an intermediate cube to hold the account code.

You can increment or decrement a variable with the INC statement:

```
INC(&Index);
DEC(&Index);
INC(&Profit, REVENUE);
DEC(&Profit, EXPENSE)
```

The lifetime of a variable is a single evaluation of the rule; the value of a variable is not preserved across multiple evaluations.

Inserting a Built-in Function into a Rule

To insert a built-in function into a rule:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.

2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. Select the place in the rule where you want to add the function.
5. Click the **Paste Built-in Function** button.

   The Choose Built-in Function drop-down list box appears.
6. Use the scroll bar to scroll through the list of built-in functions.
7. Click the desired built-in function.

   Analytic Calculation Engine pastes the built-in function and argument names into the rule bar.
8. For each argument:
   a. Highlight the argument.
   b. Replace the highlighted argument with the argument value.
9. Complete your work on the rule, and then:
   - Click the **Accept Changes** button to accept the changes.
   - Click the **Exit Formula Mode** button to keep the changes without validating the rule.
   - Click the **Cancel Changes** button to cancel the changes.

Related Links

Built-in Function Reference

Inserting a User Function into a Rule

To insert a user function into a rule:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. Select the place in the rule where you want to add the user function.
5. Click the name of the user function in the part browser.

   Analytic Calculation Engine pastes the user function into the rule bar.

Note: If you enter a user function name that does not exist, the analytic calculation engine returns an error when validating the analytic model.

Inserting a Numeric Value or Text Value into a Rule

To insert a numeric or a text value into a rule:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. To insert a numeric value in a rule, enter the value (for example, 12).

   To use a text value in a rule, enter the value and enclose it in double quotes (for example, "Smith").

Note: PeopleSoft recommends that you do not enter an assumption directly into a rule. Instead, you should create a data cube for the assumption and refer to the data cube in the formula. For example, do not calculate TAXES by multiplying INCOME by 0.38. Instead, create a data cube called TAX_RATE and enter 0.38 as its value. Then calculate TAXES by multiplying INCOME by TAX_RATE. Performing the procedure in this fashion simplifies the process of changing the assumptions and makes the analytic model easier to understand and audit.

Inserting a Data Cube Reference into a Rule

To insert a data cube reference into a rule:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. Place the cursor at the location of the rule into which you want to insert the data cube reference.
5. Perform one of these actions:
   a. In the part browser, click on the data cube to which you want to refer.
   b. Enter the name of the data cube.

      Note: If you enter a data cube reference for a data cube that does not exist, the analytic calculation engine returns an error when validating the analytic model.

      ---

   c. Select **Edit** > **Paste Cube Name** and click on the data cube to which you want to refer.

Inserting a Dimension Reference into a Rule

To insert a dimension reference into a rule:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.

2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. Place the cursor at the location of the rule into which you want to insert the dimension reference.
5. Perform one of these actions:
   a. In the part browser, click the dimension to which you want to refer.
   b. Enter the name of the dimension.

      Note: If you enter a dimension reference for a dimension that does not exist, the analytic calculation engine returns an error when validating the analytic model.

   c. Select **Edit** > **Paste Dimension** and click on the dimension to which you want to refer.

Inserting a Dimension Member Reference into a Rule

This topic discusses how to:

- Enter a member reference into a rule.
- Refer to one slice of a data cube.

Entering a Member Reference into a Rule

To enter a member reference into a rule:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. Place the cursor at the position in the rule where you want to enter a member reference.
5. Select **Edit** > **Paste Member Ref...**

   The Choose Member Reference dialog box appears.
6. Click the dimension for which you want to enter a member reference.

   The dimension and a generic member reference appears in the rule bar.
7. Highlight the word *member*.
8. Replace the word *member* with the name of the dimension member.

Referring to One Slice of a Data Cube

When you want to access particular values within a data cube, use member references to refer to a slice of the data cube.

To refer to one slice of a data cube:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define or edit.
4. Enter the data cube in the rule.

   For example, SALES, which uses the MONTHS, PRODUCTS, and REGIONS dimensions.
5. Enter a member reference.

   For example, the East region from the REGIONS dimension.

   The rule bar now displays `SALES [REGIONS.East]`. This rule returns SALES for the East region for all PRODUCTS and all MONTHS.
6. Repeat step 3 to make the slice as small as you want.

   A single value from the data cube is the smallest possible slice.

   For example, to access SALES for the East region for the Hard Drives product for 2004/03, use the following rule:

```
SALES [REGIONS.East] [PRODUCTS.Hard Drives] [MONTHS.2004/03]
```

Related Links

"Slicing Analytic Grid Data" (Applications User's Guide)

Inserting a Blank Line into a Rule

You can insert a blank line into a rule to enhance legibility. To insert a blank line into a rule, press **Ctrl** + **Enter.**

Inserting a Comment into a Rule

To insert a comment into a rule, use the symbols << and >>. This is an example of a comment in a rule:

```
<< Loop through all products >>
```

Performing Exceptions to the Rule

This topic provides an overview of performing exceptions to the rule.

Understanding Exceptions to the Rule

A typical rule contains a formula for an entire data cube that the analytic calculation engine uses to calculate every value in the data cube. If you want some values of a data cube to calculate in a different manner than other values, you must create an exception to the rule. You can create exceptions to:

- Perform different calculations for different members.
- Perform different calculations for different groups of members.

Create Different Calculations for Different Members

This topic provides an overview of the calculation of only one member and the calculation of more than one member and discusses how to:

- Create a calculation for only one member.
- Create a calculation for more than one member.

Understanding the Calculation of Only One Member

The following example describes the reason for and process of creating a special calculation for one member.

Suppose your company must allocate the Administration department's expense equally to all of the other departments. To ensure proper allocation, the Administration department requires a different calculation than the other departments. To create this allocation, you must back out the expense for Administration and divide that expense equally among the other departments.

Your company's analytic model contains data cubes called EXPENSE and ADMIN_ALLOCATION. The DEPARTMENTS dimension is attached to both data cubes. Create the following formula to calculate ADMIN_ALLOCATION:

```
IF([DEPARTMENTS:Administration], - EXPENSE, EXPENSE[DEPARTMENTS:Administration] /
(NUMMEMBERS(DEPARTMENTS) - 1))
```

The formula uses the IF function to calculate one result if a condition is true, and another result if the condition is false. Here is how the formula works:

The analytic calculation engine uses the `[DEPARTMENTS:Administration]` member reference to check whether Administration is the department that is being calculated.

- If Administration is the department that is being calculated, the formula returns minus EXPENSE, backing out the expense for Administration.
- If Administration is not the department that is being calculated, the formula returns the expense for Administration divided by the number of departments minus one.

  In other words, the formula divides the Administration expense equally among the other departments.

Note: The formula uses the data cube slice `EXPENSE[DEPARTMENTS:Administration]` to refer to the expense for Administration. The NUMMEMBERS function returns the number of members in the specified dimension.

Understanding the Calculation of More Than One Member

The following formula provides an example of a calculation for more than one member. The formula returns one result for Administration, another result for Data Processing, and a third result for all other departments:

```
CASE([DEPARTMENTS:Administration] : ADMINISTRATION_RESULT,
```



```
[DEPARTMENTS:Data Processing] : DATA_PROCESSING_RESULT,
#DEFAULT : RESULT_FOR_ALL_OTHER_DEPARTMENTS)
```

Creating a Calculation for Only One Member

To create a calculation for only one member:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define.
4. Define a formula for the result data cube.
5. Enter *IF* and an opening parenthesis.

   You are using the IF function to return different results, depending on a condition.

   See IF.
6. Insert the member reference for the exceptional member.

   Note: When you use a member reference as a condition, it returns *True* if the analytic calculation engine is calculating values for that member; otherwise, it returns *False.*

   See Inserting a Dimension Member Reference into a Rule.
7. Enter a comma, and then enter the result that should be returned if the exceptional member is being calculated.
8. Enter another comma, and then enter the result that should be returned if one of the nonexceptional members is being calculated.
9. Enter a closing parenthesis.

   Note: To perform the same calculation for several members, combine two or more member references with `.OR.` operators. For example: `[DEPARTMENTS:Administration] .OR. [DEPARTMENTS:Data Processing].`

Creating a Calculation for More Than One Member

To create a calculation for more than one member:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the data cube whose formula you want to define.
4. Define a formula for the result data cube.
5. Enter CASE and an opening parenthesis.

You are using the CASE function to evaluate a Condition:Result pair for each special case.

See CASE.

6. Enter a Condition:Result pair for each special calculation:
   a. Insert a member reference for one of the members in the dimension.

      For example: `[DEPARTMENTS:Administration]`.

      This condition tests whether results are being calculated for the specified member.
   b. Enter a colon to separate the condition from the result.
   c. Enter the appropriate result for the specified member.
   d. Enter a comma.
7. Perform these steps to enter a final Condition:Result pair to return a result for all other members in the dimension:
   a. Enter `#DEFAULT` as the condition.

      `#DEFAULT` instructs the function to return the final result for all other members.
   b. Enter a colon to separate the condition from the result.
   c. Enter the result for all other members in the dimension.
   d. Enter a closing parenthesis.

Creating Different Calculations for Different Groups of Members

You may want to calculate a data cube in different ways for different groups of members. To perform different calculations for different groups of members:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Create an association data cube that associates each member with a group.

   See Creating Association Data Cubes.
4. Define a formula for the result data cube.
5. Use the CASE function to evaluate two or more Condition:Result pairs.

   Perform the following steps for each Condition:Result pair:

   See CASE.

a. Use a member reference to compare the association data cube to one of the members in the group dimension.

   This example is a formula for the INVEST_TYPE data cube:

   ```
   [TYPES:Stock]
   ```

   See Inserting a Dimension Member Reference into a Rule.

b. Enter a colon to separate the condition and result.

c. Enter the appropriate result for that group.

d. To add another Condition:Result pair, enter a comma; otherwise, enter a closing parenthesis.

Example: Creating Different Calculations for Different Groups of Members

Suppose that you create an analytic model to track your investments in stocks, bonds, and rental properties, and you want to know your monthly income. Because the income for stocks, bonds, and rental properties is calculated differently, you need to perform different calculations for different groups of these investments.

Create a data cube that associates each investment with an investment type. Suppose the dimension of investment types is called TYPES, and the association data cube is called INVEST_TYPE. You can calculate the investment income for each investment as follows:

```
CASE(INVEST_TYPE = [TYPES:Stock] : NUMBER_OF_SHARES * DIVIDENDS_PER_SHARE,
INVEST_TYPE = [TYPES:Bond] : BOND_RATE * BOND_AMOUNT / 12,
INVEST_TYPE = [TYPES:Rent] : MONTHLY_RENT)
```

See Creating Association Data Cubes.

The CASE function evaluates multiple conditions and returns the result for the first true condition. Each Condition:Result pair is separated by a comma.

See CASE.

In the preceding formula, the CASE function compares the invest type for an investment to each member in the TYPES dimension. The formula uses a member reference (for example, `[TYPES:Stock]`) to refer to each member. When the CASE function finds the matching member from the TYPES dimension, it returns the corresponding result. For example, if the invest type for an investment is Bond, the formula returns `BOND_RATE * BOND_AMOUNT / 12`.

Working with Circular Formulas and Circular Systems

This topic provides overviews of circular formulas, circular systems and recursive systems, recursive system resolution, and circular system resolution, and discusses how to change circular formula and circular system options.

See “Circular Reference” in Understanding Design Time Rule Error Messages.

Understanding Circular Formulas

When a data cube's formula refers either directly or indirectly to that same data cube, it is considered to be a circular formula.

Note: The analytic calculation engine determines—on the data cube level—whether formulas are circular. However, the analytic calculation engine resolves circular systems and recursive systems on the cell level.

Direct Circular Formulas

This is an example of a direct circular formula for the SALES data cube:

```
SALES + SALES_GROWTH
```

This formula states that sales equals sales plus the sales growth.

It is a direct circular formula because the data cube's formula refers directly to that same data cube.

Indirect Circular Formulas

In an indirect circular formula, a data cube's formula refers indirectly to that same data cube, as in this example:

- Formula for BONUS data cube:

  ```
  BONUS_PERCENTAGE * NET_INCOME
  ```

- Formula for EXPENSE data cube:

  ```
  SALARY + BONUS
  ```

- Formula for NET_INCOME data cube:

  ```
  INCOME - EXPENSE
  ```

In this example, none of the data cubes refer directly to themselves. However, each data cube refers indirectly to itself by means of the other data cubes:

- BONUS refers to NET_INCOME.
- NET_INCOME refers to EXPENSE.
- EXPENSE refers to BONUS.

Understanding Circular Systems and Recursive Systems

When the analytic calculation engine determines—on the *data cube level*—that a circular formula exists, it analyzes the calculation conditions of the cells within the circular formula to determine whether the cells are dependent on those same cells for their values. If so, these cells either create a *recursive system* or a *circular system*.

In a recursive system, the values of the cells are *not* dependent on the values of those same cells.

In a circular system, the values of the cells *are* dependent the values of those same cells.

Understanding Recursive System Resolution

The analytic calculation engine resolves recursive systems immediately without using the process of iteration.

To understand the process of how the analytic calculation engine determines and resolves recursive systems, consider the following formula for the SALES data cube:

```
PREVSELF(MONTHS) + SALES_GROWTH
```

The analytic calculation engine determines that this is a recursive system by performing the following steps:

1. The analytic calculation engine determines that this is a circular formula because the PREVSELF built-in function, which refers to the SALES data cube, exists within the formula.
2. The analytic calculation engine analyzes the calculation conditions of the cells within this formula and determines that these cells create a recursive system, because the cells within this formula are not dependent on the values of those same cells.

The analytic engine then resolves this recursive system immediately without iteration.

Understanding Circular System Resolution

The analytic calculation engine uses the process of *iteration* to attempt to resolve all of an analytic model's circular systems. You set the iteration parameters by using the General tab of the analytic model's properties. If the cells converge on a solution within the iteration parameters, the circular system is resolved. If the cells do not converge on a solution within the iteration parameters, the analytic calculation engine returns an error. All cells within the circular system remain unresolved.

Note: Because you enable or disable iteration for *all* circular systems, you cannot enable or disable iteration for a *particular* circular system.

When you enable the resolution of circular systems through iteration, you must select one of the following iteration options:

- **Maximum number of iterations**

  When the values of the cells have not changed more than the specified maximum amount, the circular system is considered to be resolved.

- **Maximum change in values**

  When the analytic calculation engine has calculated the cells within the circular system for the specified maximum number of times, the circular system is considered to be resolved.

Example of Resolving a Circular System

In this example, an analytic model contains the following formulas (for simplicity, assume that each data cube contains only a single cell):

- Formula for BONUS data cube:

  ```
  BONUS_PERCENTAGE * NET_INCOME
  ```


- Formula for EXPENSE data cube:

```
SALARY + BONUS
```

- Formula for NET_INCOME data cube:

```
INCOME - EXPENSE
```

First, the analytic calculation engine determines that this is an indirect circular formula because the data cubes' formulas refer indirectly to those same data cubes.

Next, the analytic calculation engine analyzes the calculation conditions of the cells within this indirect circular formula and determines that these cells create a circular system. This is because the cells within this formula are dependent on the same values of those same cells.

Assuming that the value for the BONUS_PERCENTAGE cell = 5, the value for the INCOME cell = 10000, and the value for the SALARY cell = 6000, then the circular system is resolved when the analytic calculation engine iterates until it returns these solutions:

- Cell for BONUS = 190.48.
- Cell for EXPENSE = 6,190.48.
- Cell for NET_INCOME = 3,809.52.

If you plug these values into the preceding formulas, each formula is true: the left side of the formula is (almost) equal to the right side of the formula. Once this occurs, the circular system is considered to be resolved.

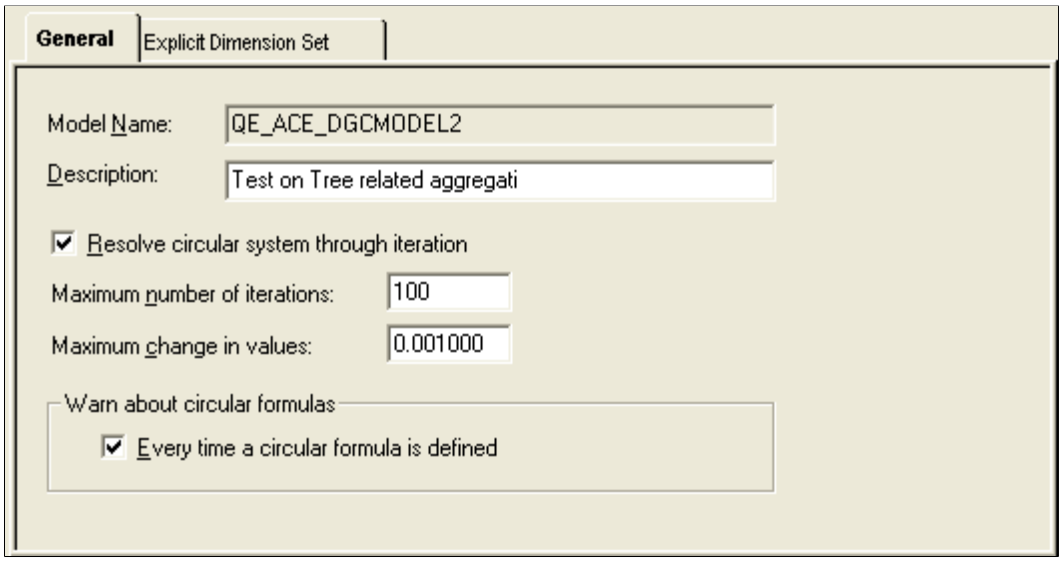
Changing Circular Formula and Circular System Options

Use the Analytic Model - General tab to change circular formula and circular system options.

Navigation:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open an analytic model definition.
3. Select the name of the analytic model in the part browser.

This example illustrates the fields and controls on the Analytic Model - General tab. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Description	Enter a description of the analytic model. **Note:** This field pertains to the analytic model as a whole, not to circular formula options.
Resolve circular system through iteration	Select to attempt to resolve all of an analytic model's circular systems through iteration. By default, this option is disabled. See Changing Circular Formula and Circular System Options.
Maximum number of iterations	Enter the number of iterations in which the analytic calculation engine is to resolve circular systems. By default, the maximum number of iterations is 100. If the analytic calculation engine cannot resolve a circular system during this number of iterations, the analytic calculation engine returns an error. **Note:** You must select the **Resolve circular system through iteration** check box to activate this option.

Field or Control	*Description*
Maximum change in values	Enter the maximum change in values. By default, the maximum change in values is 0.001000. A circular system is considered to be resolved when the values of its cells do not change more than the specified maximum change. If you enter a smaller value, the solution is more accurate but may require a longer calculation time. If you enter a larger value, the solution not as accurate but requires a shorter calculation time. **Note:** You must enable the **Resolve circular system through iteration** check box to activate this option.
Warn about circular formulas	*Every time a circular formula is defined:* Select for the analytic calculation engine to provide a circular formula warning every time a circular formula is defined.

Chapter 11

Using Built-in Functions in Analytic Models

Built-in Function Reference

This topic discusses the build-in functions that are used in an analytic model's rules and user functions.

ABS

Syntax

ABS *(Data)*

Description

The ABS function returns the absolute (positive) value of *Data.*

Returns

The absolute (positive) value of *Data.*

Example

The following examples employ the ABS built-in function:

- `ABS(5)` returns *5.*
- `ABS(-5)` returns *5.*
- `ABS(0)` returns *0.*

ACOS

Syntax

ACOS *(Data)*

Description

The ACOS function returns the arc-cosine of *Data.* The result is the angle (in radians) whose cosine equals *Data.*

The value of *Data* must fall between 1 and 1; otherwise, ACOS returns an error value.

To convert from radians to degrees, multiply by 180 / PI( ). (The PI function returns the value of PI.)

Returns

The arc-cosine of *Data.*

Example

The following examples employ the ACOS built-in function:

- `ACOS(0.5)` returns *1.0471975512* (angle in radians).
- `ACOS(0.5) * 180 / PI( )` returns *60* (angle in degrees).
- `ACOS(SQRT(2) / 2)` returns *0.7853981634* (angle in radians).
- `ACOS(SQRT(2) / 2) * 180 / PI( )` returns *45* (angle in degrees).

ARGUMENTS Declaration

Syntax

ARGUMENTS*(argument1), argument2... argumentN*

Description

The ARGUMENTS declaration passes values to functions within a user function.

Use the following guidelines to make user functions more powerful by giving them arguments:

- Enter an ARGUMENTS declaration at the beginning of the rule, followed by an opening parenthesis.
- Enter any dimension arguments next, separated by commas.

  A dimension argument always begins with a `$`, as in `$Dim`. Unlike the built-in functions, which never have more than one dimension argument, user functions can have multiple dimension arguments.
- Include any expression arguments next, separated by commas.

  An expression argument always begins with a `@`, as in `@Expr`. To give the argument a default value, follow it with `:=` and any valid expression.

  Note: All optional arguments (that is, those with default values) must appear after all required arguments.

- Enter a closing parenthesis and a semicolon.

Example

```
ARGUMENTS($Dim, @ExprToLookup, @Condition, @Direction :=
#FORWARD);FORMEMBERS($Dim, @Direction,
IF(@Condition, RETURN(@ExprToLookup)));RETURN(0)
```


ASC

Syntax

ASC*(Text)*

Description

The ASC function returns the first character of the Text argument to its ASCII equivalent number (for example, a number between 0-255). Use this function to convert a character into its ASCII value.

Returns

The first character of the Text argument to its ASCII equivalent number.

Example

For a cube formatted as a number, `ASC("ABC")` returns the *65*.

ASIN

Syntax

ASIN*(Data)*

Description

The ASIN function returns the arc-sine of *Data.* The result is the angle (in radians) whose sine equals *Data.*

The value of *Data* must fall between 1 and 1; otherwise, ASIN returns an error value.

To convert from radians to degrees, multiply by 180 / PI( ). (The PI function returns the value of PI.)

Returns

The arc-sine of *Data.*

Example

The following examples employ the ASIN built-in function:

- `ASIN(0.5)` returns *0.5235987756* (angle in radians).
- `ASIN(0.5) * 180 / PI( )` returns *30* (angle in degrees).
- `ASIN(SQRT(2) / 2)` returns *0.7853981634* (angle in radians).
- `ASIN(SQRT(2) / 2) * 180 / PI( )` returns *45* (angle in degrees).

AT

Syntax

AT*(Dimension, Member, Data)*

Description

The AT function looks up the value of *Data* for a particular member in a dimension.

You can use the AT function in the following ways:

- You can look up a value at a particular *position* in the dimension.
- You can look up a value for a particular member by *name.*
- You can associate members in one dimension with members in another dimension, and then look up an associated value for each member in the first dimension.

  For example, you can associate each *employee* with a job, and then look up the *job salary* for each employee.

Looking Up a Value at a Position

To look up a value at a particular position, use the position number for the member argument. You can also use an expression that returns the position number.

For example, to look up the value of SALES for the *first* member in the PRODUCTS dimension, use the following formula:

```
AT(PRODUCTS, 1, SALES)
```

To look up the value of SALES for the *last* member in the PRODUCTS dimension, use the following formula:

```
AT(PRODUCTS, NUMMEMBERS(PRODUCTS), SALES)
```

This formula works because the NUMMEMBERS function returns the number of members in the Products dimension, which is the position of the last member.

See NUMMEMBERS.

Looking Up a Value for a Member by Name

To look up a value for a particular member by name, use a member reference for the member argument.

For example, the following formula returns UNIT_COST divided by UNIT_PRICE for the Monitors product:

```
AT(PRODUCTS, [PRODUCTS:Monitors], UNIT_COST / UNIT_PRICE)
```

You can achieve the same result using member references after the data cube names, as follows:

```
UNIT_COST [PRODUCTS:Monitors] / UNIT_PRICE [PRODUCTS:Monitors]
```


To evaluate a complex expression for a single member, the AT function is more concise because you are not required to repeat the member reference for every data cube.

Looking Up an Associated Value

You can associate members in one dimension with members in another dimension, and then look up an associated value for each member in the first dimension. For example, suppose that you want to associate each employee with a job, and then look up the job salary for each employee. To do this, perform the following:

1. Create a dimension called *JOBS.*
2. Create a data cube called *EMPLOYEE_JOB.*

   Format this data cube as a member of the JOBS dimension.
3. Create a dimension called *EMPLOYEE.*

   Attach this dimension to the EMPLOYEE_JOB data cube.
4. Create a data cube called *SALARY_BY_JOB,* which contains the salary for each job.
5. Create a data cube called *EMPLOYEE_SALARY.*

   Create the following formula for this data cube:

   Note: You can look up the salary for each employee by using the name of the association data cube as the member argument.

   ```
   AT(JOBS, EMPLOYEE_JOB, SALARY_BY_JOB)
   ```

   For each employee, the formula looks up the number in SALARY_BY_JOB that is *at* the member indicated by EMPLOYEE_JOB.

Pushing Down Parent Member Data

The following are examples of user functions that push down parent member data:

- `AT (DIMENSION, Parent(DIMENSION), THISCUBE() * 0.2)`
- `AT (DIMENSION, Parent(DIMENSION), THISCUBE() / CHILDCOUNT(DIMENSION, #DIRECT))`

ATAN

Syntax

ATAN*(Data)*

Description

The ATAN function returns the arc-tangent of *Data.* The result is the angle (in radians) whose tangent equals *Data.*

To convert from radians to degrees, multiply by 180 / PI( ). The PI function returns the value of PI.

Returns

The arc-tangent of *Data.*

Example

The following examples employ the ATAN built-in function:

- `ATAN(0.5)` returns *0.463647609* (angle in radians).
- `ATAN(0.5) * 180 / PI( )` returns *26.5650511771* (angle in degrees).
- `ATAN(1)` returns *0.7853981634* (angle in radians).
- `ATAN(1) * 180 / PI( )` returns *45* (angle in degrees).

BREAK

Syntax

BREAK()

Description

The BREAK function causes an immediate break out of the current loop.

Example

```
SET(&Value, 1);
WHILE(&Value < THE_ABSOLUTE_MAXIMUM,
 SET(&Value, &Value * 2);
 IF(&Value = ENOUGH_ALREADY, BREAK());
 INC(&Value)
);
IF(&Value > ENOUGH_ALREADY, "More than enough", "Just right")
```

You normally use the BREAK function within an IF function to break out of a loop when a specified condition is achieved. To return `Just right` from the formula, `ENOUGH_ALREADY` must contain a value from the sequence 2, 6, 14, 30, and so on.

CASE

Syntax

CASE*(Condition A : Result A, Condition B : Result B {,...})*

Description

The CASE function returns the *Result* that corresponds to the first true *Condition,* if none of the conditions is true, it returns zero.

Returns

The *Result* that corresponds to the first true *Condition;* if none of the conditions is true, it returns zero.

Example

Suppose a company awards its salespeople the following commissions:

- A 10 percent commission if their sales are at least 50,000 USD.
- An 8 percent commission if their sales are at least 30,000 USD.
- A 5 percent commission if their sales are at least 15,000 USD.

You can calculate the commission rate for a salesperson with the following formula:

```
CASE(SALES >= 50000 : 0.10, SALES >= 30000 : 0.08,
SALES >= 15000 : 0.05)
```

If SALES is 45000, this formula returns *0.08.* Notice that the CASE function returns the result for the first true condition, even if some of the remaining conditions are true.

The above formula returns zero if SALES is less than 15000. Suppose that the company awards a 3 percent commission on all sales under 15,000 USD. You can model this with the following formula:

```
CASE(SALES >= 50000 : 0.10, SALES >= 30000 : 0.08,
SALES >= 15000 : 0.05, #DEFAULT : 0.03)
```

The last condition (#DEFAULT) is always equivalent to TRUE, so the CASE function returns *0.03* if SALES is less than 15000. If you want the CASE function to return a default value other than zero, use #DEFAULT as the last condition.

CHANGE

Syntax

CHANGE*(Dimension, Data, {Count})*

Description

The CHANGE function returns the difference between the value of *Data* for the member being calculated and the value of *Data* for *Count* members back. If *Count* is omitted, it is assumed to be 1.

Example

Suppose you wish to calculate the monthly and yearly change in a data cube called SALES. If SALES uses a dimension called MONTHS, use the following formula to calculate the MONTHLY_CHANGE data cube:

```
CHANGE(MONTHS, SALES)
```

Because the *Count* argument is omitted, the program assumes it to be 1. Consequently, the program calculates the change in SALES from the previous month to the month being calculated.

Calculate the YEARLY_CHANGE data cube by using 12 for the third argument:

```
CHANGE(MONTHS, SALES, 12)
```

This formula calculates the change in SALES from 12 months ago to the month being calculated.

CHILDCOUNT

Syntax

CHILDCOUNT*(Dimension, {#DIRECT/#ALL/#DETAILS, {Parent Member}})*

Description

The CHILDCOUNT function returns the number of a Parent member's children. This function takes the following one required argument and two optional arguments:

- *Dimension*: The dimension to use.
- For the second optional argument, select from these predefined constants:
  - *#DIRECT*
  - *#ALL*
  - *#DETAILS*

    See Understanding the Elements of Rules.
- *Parent Member*: This is an optional argument.

  If you do not use this optional argument, the function applies to the member that is currently being evaluated.

Returns

The number of a parent member's children.

Example

```
CHILDCOUNT (Region, #DIRECT, [Region:All_regions])
```

Related Links

Pushed Down Data

CHR

Syntax

CHR*(Number)*

Description

The CHR function returns the equivalent ASCII character of the number argument. The number must be in range from 0 to 255; otherwise, an invalid type error with be thrown.

Returns

The equivalent ASCII character of the number argument.

Example

For a cube formatted as text, `CHR(65)` returns the character *A*.

CONSOL

Syntax

CONSOL*(Dimension, Data)*

Description

The CONSOL function returns the value of *Data* for the total member of *Dimension.*

Returns

The value of *Data* for the total member of *Dimension.*

Example

Suppose an analytic model contains a data cube called *SALES* that uses a dimension called *PRODUCTS.* Use the following formula to calculate each product's sales as a percentage of total sales:

```
SALES / CONSOL(PRODUCTS, SALES)
```

This formula divides each product's sales by the consolidated value for SALES.

COS

Syntax

COS*(Data)*

Description

The COS function returns the cosine of *Data,* where *Data* represents an angle in radians.

To convert from degrees to radians, multiply by PI( ) / 180. The PI function returns the value of PI.

Example

The following examples employ the COS built-in function:

- `COS(PI( ) / 3)` returns *0.5* (cosine of PI / 3 radians).
- `COS(PI( ) / 2)` returns *0* (cosine of PI / 2 radians).
- `COS(45 * PI( ) / 180)` returns *0.7071067812* (cosine of 45 degrees).

CUBEID

Syntax

CUBEID*(Cube)*

Description

The CUBEID function returns the internal ID of the cube in the analytic calculation engine. Note that the actual ID for the cube may vary in the analytic calculation engine when the model has been changed. For example, when a part is added or deleted. Do not use absolute numbers to compare the return of the CUBEID function. The CUBEID function may be useful if you have a generic user function and you want to pass different data cubes as expression arguments.

Returns

The internal ID of the cube in the analytic calculation engine.

Example

```
IF(CUBEID(@MyCube) = CUBEID(REVENUE),
SPECIAL_CONDITION_CALCULATION, DEFAULT_CALCULATION)
```

This is an example of incorrect usage of the CUBEID function:

```
IF( CUBEID(@MYCUBE) = 512,
SPECIAL_CONDITION, DEFAULT_CONDITION)
```

CUMAVG

Syntax

CUMAVG*(Dimension, Data, {Count})*

Description

The CUMAVG function returns the cumulative average of *Data* for the last *Count* members of *Dimension.* If *Count* is omitted, CUMAVG returns the cumulative average of all members up to the member being calculated.

Example

The following examples provide uses of the CUMAVG function:

Example 1

Suppose an analytic model contains a data cube called *SCORES* that uses a dimension called *TESTS.* Use the following formula to compute the average of all test scores up to the test being calculated:

```
CUMAVG(TESTS, SCORES)
```

This formula calculates Cum_Avg_Score for Test 2 by averaging the scores for Test 1 and Test 2; it calculates Cum_Avg_Score for Test 3 by averaging the scores for Test 1, 2, and 3; and so on. Because the third argument is omitted, the function averages the scores for all tests up to the test being calculated.

Example 2

Suppose an analytic model contains a data cube called *SALES* that uses a dimension called *MONTHS.* Compute the average sales for the last six months as follows:

```
CUMAVG(MONTHS, SALES, 6)
```

Note that for the first five months, the CUMAVG function cannot look back six months, because this would go back before the first month in the MONTHS dimension. The program solves this problem by averaging the sales for all months up to the month being calculated. After the first five months, the function averages the sales for the six months up to the month being calculated.

CUMSUM

Syntax

CUMSUM*(Dimension, Data, {Count})*

Description

The CUMSUM function returns the cumulative sum of *Data* for the last *Count* members of *Dimension.* If *Count* is omitted, CUMSUM returns the cumulative sum of all members up to the member being calculated.

Returns

The cumulative sum of *Data* for the last *Count* members of *Dimension.*

If *Count* is omitted, CUMSUM returns the cumulative sum of all members up to the member being calculated.

Example

Suppose an analytic model contains a data cube called *PROFIT* that uses a dimension called *MONTHS.* Use the following formula to calculate the cumulative profit for all months up to the month being calculated:

```
CUMSUM(MONTHS, PROFIT)
```


Use the following formula to calculate the cumulative profit for the three months up to the month being calculated:

```
CUMSUM(MONTHS, PROFIT, 3)
```

DAVG

Syntax

DAVG*(Dimension, {Data}, {Condition})*

Description

The DAVG function returns the average of *Data* for the members in *Dimension* where *Condition* is True. If *Condition* is omitted, DAVG returns the average of *Data* for all members in *Dimension.* If *Data* is omitted, DAVG returns the average of the data cube being calculated, for all members up to the current member in *Dimension.*

Example

The following examples provide uses of the DAVG function:

Example 1

Suppose an analytic model contains a data cube called *ADVERTISING_BY_PRODUCT* and a data cube called *UNITS_SOLD*. Both data cubes use a dimension called *PRODUCTS.* Use the following formula to calculate the average units sold for all products:

```
DAVG(PRODUCTS, UNITS_SOLD)
```

The DAVG function does not include a condition, so the function averages UNITS_SOLD for all members in the PRODUCTS dimension. Use the following formula to calculate the average units sold for all products with advertising of at least USD 10,000:

```
AVG(PRODUCTS, UNITS_SOLD, ADVERTISING_BY_PRODUCT >= 10000)
```

In this case, the function averages UNITS_SOLD only for the products where ADVERTISING_BY_PRODUCT is greater than or equal to 10000.

Example 2

You can make the analysis more flexible by creating a dimension called *RANGES* and attaching it to *AVG_UNITS_SOLD*. Define a new data cube called *AD_COST_MIN* that uses the RANGES dimension. Each number in AD_COST_MIN defines the minimum value for the range, while the next number defines the upper limit for the range. Calculate the average units sold for each range as follows:

```
DAVG(PRODUCTS, UNITS_SOLD, ADVERTISING_BY_PRODUCT >=
AD_COST_MIN .AND. ADVERTISING_BY_PRODUCT <
NEXT(RANGES, AD_COST_MIN))
```

For each range, the formula averages only those products whose advertising cost is greater than or equal to the current AD_COST_MIN and less than the next AD_COST_MIN. (The NEXT function returns AD_COST_MIN for the next member in the RANGES dimension.)

Related Links
NEXT

DAY

Syntax

DAY*({Date})*

Description

The DAY function returns the day of the specified date. If *Date* is omitted, DAY returns the day of the calculation date.

Example

If A = 2004/03/15 and B = 2005/06/22, then DAY(A) returns *15* and DAY(B) returns *22*.

Now suppose an analytic model contains a data cube called *DAY_EXAMPLE* that uses a dimension called *DAYS,* and contains the formula *DAY_EXAMPLE = DAY( )*. Because the argument is omitted, DAY returns the day for each date in the DAYS dimension.

Following is a more useful example of the DAY function: suppose you define a data cube called *DAILY_RECEIPTS* that uses a dimension called *DAYS.* You want to calculate the average receipts for each day of the month. In other words, you want to know the average receipts for the first day of each month, the average receipts for the second day of each month, and so on. To do this, create a dimension called *DAY_NUM* that contains members numbered 1 to 31. Then define a data cube called *AVG_RECEIPTS_BY_DAY* that uses the DAY_NUM dimension. Finally, enter the following formula for the AVG_RECEIPTS_BY_DAY data cube:

```
DAVG(DAYS, DAILY_RECEIPTS, DAY( ) = MEMBER(DAY_NUM))
```

For each DAY_NUM member in AVG_RECEIPTS_BY_DAY, the formula averages all DAILY_RECEIPTS where the day of the month equals the index of the DAY_NUM member. Thus, if the program is calculating the fifth DAY_NUM member for AVG_RECEIPTS_BY_DAY, it averages the receipts for the dates 2005/01/05, 2005/02/05, 2005/03/05, 2005/04/05, and so on, because these are the dates where the DAY( ) function returns *5*.

Related Links
MEMBER

DCOUNT

Syntax

DCOUNT*(Dimension, {Condition})*

Description

The DCOUNT function returns the number of members in *Dimension* for which *Condition* is true. If *Condition* is omitted, DCOUNT returns the number of members in *Dimension.*

Returns

The number of members in *Dimension* for which *Condition* is true. If *Condition* is omitted, DCOUNT returns the number of members in *Dimension.*

Example

Suppose an analytic model contains a data cube called *UNITS_SOLD* that uses a dimension called *PRODUCTS.* Use the following formula to find the number of products that sold more than 5000 units:

```
DCOUNT(PRODUCTS, UNITS_SOLD > 5000)
```

For an example of how to tabulate data for a series of ranges, see the entry for the DAVG function.

DDB

Syntax

DDB*(Cost, Salvage, Life, Period)*

Description

The DDB function returns the depreciation on an asset using the Double Declining Balance method. This is an accelerated depreciation method.

Parameters

Parameter	***Description***
Cost	The cost of the asset.
Salvage	The worth of the asset at the end of its useful life.
Life	The number of periods in the asset's useful life.
Period	The period for which you wish to determine the depreciation.

Returns

The depreciation on an asset using the Double Declining Balance method.

Example

Suppose you purchase a machine for USD 6000, and you plan to sell it for USD 500 after 5 years. You can calculate the depreciation for each year as follows:

- `DDB(6000, 500, 5, 1) = 2400`
- `DDB(6000, 500, 5, 2) = 1440`
- `DDB(6000, 500, 5, 3) = 864`
- `DDB(6000, 500, 5, 4) = 518`
- `DDB(6000, 500, 5, 5) = 278`

DEC

Syntax

DEC*(Number Original Value, Number Amount to Decrement)*

Description

The DEC function returns an decremented value based on an original value and the amount to increment.

Returns

An decremented value based on an original value and the amount to increment.

Example

```
DEC(&NumMonths, &EndMonth - &StartMonth - 1)
```

This formula subtracts the months between the start and end month to the variable *&NumMonths.* DEC function is useful in FOR or WHILE functions to decrement loop variables.

Related Links

FOR
WHILE

DLOOKUP

Syntax

DLOOKUP*(Dimension, Data, Condition, {Direction})*

Description

The DLOOKUP function returns *Data* for the first Member in *Dimension* where *Condition* is true. If *Direction* is omitted or zero, the function scans forward from the first member. If Direction is nonzero,

the function scans backward from the last member. If no member in *Dimension* fulfills the *Condition,* the function returns zero.

Returns

Data for the first member in *Dimension* where *Condition* is true. If *Direction* is omitted or zero, the function scans forward from the first member. If Direction is nonzero, the function scans backward from the last member. If no member in *Dimension* fulfills the *Condition,* the function returns zero.

Example

Suppose that a company awards its salespeople a 10 percent commission if their sales are at least USD 50000, an 8 percent commission if their sales are at least USD 30000, a 5 percent commission if their sales are at least USD 15000, and a 1 percent commission if their sales are less than USD 15000. One way to calculate the commission is to create a lookup table. Define a dimension called *RANGES* and attach it to data cubes called *SALES_MINIMUM* and *LOOKUP_RATE.* Each number in SALES_MINIMUM defines the minimum value for the sales range, while the next number defines the upper limit for the range. LOOKUP_RATE holds the commission rate for each range. Use the following formula to calculate the commission rate:

```
DLOOKUP(RANGES, LOOLUP_RATE, SALES >= SALES_MINIMUM, 1)
```

Because the last argument of DLOOKUP is 1, the function starts with the last member of RANGES and scans backwards until SALES is greater than or equal to SALES_MINIMUM. It is important to scan backwards to find the highest lookup rate for which the condition is true. Otherwise, the formula returns the lowest lookup rate no matter how high the value of SALES is.

DMAX

Syntax

DMAX*(Dimension, Data, {Condition})*

Description

The DMAX function returns the maximum of *Data* for the members in *Dimension* where *Condition* is True. If *Condition* is omitted, DMAX returns the maximum of *Data* for all members in *Dimension.* If *Data* is omitted, DMAX returns the maximum of the data cube being calculated, for all members up to the current member in *Dimension.*

Returns

The maximum of *Data* for the members in *Dimension* where *Condition* is True. If *Condition* is omitted, DMAX returns the maximum of *Data* for all members in *Dimension.* If *Data* is omitted, DMAX returns the maximum of the data cube being calculated, for all members up to the current member in *Dimension.*

Example

Suppose that an analytic model contains a data cube called *ADVERTISING_BY_PRODUCT* and a data cube called *UNITS_SOLD*. Both data cubes use a dimension called *PRODUCTS*. Use the following formula to calculate the maximum units sold for all products:

```
MAX(PRODUCTS, UNITS_SOLD)
```

The DMAX function does not include a condition, so the function finds the maximum of UNITS_SOLD for all members in the PRODUCTS dimension. Use the following formula to calculate the maximum units sold for all products with advertising under USD 10000:

```
DMAX(PRODUCTS, UNITS_SOLD, ADVERTISING_BY_PRODUCT < 10000)
```

In this case, the function finds the maximum units sold only for the products where ADVERTISING_BY_PRODUCT is less than 10000.

For an example of how to tabulate data for a series of ranges, see the entry for the DAVG function.

Related Links

DAVG

DMIN

Syntax

DMIN*(Dimension, Data, {Condition})*

Description

The DMIN function returns the minimum of *Data* for the members in *Dimension* where *Condition* is True. If *Condition* is omitted, DMIN returns the minimum of *Data* for all members in *Dimension*. If *Data* is omitted, DMIN returns the minimum of the data cube being calculated, for all members up to the current member in *Dimension*.

Returns

The minimum of *Data* for the members in *Dimension* where *Condition* is True. If *Condition* is omitted, DMIN returns the minimum of *Data* for all members in *Dimension*. If *Data* is omitted, DMIN returns the minimum of the data cube being calculated, for all members up to the current member in *Dimension*.

Example

Suppose that an analytic model contains a data cube called ADVERTISING_BY_PRODUCT and a data cube called UNITS_SOLD. Both data cubes use a dimension called PRODUCTS. Use the following formula to calculate the minimum units sold for all products:

```
DMIN(PRODUCTS, UNITS_SOLD)
```


The DMIN function does not include a condition, so the function finds the minimum of UNITS_SOLD for all members in the PRODUCTS dimension. Use the following formula to calculate the minimum units sold for all products with advertising of at least 10000 USD:

```
MIN(PRODUCTS, UNITS_SOLD, ADVERTISING_BY_PRODUCT >= 10000)
```

In this case, the function finds the minimum units sold only for the products where ADVERTISING_BY_PRODUCT is greater than or equal to 10000.

For an example of how to tabulate data for a series of ranges, see the entry for the DAVG function.

Related Links

DAVG

DSUM

Syntax

DSUM*(Dimension, {Data}, {Condition})*

Description

The DSUM function returns the sum of *Data* for the members in *Dimension* where *Condition* is True. If *Condition* is omitted, DSUM returns the sum of *Data* for all members in *Dimension.* If *Data* is omitted, DSUM returns the sum of the data cube being calculated for all members up to the current member in *Dimension.*

Example

Suppose that an analytic model contains a data cube called ADVERTISING_BY_PRODUCT and a data cube called UNITS_SOLD. Both data cubes use a dimension called PRODUCTS. Use the following formula to calculate the total units sold for all products:

```
DSUM(PRODUCTS, UNITS_SOLD)
```

The DSUM function does not include a condition, so the function computes the sum of UNITS_SOLD for all members in the PRODUCTS dimension. Use the following formula to calculate the sum of units sold for all products with advertising of at least 10000 USD:

```
DSUM(PRODUCTS, UNITS_SOLD, ADVERTISING_BY_PRODUCT >= 10000)
```

In this case, the function finds the sum of UNITS_SOLD only for the products where ADVERTISING_BY_PRODUCT is greater than or equal to 10000.

For an example of how to tabulate data for a series of ranges, see the entry for the DAVG function.

You can use the DSUM function without the *Data* argument to exercise complete control over the calculation of dimension totals for a particular data cube.

Related Links

DAVG

E

Syntax

E*( )*

Description

Use the E function to return the value of e, which is the base of natural logarithms.

Returns

The value of e.

Example

These examples employ the E built-in function:

- `E( )` returns *2.7182818285.*
- `E( ) ^ 5` returns *148.4131591026* (e raised to the 5^{th} power).

FIND

Syntax

FIND*(Text Original String, Text Sub String, Number Starting Position)*

Description

Use the FIND function to find a substring in the original string passed in starting from a specified start position in the original string.

Parameters

Parameter	***Description***
Text Original String	The text of the original string.
Text Sub String	The substring text to find.
Number Starting Position	The start position in the original string.

Returns

The position of the substring in the original string. The index is 1-based.

Example

The following formula finds the account name that begins with Expense:

```
IF(FIND(ACCOUNT_NAME, "Expense", 1) = 1, #TRUE, #FALSE)
```

FIRST

Syntax

FIRST*(Dimension)*

Description

Use the FIRST function to test for special cases that occur when the first member of a dimension is being calculated.

Parameters

Parameter	***Description***
Dimension	The dimension to test.

Returns

The FIRST function returns the first detail member regardless if the detail member is created out of a tree that is attached to the dimension. The FIRST function also returns True if the first member of *Dimension* is being calculated; otherwise, it returns False.

FOR

Syntax

FOR*(Index, Start, Finish, Loop Body)*

Description

The FOR function loops through a series of values.

Parameters

Parameter	***Description***
Index	The name of the variable that holds the index number.
Start	The index value at which to start the loop.

Parameter	*Description*
Finish	The index value at which to finish the loop.
Loop Body	The action to take at the current index.

Example

The following formula raises a base to an integral exponent without using the ^ operator:

```
IF(EXPONENT <> TRUNC(EXPONENT), RETURN(0.0));
 SET(&Result, 1);
FOR(&Index, 1, ABS(EXPONENT),
 SET(&Result, &Result * BASE));
IF(EXPONENT >= 0, &Result, 1 / &Result)
```

In this formula, the FOR function sets the specified variable to each value at the beginning of the loop, counting up if *Finish* is greater than *Start,* and counting down if *Start* is greater than *Finish.*

FORCHILDREN

Syntax

FORCHILDREN*(Dimension, Expression, {#DIRECT/#ALL/#DETAILS, {Parent Member}})*

Note: The third and fourth arguments are optional.

Description

The FORCHILDREN function loops through all child members of a dimension's parent member, unless you interrupt the FORCHILDREN function with a BREAK function.

Parameters

The FORCHILDREN function takes two required arguments and two optional arguments. The first and second arguments are required. The third and fourth arguments are optional.

Parameter	*Description*
Dimension	The dimension to use.
Expression	The expression to evaluate for each iteration.
#DIRECT, #ALL, #DETAILS	This argument is optional. Select from one of these predefined constants. **Note:** #DIRECT is the default constant.

Parameter	*Description*
Parent Member	This argument is optional. If you do not use this argument, the function applies to the member that is currently being evaluated.

Example

```
FORCHILDREN(Region,
    IF(Sales > & MaxSales,
   &MaxSales := Sales;
   &Region:= Member;
  ),
  #DIRECT,
  [Region:USA]
 );
  &Region;
```

Related Links

Pushed Down Data

FORMEMBERS

Syntax

FORMEMBERS*(Dimension, Direction, Expression)*

Description

The FORMEMBERS function loops through all of the members of a dimension (unless you interrupt it with the BREAK function).

Parameters

Parameter	*Description*
Dimension	The dimension to use.
Direction	The direction to loop through the members (#FORWARD or #REVERSE).
Expression	The expression to evaluate for each iteration.

Example

Consider the following formula that uses DLOOKUP:

```
DLOOKUP(RANGES, COMMISSION_RATE, SALES >= SALES_LEVEL, #REVERSE)
```

You could achieve the same thing with the FORMEMBERS function:

```
FORMEMBERS(RANGES, #REVERSE,
 IF(SALES >= SALES_LEVEL, RETURN(COMMISSION_RATE))
);
RETURN(0)
```

Of course, in this case it is simpler just to use the DLOOKUP function, but the FORMEMBERS function makes it possible to perform more sophisticated lookups and tabulations. For example, the following formula returns the product that has the highest sales:

```
FORMEMBERS(PRODUCTS, #FORWARD,
 IF(SALES > &MaxSales,
  SET(&MaxSales, SALES);
  SET(&Product, MEMBER(PRODUCTS))
 )
);
&Product
```

Following is how you would have to do it without procedural logic:

```
DLOOKUP(PRODUCTS, MEMBER(PRODUCTS), SALES = DMAX(PRODUCTS, SALES))
```

The above version is shorter, but it is much less efficient than the version that uses procedural logic, because it calculates the DMAX repeatedly for every product.

You could eliminate some of the redundancy by using an expression block and a variable:

```
SET(&MaxSales, DMAX(PRODUCTS, SALES));
DLOOKUP(PRODUCTS, MEMBER(PRODUCTS), SALES = &MaxSales)
```

The previous version is more effective than the version that does not use procedural logic, but it is not as effective as the version that uses procedural logic. This is because in the version that does not use procedural logic, the FORMEMBERS function only loops through the products once. In the previous version, it loops through them twice—once for the DMAX and once for the DLOOKUP—although the DLOOKUP stops when it finds the right product.

FV

Syntax

FV*(Rate, NPer, Pmt, PV, {Type})*

Description

The FV function returns the Future Value of an investment with a present value of *PV,* where *Pmt* is invested for *NPer* periods at *Rate* per period. If *Type* is omitted or zero, FV assumes the investment is an ordinary annuity. If *Type* is nonzero, FV assumes the investment is an annuity due.

Note: Enter negative amounts for money out of your pocket, or positive amounts for money coming to you.

Example

Suppose that you deposit USD 1000 at the end of each year in a savings account that earns 6 percent per year. To determine the value of the account after 8 years, use the following formula:

```
FV(0.06, 8, -1000, 0) = 9897
```

If you deposit the USD 1000 at the start of each year, use the following formula for the VALUE_OF_ACCT data cube. The 1 for the Type argument indicates an annuity due:

```
FV(0.06, 8, -1000, 0, 1) = 10491
```

If the account already has USD 3000 in it at the start of the 8 years, use the following formula:

```
FV(0.06, 8, -1000, -3000, 1) = 15273
```

GROUPAVG

Syntax

GROUPAVG*(Dimension to Group, Expression, Association 1, {Association 2 ...})*

Description

Use the GROUPAVG function to average information by group. *Expression* contains the data to sum. The *Assocation(s)* indicate for which group(s) to average.

Example

Suppose that you want to average employee salaries by department. Create an analytic model definition that contains the following data cubes:

1. EMPLOYEE_SALARY, which uses a dimension called *EMPLOYEES*.

   This data cube contains the salary for each employee.

2. AVERAGE_DEPARTMENT_SALARY, which uses a dimension called *DEPARTMENTS*.

   This data cube contains the average salaries for each department.

3. An association data cube called *EMPLOYEE_DEPT* by performing the following:

   - Create the EMPLOYEE_DEPT data cube.
   - Format the EMPLOYEE_DEPT data cube as a member of the DEPARTMENTS dimension.
   - Attach the EMPLOYEES dimension to the EMPLOYEE_DEPT data cube.

Calculate Department Salary with the following formula:

```
GROUPAVG(EMPLOYEES, EMPLOYEE_SALARY, EMPLOYEE_DEPT)
```

You can read this formula as follows: Average the employees' salaries by department.

To calculate group averages of all members that meet a condition, use an IF function as the expression, with *#N/A* as the third argument. For example, to calculate average officer salaries by department, you could use `IF(IS_OFFICER, EMPLOYEE_SALARY, #N/A)` instead of `EMPLOYEE_SALARY` in the formula above.

GROUPBY

Syntax

GROUPBY*(Association)*

Description

Use the GROUPBY function in a condition to group detail members by summary members (for example, employees by department). The argument must be an association data cube; otherwise, the function returns an error.

Example

Suppose that an analytic model contains an association data cube called *DEPARTMENTS,* which associates each employee with a particular department. The following formula for the EMPLOYEES_IN_DEPT cube uses DCOUNT and GROUPBY to calculate the number of employees in each department:

```
DCOUNT(EMPLOYEES, GROUPBY(DEPARTMENTS))
```

The following formula for the AVG_SALARY_BY_DEPT data cube uses DAVG and GROUPBY to calculate the average salary for each department:

```
DAVG(EMPLOYEES, EMPLOYEE_SALARY, GROUPBY(DEPARTMENTS))
```

You can combine the GROUPBY function with other conditions. For example, the following formula for the OFFICER_SALARIES_BY_DEPT cube uses the DSUM function to calculate the total *officer* salaries in each department. By combining `IS_OFFICER` with the GROUPBY function, the formula ensures that only officers are included in the sum:

```
DSUM(EMPLOYEES, EMPLOYEE_SALARY, GROUPBY(DEPARTMENTS) .AND. IS_OFFICER)
```

Note that `DSUM(EMPLOYEES, EMPLOYEE_SALARY, GROUPBY(DEPARTMENTS))` is equivalent to `GROUPSUM(EMPLOYEES, EMPLOYEE_SALARY, DEPARTMENTS)`. Using DSUM with GROUPBY is more flexible, because you can include other conditions, as shown in the formula above. On the other hand, the GROUPSUM function calculates significantly faster. For this reason, if you want to sum by group and you do not need to include other conditions, use the GROUPSUM function.

Related Links

DCOUNT
DAVG
DSUM
GROUPSUM

GROUPMAX

Syntax

GROUPMAX*(Dimension to Group, Expression, Association 1, {Association 2 ...})*

Description

Use the GROUPMAX function to maximize information by group. *Expression* contains the data to maximize. The *Association(s)* indicate for which group(s) to maximize.

Example

Suppose that you want to maximize sales information by product. Create an analytic model definition that contains the following dimensions:

1. TRANSACTIONS, which contains a series of sales transactions.
2. PRODUCTS, which contains a dimension of products.

Define the following data cubes:

1. SALE_AMOUNT, which uses the TRANSACTIONS dimension. This data cube contains the amount of each sale.
2. An association data cube called *PRODUCT_SOLD*, which associates TRANSACTIONS with PRODUCTS.

   See Creating Association Data Cubes.
3. MAXIMUM_SALES_BY_PRODUCT, which uses the PRODUCTS dimension.

   Calculate this data cube with the following formula:

```
GROUPMAX(TRANSACTIONS, SALE_AMOUNT, PRODUCT_SOLD)
```

You can read this formula as follows: Find the maximum transactions' sale amounts by product.

To calculate group maximums of all members that meet a condition, use an IF function as the expression, with *#N/A* as the third argument. For example, use `IF(VALID, SALE_AMOUNT, #N/A)` instead of `SALE_AMOUNT` in the formula above.

GROUPMIN

Syntax

GROUPMIN*(Dimension to Group, Expression, Association 1, {Association 2 ...})*

Description

Use the GROUPMIN function to minimize information by group. *Expression* contains the data to minimize. The *Association(s)* indicate for which group(s) to minimize.

Example

Suppose that you want to minimize sales information by product. Create an analytic model definition that contains the following dimensions:

1. TRANSACTIONS, which contains a series of sales transactions.
2. PRODUCTS, which contains a series of products.

Define the following data cubes:

1. SALE_AMOUNT, which uses the TRANSACTIONS dimension.

   This data cube contains the amount of each sale.

2. An association data cube called *PRODUCT_SOLD,* which associates TRANSACTIONS with PRODUCTS.

   See Creating Association Data Cubes.

3. MINIMUM_SALES_BY_PRODUCT, which uses the PRODUCTS dimension. Calculate this data cube with the following formula:

   ```
   GROUPMAX(TRANSACTIONS, SALE_AMOUNT, PRODUCT_SOLD)
   ```

You can read this formula as follows: Find the maximum transactions' sale amounts by product.

To calculate group maximums of all members that meet a condition, use an IF function as the expression, with *#N/A* as the third argument. For example, use `IF(VALID, SALE_AMOUNT, #N/A)` instead of `SALE_AMOUNT` in the formula above.

GROUPSUM

Syntax

GROUPSUM*(Dimension to Group, Expression, Association 1, {Association 2 ...})*

Description

Use the GROUPSUM function to sum information by group. *Expression* contains the data to sum. The *Association(s)* indicate what group(s) to sum by.

Example

The following examples provide uses of the GROUPSUM function.

Example 1

For example, suppose you want to sum employee salaries by department. Create an analytic model definition that contains the following data cubes:

1. EMPLOYEE_SALARY, which uses a dimension called *EMPLOYEES.*

   This data cube contains the salary for each employee.

2. DEPARTMENT_SALARY, which uses a dimension called *DEPARTMENTS.*

   This data cube contains the total salaries for each department.

3. An association data cube called *EMPLOYEE_DEPT,* which associates each employee with a particular department.

Calculate DEPARTMENT_SALARY with the following formula:

```
GROUPSUM(EMPLOYEES, EMPLOYEE_SALARY, EMPLOYEE_DEPT)
```

You can read this formula as follows: Sum the employees' salaries by department.

Example 2

The GROUPSUM function can also perform more complex groupings. For example, suppose you want to sum employee salaries by branch and department. To do this, perform the following additional steps:

1. Create a dimension called *BRANCHES,* which contains a dimension of the branches.

2. Create an association data cube called *EMPLOYEE_BRANCH,* which associates the EMPLOYEES dimension with the BRANCHES dimension.

3. Define a data cube called *SALARY_BY_BRANCH_AND_DEPT,* which uses both the BRANCHES and DEPARTMENTS dimensions. Calculate this data cube with the following formula:

   ```
   GROUPSUM(EMPLOYEES, EMPLOYEE_SALARY, EMPLOYEE_BRANCH, EMPLOYEE_DEPT)
   ```

You can read this formula as follows: Sum the employees' salaries by branch and department.

As this example demonstrates, you can summarize detail information for a combination of dimensions by using an association for each dimension.

Example 3

In many cases, it is useful to summarize information by date. In these cases, use a data cube with a *Date* format instead of an association.

For example, suppose you want to summarize sales information by product and month. Create an analytic model definition that contains the following dimensions:

1. TRANSACTIONS, which contains a series of sales transactions.

2. PRODUCTS, which contains a dimension of products.

3. MONTHS, which contains a series of months.

Define the following data cubes:

1. SALE_AMOUNT, which uses the TRANSACTIONS dimension.

   This data cube contains the amount of each sale.

2. An association data cube called *PRODUCT_SOLD,* which associates TRANSACTIONS with PRODUCTS.

3. SALE_DATE, which uses the TRANSACTIONS dimension and the YYYY/MM/DD format. This data cube contains the date for each transaction.
4. MONTHLY_SALES, which uses the PRODUCTS and MONTHS dimension. Calculate this data cube with the following formula:

```
GROUPSUM(TRANSACTIONS, SALE_AMOUNT, PRODUCT_SOLD, SALE_DATE)
```

You can read this formula as follows: Sum the transactions' sale amounts by product and sale date. Because SALE_DATE is Date formatted (YYYY/MM/DD), the GROUPSUM function knows to sum by date.

To calculate group sums of all members which meet a condition, use an IF function as the expression, with *#N/A* as the third argument. For example, use `IF(Valid, SALE_AMOUNT, #N/A)` instead of `SALE_AMOUNT` in the MONTHLY_SALES formula above.

GROW

Syntax

GROW*(Dimension, Start Value, Growth Rate)*

Description

The GROW function returns a number representing the specified *Growth Rate* per Member from *Start Value.* This is a straight line growth function.

Example

Suppose that an analytic model contains single value data cubes called *SALES_START* and *ANNUAL_GROWTH.* You can project the monthly sales with the following formula:

```
GROW(MONTHS, SALES_START, ANNUAL_GROWTH / 12)
```

Note that you must divide ANNUAL_GROWTH by 12, because the GROW function expects a growth rate *per member,* and the members in this case are months.

Note: For the GROW function to return meaningful results, the *Start Value* and *Growth Rate* arguments should not use the dimension indicated by the dimension argument. For example, if you are calculating monthly values, the *Start Value* and *Growth Rate* arguments should not use the MONTHS dimension.

IF

Syntax

IF*(Condition, Result if True, Result if False)*

Description

The IF function returns *Result if True* if Condition is true; otherwise, the function returns *Result if False.*

Returns

Result if True if Condition is true; otherwise, the function returns *Result if False.*

Example

For example, suppose a company awards its salespeople a 10 percent commission on sales of at least USD 20000, and a 5 percent commission on sales under USD 20000. You create a COMMISSION cube and can compute the commission for each person as follows:

```
IF(SALES >= 20000, 0.1 * SALES, 0.05 * SALES)
```

The IF function in this formula tests whether SALES is greater than or equal to 20000. If so, the function returns 10 percent of SALES; otherwise, the function returns 5 percent of SALES.

INC

Syntax

INC*(Original Value, Amount to Increment)*

Description

The INC function returns an incremental value based on an original value and the amount to increment.

Returns

An incremental value based on an original value and the amount to increment.

Example

```
INC(&NumMonths, &EndMonth - &StartMonth - 1)
```

This formula adds the months between the start and end month to the variable &NumMonths.

INCDATE

Syntax

INCDATE*(Date, Months, Years)*

Description

The INCDATE returns the value of *Date* incremented by *Months* and *Years.*

Returns

The value of *Date* incremented by *Months* and *Years.*

Example

If *Date* contains the date 2001/04/18, `INCDATE(Date, 3, 2)` returns the date 2003/07/18.

If *Date* falls on the last day of a month, `INCDATE` returns a date that falls on the last day of a month, even if it has to change the day. For example, if *Date* contains the date 2003/04/30, then `INCDATE(Date, 3, 2)` returns the date 2005/07/31 rather than 2005/07/30. Because *Date* contains the last day of April, `INCDATE` returns the last day of July.

Suppose that an analytic model contains a data cube called *HIRE_DATE* that uses a dimension called *EMPLOYEES*. Company policy starts benefits for an employee three months after the hire date. The following formula calculates the benefits date for each employee as follows:

```
INCDATE(HIRE_DATE, 3, 0)
```

Related Links

IF

INPUT

Syntax

INPUT*()*

Description

The INPUT function returns the value that an end user has entered into a cell, and supports both calculated cells and input cells in a single data cube.

Returns

The value that an end user has entered into a cell.

Example

You can use the INPUT function in both an IF function and a CASE function. For example:

```
IF([SCENARIOS:Actual], INPUT( ), FORECAST_REVENUE_CALCULATION)
```

This formula returns `88` if the Scenario value is Actual and the end user enters *88*. This formula causes all cells for the Actual dimension member to become input cells, leaving the remaining cells to be calculated.

When a formula uses the INPUT function, the analytic calculation engine evaluates the formula for a particular cell to determine whether it should be an input cell. As long as the input condition in the formula refers to input cubes and member references, no recalculation is necessary to ensure that the correct cells are treated as input cells.

The INPUT function works a lot like the RETURN function; it causes the analytic calculation engine to stop evaluating the formula and to immediately return a value, which in this case is the current value of the cell. The INPUT function works like `RETURN(SELF( ))`, and additionally makes the cell editable.

Related Links

Input Data Cubes
ISINPUT

INSUBTREE

Syntax

INSUBTREE*(Dimension, Parent Member, {Child Member})*

Description

The INSUBTREE function returns a boolean value identifying whether a child member is in a sub-tree that contains a parent member as its root.

Parameters

The INSUBTREE function takes two required arguments and one optional argument. The first and second arguments are required. The third argument is optional.

Parameter	***Description***
Dimension	The dimension to use.
Parent Member	The parent member to use.
Child Member	This optional argument tests whether the child member lies within a sub-tree that contains the parent member (as supplied in the second argument) as its root.

Example

```
INSUBTREE(Region, [Region:USA], [Region:Oakland]);
```

ISINPUT

Syntax

ISINPUT*(Cube)*

Description

The ISINPUT function returns True if the user has entered the current value of the cube.

Example

`ISINPUT`(Cube with no formula) returns *True.*

`ISINPUT`(Cube with formula) returns *False.*

The ISINPUT function provides an easy way to filter tables so that they show input cells. The ISINPUT function takes a single argument, which must be a cube.

To work well with filter functions, the function maps totals to the first member of the dimension if a first member exists. Because the row filters are not aware of members in the columns—and vice versa—the analytic calculation engine usually evaluates totals in formulas. The analytic calculation engine already *bends* the total mapping rules for filters for this reason; the behavior of ISINPUT is just an extension of this behavior.

INTERCEPT

Syntax

INTERCEPT(*Dimension, Y, X, {Condition})*

Description

The INTERCEPT function returns the Y-intercept of the line that has the closest fit to the points represented by Y and X. (The Y-intercept is the point at which the line crosses the Y axis.) If *Condition* is omitted, the function fits the line to all of the members in *Dimension*. If *Condition* is included, the function fits the line only to those members that meet the condition.

Use the INTERCEPT function together with the SLOPE function to find the line with the closest fit to a set of points. You can use these functions to analyze a historical trend, and then use the trend to make forecasts. For interesting examples of how to use these functions, see the entry for the SLOPE function.

Returns

The Y-intercept of the line that has the closest fit to the points represented by Y and X. (The Y-intercept is the point at which the line crosses the Y axis.) If *Condition* is omitted, the function fits the line to all of the members in *Dimension.* If *Condition* is included, the function fits the line only to those members that meet the condition.

Related Links

SLOPE

IRR

Syntax

IRR*(Dimension, Cash Flow, {Guess}, {Condition})*

Description

The IRR function returns the Internal Rate of Return for *Cash Flow. Guess* can be omitted (or zero) in most cases. If IRR is unable to find the Internal Rate of Return, it returns an error value. In such cases, you can use a nonzero *Guess* to nudge IRR toward the correct answer.

Note: Some cash flows have no valid Internal Rate of Return, in which case IRR returns an error value for any *Guess*.

If *Condition* is omitted, the function uses all values of *Cash Flow*. If *Condition* is included, the function uses only those values of *Cash Flow* for which *Condition* is True.

The initial values in the cash flow should be negative to represent a cash outflow. The remaining values may be all positive (representing cash inflows) or a combination of positive and negative.

Returns

The Internal Rate of Return for *Cash Flow*. *Guess* can be omitted (or zero) in most cases. If IRR is unable to find the Internal Rate of Return, it returns an error value. In such cases, you can use a nonzero *Guess* to nudge IRR toward the correct answer.

Example

You can calculate the internal rate of return for a data cube called IRR_OF_CASH_FLOW with this formula:

```
IRR(MONTHS, CASH_FLOW)
```

You can calculate the internal rate of return for the first 12 months for a data cube called RR_FOR_1ST_12_MONTHS with this formula:

```
RR(MONTHS, CASH_FLOW, 0, MEMBER(MONTHS) <= 12)
```

The Condition ensures that the IRR function uses only those values for which the month index is 12 or less.

LN

Syntax

LN*(Data)*

Description

The LN function returns the natural logarithm of *Data*. The value of *Data* must be greater than zero; otherwise, LN returns an error value.

Example

These examples employ the LN built-in function:

- `LN(7)` returns *1.9459101491*.
- `LN(E( ) ^ 5)` returns *5*.
- `LN(25) / LN(5)` returns *2*.

- `LN(-7)` returns an error.

LEFT

Syntax

LEFT*(Text, Count)*

Description

The LEFT function returns the first *Count* characters of *Text.*

Returns

The first *Count* characters of *Text.*

Example

`LEFT("StringFun", 6)` returns *String.*

LEN

Syntax

LEN*(Text)*

Description

The LEN FUNCTION returns the number of characters in text string.

Returns

The number of characters in text string.

Example

`LEN("StringFun")` returns *9.*

LOWER

Syntax

LOWER*(Text)*

Description

The LOWER FUNCTION returns text converted to lower case.

Returns

Text converted to lower case.

Example

`LOWER("StringFun")` returns *stringfun.*

MATCH

Syntax

MATCH*(Text Expression* or *Text Cube, Pattern, {Case Sensitive}, {Match Type}})*

Description

The MATCH function returns True if Text Expression or *Text Cube* matches *Pattern.*

Pattern can be any of the following:

- A text value in quotes (for example, *"hello").*
- A data cube with a Text format.
- The VALUE function, as in *VALUE("Name").*

If *Case Sensitive* is omitted or zero, the function ignores case. If *Case Sensitive* is nonzero, the function performs a case sensitive match.

Match Type can be one of these values:

- 0: Text contains *Pattern.*
- 1: Text matches *Pattern* exactly.
- 2: Text begins with *Pattern.*
- 3: Text ends with *Pattern.*

If the *Match Type* argument is omitted, it is assumed to be zero (text contains *Pattern).*

Returns

True if Text Expression or *Text Cube* matches *Pattern.*

Example

Suppose that *Title* = "A Quick Brown Fox" and *Pattern* = "brown." These results occur:

- `MATCH(Title, "A quick brown fox")` returns True. *(Case Sensitive* argument is omitted, so the case does not have to match.)

- `MATCH(Title, "a quick brown fox", 1)` returns False. *(Case Sensitive* argument is 1, and the case does not match.)
- `MATCH(Title, "A Quick Brown", 1, 1)` returns False. *(Match Type* argument is 1, and the pattern does not match exactly.)
- `MATCH(Title, "brown")` returns True. *(Title* contains *Brown.)*
- `MATCH(Title, Pattern)` returns True. *(Pattern* equals *brown,* and *Title* contains the word *Brown.)*
- `MATCH(Title, "a quick", 0, 2)` returns True. *(Title* begins with *A Quick.)*
- `MATCH(Title, "fox", 0, 2)` returns False. *(Title* does not begin with *fox.)*
- `MATCH(Title, "fox", 0, 3)` returns True. *(Title* ends with *Fox.)*
- `MATCH(LEFT(Title, 6), "Brown Fox", 1, 2)` returns False *(Title* does not begin with *Brown Fox.)*
- `MATCH(MID(Title, 0, 7), "A Quick", 1)` returns True *(Title* contains *A Quick.)*
- `MATCH(RIGHT(Title, 9), "Brown Fox", 1, 3)` returns True *(Title* ends with *Brown Fox.)*

MAX

Syntax

MAX*(arg1, arg2, . . . arg16)*

Description

The MAX Function returns the maximum of a series of values. The MAX Function accepts up to 16 arguments.

Example

Given A = 4, B = 3, C = 2, D = 1

`MAX(A, B, C, D)` returns **A.**

You can sometimes simplify formulas by using the MAX function instead of the IF function.
For example, suppose an analytic model contains data cubes called CASH_BALANCE and CASH_MINIMUM. You might be tempted to calculate the CASH_NEEDED cube by using the following formula:

```
IF(CASH_BALANCE < CASH_MINIMUM,CASH_MINIMUM - CASH_BALANCE, 0)
```

In other words, if CASH_BALANCE is less than CASH_MINIMUM, return the amount required to attain the minimum cash level; otherwise, return zero. Although the IF function does the job, it is simpler to use the MAX function:

```
MAX(CASH_MINIMUM - CASH_BALANCE, 0)
```


If CASH_BALANCE is greater than CASH_MINIMUM, the first argument is negative, so the formula returns zero for CASH_NEEDED. If CASH_BALANCE is less than CASH_MINIMUM, the first argument is positive, so the formula returns the amount required to attain the minimum cash level.

Related Links

IF
MIN

MBR2TEXT

Syntax

MBR2TEXT*(Dimension, {Member})*

Description

The MBR2TEXT function converts a member to text by returning its name. The *Member* argument can be an association data cube, a member reference, a member index, or a function or expression that returns a member. If *Member* is omitted, the function returns the name of the current member in *Dimension.* In other words, it is equivalent to:

```
MBR2TEXT(DIMENSION, MEMBER(DIMENSION))
```

Example

`MBR2TEXT(MONTH)` returns *January.*

MEDIAN

Syntax

MEDIAN*(Dimension, Values, {Condition})*

Description

The MEDIAN function returns the median of Values. If *Condition* is omitted, the function uses all *Values.* If *Condition* is included, the function uses only those *Values* for which *Condition* is true.

Returns

The median of Values. If *Condition* is omitted, the function uses all *Values.* If *Condition* is included, the function uses only those *Values* for which *Condition* is true.

Example

Suppose that a cube collection contains a data cube called SALES that uses dimensions called PEOPLE and MONTHS. It also contains a data cube called MEDIAN_OF_SALES that contains the following formula for calculating the median over time for each person:

```
MEDIAN(MONTHS, SALES)
```

The cube collection also contains a data cube called MEDIAN_OF_SALES_IN_FIRST_6_MONTHS that contains this formula:

```
MEDIAN(MONTHS, SALES, MEMBER(MONTHS) <= 6)
```

MEMBER

Syntax

MEMBER*(Dimension)*

Description

The MEMBER function returns the Member being calculated.

Returns

The member being calculated.

Example

The following examples employ the MEMBER function:

Example 1

You can perform different calculations for different ranges of members by comparing the MEMBER function to a number. For example, to perform a special calculation for the first six months, use the MEMBER function with the IF function:

```
IF(MEMBER(MONTHS) <= 6, EXPR_FOR_1ST_6_MONTHS, EXPR_FOR_OTHER_MONTHS)
```

Example 2

You can perform special calculations for a particular member in a dimension by comparing the MEMBER function to a member reference. For example, suppose that your company allocates the Administration department's expense equally to all of the other departments. Your analytic model contains a DEPARTMENTS dimension, of which Admin is a member. Your analytic model also contains a data cube, TOTAL_EXPENSE, that contains the total expense for each department. The following formula describes how you would calculate the administration allocation for each department in a data cube called ADMIN_ALLOCATION:

```
IF(MEMBER(DEPARTMENTS) = [DEPARTMENTS:Admin], -TOTAL_EXPENSE,
TOTAL_EXPENSE [DEPARTMENTS:Admin] / (NUMMEMBERS(DEPARTMENTS) - 1))
```


The NUMMEMBERS function returns the number of members in a dimension. The key to this formula is that the allocation for each department is the same except for Admin. The IF function checks which department is being calculated. If the department is Admin, the result is minus Total Expense so that it backs out the expense for the Admin department. If the department is not Admin, the result is Total Expense for Admin—notice the data cube slice—divided by the number of departments other than Admin.

Related Links

IF
NUMMEMBERS

MID

Syntax

MID*(Text, Start, {Count})*

Description

The MID function returns *Count* characters from text, beginning with *Start.* If *Count* is omitted, returns all characters to the end of text.

Example

`MID("StringFun", 6, 3)` returns *Fun.*

MIN

Syntax

MIN*(X, Y)*

MIN*(arg1, arg2, . . . arg16)*

Description

The MIN function returns the minimum of a series of values. It accepts up to 16 arguments.

Example

Given A = 4, B = 3, C = 2, D = 1.

`MIN(A, B, C, D)` returns *D.*

You can sometimes simplify formulas by using the MIN function instead of the IF function. For example, suppose that an analytic model contains data cubes called CASH_NEEDED, CREDIT_BALANCE, and MAX_CREDIT. You might be tempted to calculate the CREDIT_DRAW by using the following formula:

```
IF(CASH_NEEDED <= MAX_CREDIT - CREDIT_BALANCE,
CASH_NEEDED, MAX_CREDIT - CREDIT_BALANCE)
```


In other words, if CASH_NEEDED is less than or equal to the available credit, draw the full CASH_NEEDED; otherwise, draw the available credit. Although the IF function does the job, the MIN function is simpler:

```
MIN(CASH_NEEDED, MAX_CREDIT - CREDIT_BALANCE)
```

If CASH_NEEDED is less than the available credit, the formula returns CASH_NEEDED; otherwise, the formula returns the available credit.

Related Links

IF
MAX

MOD

Syntax

MOD*(X, Y)*

Description

The MOD function returns the remainder of X divided by Y. If Y is zero, MOD returns an error value.

Returns

The remainder of X divided by Y. If Y is zero, MOD returns an error value.

Example

The following examples employ the MOD built-in function:

- `MOD(10, 4)` returns *2*.
- `MOD(15, 10)` returns *5*.
- `MOD(15, 5)` returns *0*.
- `MOD(15, 0)` returns an error value.

MONTH

Syntax

MONTH*({Date})*

Description

The MONTH function returns the month of *Date*. If *Date* is omitted, the function returns the month of the calculation date.

Returns

The month of *Date.* If *Date* is omitted, the function returns the month of the calculation date.

Example

If A = 2004/03/15 and B = 2005/06/22, `MONTH(A)` returns *3* and `MONTH(B)` returns *6.*

Now suppose that an analytic model contains a data cube called MONTH_EXAMPLE that uses a dimension called MONTHS and has the formula `MONTH_EXAMPLE = MONTH( )`. Because the argument is omitted, `MONTH` returns the month for each date in the MONTHS dimension.

Following is a useful example of the MONTH function. Suppose that you define a data cube called MONTHLY_SALES that uses a dimension called MONTHS. You want to calculate the average sales for each month of the year. In other words, you want to know the average sales for the first month of each year, the average sales for the second month of each year, and so on. To do this, create a dimension called MONTH_NUM that contains members numbered 1 to 12. Then define a data cube called AVG_SALES_BY_MONTH that uses the MONTH_NUM dimension. Finally, enter the following formula for the AVG_SALES_BY_MONTH cube:

```
DAVG(MONTHS, MONTHLY_SALES, MONTH( ) = MEMBER(MONTH_NUM))
```

See the entries for DAVG and MEMBER if you are unfamiliar with these functions. For each MONTH_NUM member in AVG_SALES_BY_MONTH, the formula averages all Monthly Sales for which the month of the year equals the index of the MONTH_NUM member. Thus, if the analytic calculation engine calculates the fifth MONTH_NUM member for AVG_SALES_BY_MONTH, it averages the sales for the dates 2004/05/03, 2004/05/04, and 2004/05/05, because these are the dates for which the MONTH( ) function returns *5*.

Related Links

DAVG
MEMBER

NEXT

Syntax

NEXT*(Dimension, Data, {Count})*

Description

The NEXT function returns the value of *Data* from *Count* members forward in *Dimension*. If *Count* is omitted, it is assumed to be 1.

Note: The NEXT function operates on detail member names that are persisted in the main record. This function does not use trees to determine the order of members.

Returns

The value of *Data* from *Count* members forward in *Dimension*. If *Count* is omitted, it is assumed to be 1.

Example

To refer to the next month's sales in a rule, use `NEXT(MONTHS, SALES)`.

The NEXT function can be used together with the CUMAVG function to calculate a *centered moving average,* such as the average sales for the six months before and after a given month. The centered moving average gives a sense of the normal monthly value for the year surrounding a particular month. You can then compare the actual monthly value to the normal monthly value to see how seasonality affected the sales. Thus, if the actual monthly value for August is higher than the normal monthly value for the year surrounding August, this may indicate that sales tend to be higher than average in August.

Suppose that the actual monthly sales are stored in a data cube called ACTUAL_SALES. Calculate the CENTERED_AVG_SALES cube as follows:

```
NEXT(MONTHS, CUMAVG(MONTHS, ACTUAL_SALES, 13), 6)
```

This formula looks six months ahead `(NEXT(MONTHS, ..., 6))`, and then calculates the cumulative average of the 13 months of sales preceding that time `(CUMAVG(MONTHS, ACTUAL_SALES, 13))`. For example, when the analytic calculation engine calculates CENTERED_AVG_SALES for 2005/03, it looks ahead six months to 2005/09, and then calculates the average sales for the 13 months preceding 2005/09. Thus, the analytic calculation engine calculates the average sales for 2004/09 to 2005/09, which is the year surrounding 2005/03.

Actually, this formula is not quite complete. You cannot calculate accurate results for the first six months or the last six months of the analytic model because the analytic calculation engine is unable to look six months back and six months ahead during those months. Therefore, the formula should return zero for those months:

```
IF(MEMBER(MONTHS) > 6 .AND. MEMBER(MONTHS) <= NUMMEMBERS(MONTHS) - 6,
NEXT(MONTHS, CUMAVG(MONTHS, ACTUAL_SALES, 13), 6), 0)
```

The condition of the IF statement ensures that the month being calculated is after the first six months and before the last six months of the analytic model. If the condition is true, the IF function returns the centered moving average calculated by the second argument; otherwise, the IF function returns zero.

Related Links

DAVG
MEMBER
NUMMEMBERS

NPER

Syntax

NPER*(Rate, Pmt, PV, FV, {Type})*

Description

The NPER function returns the number of payment periods required to accumulate a future value of *FV* when the present value is *PV,* the payment is *Pmt,* and the rate is *Rate.* If *Type* is omitted or zero, NPER assumes that the investment is an ordinary annuity. If *Type* is nonzero, NPER assumes that the investment is an annuity due.

Note: Enter negative amounts for money out of your pocket, or positive amounts for money coming to you.

Example

Suppose that you deposit 1000 USD at the end of each year in a savings account that earns 6 percent per year. To determine how many years it takes before the account is worth 20000 USD , use the following formula for the YEARS_REQUIRED cube:

```
NPER(0.06, -1000, 0, 20000) = 13.53
```

Note: The decimal part of the answer is not particularly meaningful; you cannot be sure of getting the 20000 USD until the end of the 14th year.

If you deposit the 1000 USD at the start of each year, use the following formula. The 1 for the *Type* argument indicates an annuity due:

```
NPER(0.06, -1000, 0, 20000, 1) = 12.99
```

If the account already has 5000 USD in it at the start, use the following formula:

```
NPER(0.06, -1000, -5000, 20000, 1) = 8.72
```

NPV

Syntax

NPV*(Dimension, Rate, Cash Flow, {Type},{Condition})*

Description

The NPV function returns the Net Present Value for Cash Flow, where *Rate* is the rate per period. If *Type* is zero or omitted, NPV treats the investment as an ordinary annuity; otherwise, NPV treats the investment as an annuity due. If *Condition* is omitted, the function uses all values of Cash Flow. If *Condition* is included, the function uses only those values of Cash Flow for which *Condition* is true.

The initial values in the cash flow should be negative to represent a cash outflow. The remaining values may be all positive (representing cash inflows), or a combination of positive and negative values.

Example

You can create a data cube called NET_PRESENT_VALUE and calculate the net present value for a data cube called CASH_FLOW with the following formula:

```
NPV(MONTHS, ANNUAL_RATE / 12, CASH_FLOW)
```

You can calculate the net present value for the first 12 months with the following formula:

```
NPV(MONTHS, ANNUAL_RATE / 12, CASH_FLOW, 0, MEMBER(MONTHS) <= 12)
```

The *Condition* ensures that the NPV function uses only those values for which the month index is 12 or less.

Related Links

MEMBER

NUM2TEXT

Syntax

NUM2TEXT*(Number, {Decimal Places})*

Description

The NUM2TEXT function converts *Number* to Text. *Decimal Places* specifies the number of decimal places that are used to convert the number to text.

Example

`NUM2TEXT(SALESPRICE, 3)` for SALESPRICE's value of 10.23457 as the string *10.234*.

NUMMEMBERS

Syntax

NUMMEMBERS*(Dimension)*

Description

The NUMMEMBERS function returns the number of members in *Dimension*.

Returns

Returns the number of members in *Dimension*.

Example

If a dimension called PRODUCTS contains eight members; `NUMMEMBERS(Products)` returns *8*.

OPRID

Syntax

OPRID*()*

Description

The OPRID function returns the user ID of the user who currently has the analytic instance checked out.

Use the OPRID function within a filter user function whose purpose is to limit user ID access to only certain rows of data.

Example

```
IF(AT(USERID, TXT2MBR(USERID, OPERID()),
DEPT_CUBE) =  MEMBER(DEPT_DIM),RETURN(1), RETURN(0))
```

This filter user function restricts user access to bonus amount data. Each user ID has access to only the bonus amount that pertains to them. The filter user function contains these data cubes and dimensions:

- USERID dimension, which is mapped to the USERID field.

  The USERID field contains the user IDs of the users that currently have the analytic instance loaded.

- DEPT_CUBE data cube, which is mapped to the DEPT_CUBE field.

  This data cube is formatted as a member of the DEPT_DIM dimension.

- DEPT_DIM dimension, which is mapped to the DEPT_DIM field.

  Note: The filter user function is applied to this dimension.

- BONUS_AMT data cube, which is mapped to the BONUS_AMT field.

The following table lists the values of the fields that are mapped to the USERID dimension and DEPT_CUBE data cube.

USERID	***DEPT_CUBE***
Juan	Doc
Albert	Dev
Nigel	PM

The following table lists the values of the fields that are mapped to the DEPT_DIM dimension and BONUS_AMT data cube.

DEPT_DIM	***BONUS_AMT***
Dev	5000
Doc	4000
PM	7000

The analytic calculation engine performs these steps to calculate the filter user function:

1. The OPRID function returns the user ID of the current user in text format.

2. The TXT2MBR function compares the user ID with the member in the USERID dimension to determine if they match.

   If the user ID matches the member in the USERID dimension, the AT function searches for the coordinates of the user ID member that is returned by TXT2MBR and returns the corresponding value of DEPT_CUBE.

   On the right-hand side of the equation, the MEMBER function returns the corresponding member of DEPT_DIM.

3. The analytic calculation performs one of these actions:

   - If the value returned from DEPT_CUBE matches the member returned from DEPT_DIM, the user ID can see the bonus amount.

     For example, the Dev value returned from DEPT_CUBE matches the Dev member returned from DEPT_DIM. For this reason, Albert can see his bonus amount of 5000.

   - If the value returned from DEPT_CUBE does not match the member returned from DEPT_DIM, the user ID cannot see the bonus amount.

Related Links

AT
MEMBER
TEXT2MBR
Filter User Functions

PARENT

Syntax

PARENT*(Dimension, {Child Member})*

Description

The PARENT function returns the member reference to the parent of the specified member.

Parameters

The PARENT function takes one required argument and one optional argument. The first argument is required. The second argument is optional.

Parameter	***Description***
Dimension	The dimension to use.

Parameter	*Description*
Child Member	If this optional argument is not supplied, use the current calculated member for this dimension. **Note:** If the child member is the root, this function returns *1*.

Returns

The member reference to the parent of the specified member.

Example

`PARENT (Region, [Region:West])` returns a reference to the parent of [Region:West], which is *[Region:USA]*.

Related Links

Pushed Down Data

PCT

Syntax

PCT*(Dimension, Data, {Count})*

Description

The PCT function returns the percentage change between the value of *Data* for the *Member* being calculated and the value of *Data* for *Count* members back. If *Count* is omitted, it is assumed to be 1.

Returns

The percentage change between the value of *Data* for the *Member* being calculated and the value of *Data* for *Count* members back. If *Count* is omitted, it is assumed to be 1.

Example

Suppose that you wish to calculate the monthly and yearly percentage change in a data cube called SALES. If SALES uses a dimension called MONTHS, use the following formula:

```
PCT(MONTHS, SALES)
```

Because the *Count* argument is omitted, the program assumes it to be 1. Thus, the program calculates the percentage change in sales from the previous month to the month being calculated. Calculate the YEARLY_PERCENT_CHANGE cube by using 12 for the third argument:

```
PCT(MONTHS, SALES, 12)
```

This formula calculates the percentage change in SALES from 12 months ago to the month being calculated.

PERCENTILE

Syntax

PERCENTILE*(Dimension, Values, Percentile, {Type}, {Condition})*

Description

The PERCENTILE function returns a percentile of *Values.* The *Percentile* argument sets which percentile is calculated. If *Type* is zero or omitted, PERCENTILE calculates a population percentile; otherwise, PERCENTILE calculates a sample percentile. If *Condition* is omitted, the function uses all *Values.* If *Condition* is included, the function uses only those *Values* for which *Condition* is true.

Example

Suppose that an analytic model contains a data cube called SCORES that uses dimensions called STUDENTS and TESTS.

The following formula calculates the 90th percentile of the scores for each test:

```
PERCENTILE(STUDENTS, SCORES, 90%)
```

The following formula calculates the 50th percentile of the first 10 students:

```
PERCENTILE(STUDENTS, SCORES, 50%, MEMBER(STUDENTS) <= 10)
```

This formula calculates the 50th percentile (also knows as median) of the first 10 students for each test.

PI

Syntax

PI( )

Description

The PI function returns the value of PI (3.1415926536), the ratio of a circle's circumference to its diameter.

The following formula calculates the area of a circle:

```
PI( ) * RADIUS ^ 2
```

Returns

The value of PI (3.1415926536), the ratio of a circle's circumference to its diameter.

Example

The following examples employ the PI function:

Example 1

`PI( ) * 7 = 21.99` (circumference of a circle with a diameter of 7).

Example 2

`PI( ) * 36 ^ 2 = 4071.50` (area of a circle with a radius of 36).

PMT

Syntax

PMT*(Rate, NPer, PV, FV, {Type})*

Description

The PMT function returns the payment required to repay a loan of *PV,* at an interest rate of *Rate,* where there are *NPer* payments and an ending balance of *FV.* If *Type* is omitted or zero, PMT assumes that the loan is an ordinary annuity. If *Type* is nonzero, PMT assumes that the loan is an annuity due.

Note: Enter negative amounts for money out of your pocket, or positive amounts for money coming to you.

Returns

The payment required to repay a loan of *PV*, at an interest rate of *Rate*, where there are *NPer* payments and an ending balance of *FV.* If *Type* is omitted or zero, PMT assumes that the loan is an ordinary annuity. If *Type* is nonzero, PMT assumes that the loan is an annuity due.

Note: Enter negative amounts for money out of your pocket, or positive amounts for money coming to you.

Example

If you take out a loan for 50000 USD at a rate of 14 percent per year and 120 monthly payments, you can create a PAYMENT cube and compute the payment required to repay the loan as follows:

```
PMT(0.14 / 12, 120, 50000, 0) = -776.33
```

If the loan has a balloon payment of 30000 USD at the end of the 120 months, compute the payment as follows:

```
PMT(0.14 / 12, 120, 50000, -30000) = -660.53
```

If the payments are made at the start of the month rather than the end of the month, use the following formula:

```
PMT(0.14 / 12, 120, 50000, -30000, 1) = -652.92
```


PREV

Syntax

PREV*(Dimension, Data, {Count})*

Description

The PREV function returns the value of *Data* from *Count* members back in *Dimension.* If *Count* is omitted, it is assumed to be 1.

Note: The PREV function operates on detail member names that are persisted in the main record. This function does not use trees to determine the order of members.

Returns

The PREV function returns the value of *Data* from *Count* members back in *Dimension.* If *Count* is omitted, it is assumed to be 1.

Example

To refer to the previous month's sales in a formula, use `PREV(MONTHS, SALES)`.

Suppose that you want to forecast the total monthly receipts for a company, assuming that some of each month's sales are received immediately, some are received in one month, some are received in two months, and some are received in three months. First, define data cubes that contain the estimated percentage of sales received for each time period: PCT_RECV_IMMEDIATELY, PCT_RECV_IN_1_MONTH, PCT_RECV_IN_2_MONTHS, PCT_RECV_IN_3_MONTHS. Next, define a monthly data cube called SALES that contains the sales forecast for each month. Calculate the TOTAL_MONTHLY_RECEIPTS data cube with these formulas:

- RECV_IMMEDIATELY data cube formula:

  ```
  PCT_RECV_IMMEDIATELY * SALES
  ```

- RECV_IN_1_MONTH data cube formula:

  ```
  PCT_RECV_IN_1_MONTH * PREV(MONTHS, SALES)
  ```

- RECV_IN_2_MONTHS data cube formula:

  ```
  PCT_RECV_IN_2_MONTHS * PREV(MONTHS, SALES, 2)
  ```

- RECV_IN_3_MONTHS data cube formula:

  ```
  PCT_RECV_IN_2_MONTHS * PREV(MONTHS, SALES, 3)
  ```

- TOTAL_MONTHLY_RECEIPTS data cube formula:

  ```
  RECV_IMMEDIATELY + RECV_IN_1_MONTH + RECV_IN_2_MONTHS + RECV_IN_3_MONTHS
  ```

RECV_IMMEDIATELY contains the amount received from the current month's sales, RECV_IN_1_MONTH contains the amount received from the previous month's sales, and so on. Add all of these amounts together to calculate the total receipts for the month.

PREVSELF

Syntax

PREVSELF*(Dimension, {Start Value}, {Count})*

Description

The PREVSELF function returns the value of the current data cube from *Count* members back in *Dimension.* When the program is calculating the first *Count* members of *Dimension,* PREVSELF returns *Start Value.* If *Start Value* is omitted, it is assumed to be 0. If *Count* is omitted, it is assumed to be 1.

Note: The PREVSELF function operates on detail member names that are persisted in the main record. This function does not use trees to determine the order of members.

Returns

The PREVSELF function returns the value of the current data cube from *Count* members back in *Dimension.* When the program is calculating the first *Count* members of *Dimension,* PREVSELF returns *Start Value.* If *Start Value* is omitted, it is assumed to be 0. If *Count* is omitted, it is assumed to be 1.

Example

Suppose that you want to forecast sales. For each month, you want to add an estimated Sales Growth to the previous month's sales. When calculating the first month, you want to add sales growth to starting sales. You can do this with the following formula for the SALES cube:

```
PREVSELF(MONTHS, STARTING_SALES) + SALES_GROWTH
```

For the first month, this formula returns the starting sales plus sales growth. For every other month, the formula returns the previous month's sales plus sales growth.

The PREVSELF function is useful for keeping a running balance of transactions. For example, suppose that an analytic model contains monthly data cubes called DEPOSITS, WITHDRAWALS, and BALANCE, and a single value data cube called START_BALANCE. You can calculate the BALANCE cube with the following formula:

```
PREVSELF(MONTHS, START_BALANCE) + DEPOSITS - WITHDRAWALS
```

This formula calculates the ending balance for each month by adding DEPOSITS and subtracting WITHDRAWALS from the ending balance for the previous month. Because no previous balance is available for the first month, the PREVSELF function returns the value of *Start Balance.*

PV

Syntax

PV*(Rate, NPer, Pmt, FV, {Type})*

Description

The PV function returns the Present Value of an investment with a future value of *FV,* where *Pmt* is received for *NPer* periods and is discounted at the rate of *Rate* per period. If *Type* is omitted or zero, PV assumes that the investment is an ordinary annuity. If *Type* is nonzero, PV assumes that the investment is an annuity due.

Note: Enter negative amounts for money out of your pocket, or positive amounts for money coming to you.

Example

Suppose that a machine that sells for 80000 USD saves your company 11000 USD a year for 10 years. Assuming that the money saved could be invested at 8 percent per year, you can calculate the PRESENT_VALUE cube as follows:

```
PV(0.08, 10, 11000, 0) = -73811
```

The present value of the machine is 73811 USD, indicating that you might be better off investing the 80000 USD in another way. But suppose that you can sell the machine for 30000 USD at the end of the 10 years. You can calculate the PRESENT_VALUE cube as follows:

```
PV(0.08, 10, 11000, 30000) = -87707
```

In this case, the present value is higher than the cost of the machine, indicating a profitable investment.

QUARTILE

Syntax

QUARTILE*(Dimension, Values, Quartile, {Type}, {Condition})*

Description

The QUARTILE function returns a quartile of *Values.*

The *Quartile* argument sets which quartile (0, 1 ,2 , 3, or 4) is calculated. If *Type* is zero or omitted, QUARTILE calculates a population quartile; otherwise, QUARTILE calculates a sample quartile. If *Condition* is omitted, the function uses all *Values*. If *Condition* is included, the function uses only those *Values* for which *Condition* is true.

Returns

The QUARTILE function returns a quartile of *Values.* The Quartile argument sets the quartile (0, 1 ,2 , 3, or 4) that is calculated. If *Type* is zero or omitted, QUARTILE calculates a population quartile; otherwise, QUARTILE calculates a sample quartile. If *Condition* is omitted, the function uses all *Values.* If *Condition* is included, the function uses only those *Values* for which *Condition* is true.

Example

For example, suppose that an analytic model contains a data cube called *SCORES* that uses dimensions called *STUDENTS* and *TESTS.*

The following formula calculates the third quartile of the scores for each test.

```
QUARTILE(STUDENTS, SCORES, 3)
```

The following formula calculates the second quartile (also known as the median) of the first ten students:

```
QUARTILE(STUDENTS, SCORES, 2, MEMBER(STUDENTS) <= 10)
```

RAND

Syntax

RAND*()*

Description

The RAND function returns a random decimal number greater than or equal to zero and less than one.

The RAND function uses an industrial strength random number generator with an extremely long period. Thus, it is suitable for use in statistical simulation.

The Analytic Calculation Engine RAND function does not cause a cube to be calculated during every recalculation, unlike in Microsoft Excel. Formulas that use RAND typically refer to some other data in the analytic model, and the analytic calculation engine recalculates the cube only when the other data changes. If you use the RAND function to populate a cube with data for demos or testing and you do not refer to other cubes the data in the cube does not change unless you edit the formula or calculate the data cube.

Returns

A random decimal number greater than or equal to zero and less than one.

Example

`RAND()` returns `0.938119738`.

RATE

Syntax

RATE*(NPer, Pmt, PV, FV, {Type})*

Description

The RATE function returns the rate required to accumulate a future value of *FV* when the present value is *PV,* the number of periods is *NPer,* and the payment is *Pmt.* If *Type* is omitted or zero, RATE assumes that

the investment is an ordinary annuity. If *Type* is nonzero, RATE assumes that the investment is an annuity due.

Note: Enter negative amounts for money out of your pocket, or positive amounts for money coming to you.

Returns

The RATE function returns the rate required to accumulate a future value of *FV* when the present value is *PV,* the number of periods is *NPer,* and the payment is *Pmt.* If *Type* is omitted or zero, RATE assumes that the investment is an ordinary annuity. If *Type* is nonzero, RATE assumes that the investment is an annuity due.

Note: Enter negative amounts for money out of your pocket, or positive amounts for money coming to you.

Example

Suppose that you wish to invest 5000 USD at the end of each year for 10 years. You can create a data cube called RATE_REQUIRED and calculate the rate of return required to earn 100000 USD as follows:

```
RATE(10, -5000, 0, 100000) = 14.69%
```

Now suppose that you initially invest 15000 USD in addition to the yearly payments. Use the following formula:

```
RATE(10, -5000, -15000, 100000) = 7.23%
```

Finally, suppose that you make the payments at the *start* of the year. You can use the following formula:

```
RATE(10, -5000, -15000, 100000, 1) = 6.50%
```

REPLACE

Syntax

REPLACE *(Text, Old, New)*

Description

The REPLACE function replaces all occurrences of *Old* with *New* in text and returns the result.

Example

`REPLACE("StringFun", "Fun", "Number")` returns `StringNumber`.

RETURN

Description

The RETURN function stops the evaluation of a rule and returns the value of the RETURN function's argument.

Example

```
WHILE(&Balance < TARGET_BALANCE,
 IF(&Month > NUMMEMBERS(MONTHS), RETURN(#N/A));
 INC(&Month);
 INC(&Balance, AT(MONTHS, &Month, CASH_FLOW))
);
RETURN(&Month)
```

This formula calculates the number of months required to accumulate a target balance, but returns an error value if the maximum number of months is exceeded. This makes it unnecessary to repeat the condition at the end of the formula.

Note: The RETURN at the end of the formula is not necessary; however, you can use it for clarity.

RIGHT

Description

The RIGHT function returns the right most *Count* characters of *Text.*

Returns

The right most *Count* characters of *Text.*

Example

`RIGHT("StringFun", 3)` returns `Fun`.

ROUND

Syntax

ROUND*(Data, Integer)*

Note: The *Integer* argument is *optional.*

Description

The ROUND function, when you use *only* the first argument, returns the value of *Data* rounded to the nearest whole number. If you use the *optional* second argument, the ROUND function returns the value of *Data* rounded to the number of decimal places that you specify with the *Integer* argument.

The default value for the *Integer* argument is zero.

Example

The following examples employ the ROUND built-in function:

- `ROUND(14)` returns `14`.
- `ROUND(14.3)` returns `14`.
- `ROUND(14, 0)` returns `14`.
- `ROUND(14.3, 0)` returns `14`.
- `ROUND(14.5, 0)` returns `15`.
- `ROUND(14.7, 0)` returns `15`.
- `ROUND(34.56789, 4)` returns `34.5679`.

SELF

Syntax

SELF*( )*

Description

The SELF function returns the current value of the data cube that is being calculated. The SELF function recalculates the data cube only if a certain condition is true; otherwise, the data cube retains its current value.

Returns

The current value of the data cube being calculated.

Example

Suppose that you would like to update your sales forecast on a monthly basis, but you also would like to save the original forecast. If the current forecast is stored in a data cube called *SALES_FORECAST*, you can calculate the ORIGINAL_SALES_FORECAST cube as follows:

```
IF(UPDATE_ORIGINAL, SALES_FORECAST, SELF( ))
```

(See the entry for the IF built-in function if you are unfamiliar with this function.) UPDATE_ORIGINAL is a single value data cube that contains either a true or false value. If UPDATE_ORIGINAL is false, the SELF function returns the current value of ORIGINAL_SALES_FORECAST, thereby leaving the original forecast unchanged. If UPDATE_ORIGINAL is true, the IF function returns the value of SALES_FORECAST, thereby updating the original forecast.

Related Links

IF

SET

Syntax

SET*(VariableExpression)*

Note: The second argument of the SET function can be any valid expression.

Description

The SET function sets a value to a variable.

Example

The following formula sets the *&Index* variable to 1.

```
SET(&Index, 1)
```

SIN

Syntax

SIN*(Data)*

Description

The SIN function returns the sine of *Data,* where *Data* represents an angle in radians.

To convert from degrees to radians, multiply by PI( ) / 180. (The PI function returns the value of PI.)

Returns

The sine of *Data,* where *Data* represents an angle in radians.

Example

The following examples employ the SIN built-in function:

- `SIN(PI( ) / 6)` returns `0.5` (sine of PI / 6 radians).
- `SIN(PI( ) / 2)` returns `1` (sine of PI / 2 radians).
- `SIN(45 * PI( ) / 180)` returns `0.7071067812` (sine of 45 degrees).

SLN

Syntax

SLN*(Cost, Salvage, Life)*

Description

The SLN function returns the depreciation on an asset by using the straight line method, which is a single programming statement. This function returns the same depreciation for each period.

Parameters

Parameter	***Description***
Cost	The cost of the asset.
Salvage	The worth of the asset at the end of its useful life.
Life	The number of periods in the asset's useful life.

Returns

The depreciation on an asset using the Straight Line method. This function returns the same depreciation for each period.

Example

Suppose that you purchase a machine for 6000 USD, and you plan to sell it for 500 USD after five years. You can calculate the depreciation for each year as follows:

```
SLN(6000, 500, 5) = 1100
```

SLOPE

Syntax

SLOPE*(Dimension, Y, X, {Condition})*

Description

The SLOPE function returns the slope of the line that has the closest fit to the points represented by *Y* and *X.* (The slope is the change in *Y* divided by the change in *X.)* If *Condition* is omitted, the function fits the line to all of the members in *Dimension.* If *Condition* is included, the function fits the line only to those members that meet the condition.

Use the SLOPE function together with the INTERCEPT function to find the line with the closest fit to a set of points. You can use these functions to analyze a historical trend, and then use the trend to make forecasts. You can also use these functions to analyze the relationship between different variables, such as sales and travel expense.

Analyzing a Historical Trend

To analyze a historical trend:

1. Calculate the slope for the trend line with this formula for the TREND_SLOPE cube:

   ```
   SLOPE(DATE_DIMENSION, HISTORICAL_DATA,
   MEMBER(DATE_DIMENSION), MEMBER(MONTHS) <= LAST_ACTUAL_DATE)
   ```

   HISTORICAL_DATA is the data cube that you want to analyze. DATE_DIMENSION is the dimension used by the data cube, which is normally a date dimension. Because you want to know how HISTORICAL_DATA is affected by time, use the date index `MEMBER(DATE_DIMENSION)` as the independent (X) argument. LAST_ACTUAL_DATE is a data cube containing the last date that you want to analyze. If you want to analyze all of the dates in DATE_DIMENSION, you may omit the condition.

   See MEMBER.

2. Calculate the intercept for the trend line with the following formula for the TREND_START cube:

   ```
   INTERCEPT(DATE_DIMENSION, HISTORICAL_DATA,
   MEMBER(DATE_DIMENSION), MEMBER(MONTHS) <= LAST_ACTUAL_DATE)
   ```

3. You can now calculate the values for the trend line with the following formula for the TREND_VALUES cube:

   ```
   TREND_START + TREND_SLOPE * MEMBER(DATE_DIMENSION)
   ```

Analyzing the Relationship Between Two Data Cubes

To analyze the relationship between two data cubes:

1. Calculate the slope for the relationship line with this formula for the RELATION_SLOPE cube:

   ```
   SLOPE(DIMENSION, DEPENDENT_VARIABLE, INDEPENDENT_VARIABLE)
   ```

   DEPENDENT_VARIABLE is the variable whose values are influenced by *INDEPENDENT_VARIABLE*. For example, if you want to know how sales are influenced by advertising, SALES is the dependent variable and ADVERTISING is the independent variable. If necessary, you may restrict the analysis to selected members of *DIMENSION* by using a condition for the fourth argument.

2. Calculate the intercept for the relationship line with this formula for the RELATION_START cube:

   ```
   INTERCEPT(DIMENSION, DEPENDENT_VARIABLE, INDEPENDENT_VARIABLE)
   ```

   If you included a condition in the formula for RELATION_SLOPE, be sure to include it in this formula as well.

3. Given an independent variable, you can now estimate a corresponding dependent value with this formula for the DEPENDENT_VALUE cube:

```
RELATION_START + INDEPENDENT_VALUE * RELATION_SLOPE
```

Returns

The slope of the line that has the closest fit to the points represented by *Y* and *X*. (The slope is the change in *Y* divided by the change in *X.)* If *Condition* is omitted, the function fits the line to all of the members in *Dimension.* If *Condition* is included, the function fits the line only to those members that meet the condition.

Example

The following sections provide examples of analyzing a historical trend and analyzing a relationship between data cubes.

Example 1: Analyzing a Historical Trend

Suppose that you want to analyze the trend in historical sales to forecast future sales. The historical sales are stored in a data cube called ACTUAL_SALES that uses a dimension called MONTHS. The date of the last actual value is stored in a data cube called LAST_ACTUAL_DATE. Calculate the sales trend with the following formulas:

- TREND_SLOPE data cube formula:

```
SLOPE(MONTHS, ACTUAL_SALES, MEMBER(MONTHS),
MEMBER(MONTHS) <= LAST_ACTUAL_DATE)
```

- TREND_START data cube formula:

```
INTERCEPT(MONTHS, ACTUAL_SALES, MEMBER(MONTHS),
MEMBER(MONTHS) <= LAST_ACTUAL-DATE)
```

- SALES_TREND data cube formula:

```
TREND_START + TREND_SLOPE * MEMBER(MONTHS)
```

Note: TREND_SLOPE and TREND_START do not use the MONTHS dimension.

Example 2: Analyzing the Relationship Between Data Cubes

Suppose that you want to analyze how UNITS_SOLD has affected SUPPORT_COSTS. Both of these data cubes use a dimension called MONTHS. The date of the last actual value is stored in a data cube called LAST_ACTUAL_DATE. Enter the estimates for future unit sales in a data cube called SALES_FORECAST, and then calculate the resulting SUPPORT_FORECAST data cube as follows:

- RELATION_SLOPE data cube formula:

```
SLOPE(MONTHS, SUPPORT_COSTS, UNITS_SOLD,
MEMBER(MONTHS) <= LAST_ACTUAL_DATE)
```

- RELATION_START data cube formula:

```
INTERCEPT(MONTHS, SUPPORT_COSTS, UNITS_SOLD,
MEMBER(MONTHS) <= LAST_ACTUAL_DATE)
```


- SUPPORT_FORECAST data cube formula:

```
IF(DATE( ) > LAST_ACTUAL_DATE,
RELATION_START + SALES_FORECAST * RELATION_SLOPE, 0)
```

Notice that this example uses a different approach than the previous example. In the first example, you analyzed how sales were affected by time, and then used the results to predict future sales based on the passage of time. In this example, you analyzed how support was affected by sales, and then used the results to predict future support costs based on future sales.

SQRT

Syntax

SQRT*(Data)*

Description

The SQRT function returns the square root of *Data.* If the value of *Data* is negative, SQRT returns an error value.

Returns

The square root of *Data.* If the value of *Data* is negative, SQRT returns an error value.

Example

These examples employ the SQRT built-in function:

- `SQRT(25)` returns *5.*
- `SQRT(2)` returns *1.4142135624.*
- `SQRT(-25)` returns an error value.

STDEV

Syntax

STDEV*(Dimension, Values, {Type}, {Condition})*

Description

The STDEV function returns the standard deviation of *Values.* If *Type* is zero or omitted, STDEV calculates a population standard deviation; otherwise, STDEV calculates a sample standard deviation. If *Condition* is omitted, the function uses all *Values.* If *Condition* is included, the function uses only those *Values* where *Condition* is true.

Example

Suppose that an analytic model contains a data cube called SALES that uses dimensions called PEOPLE and MONTHS.

Use this formula to calculate the standard deviation over time for each person:

```
STDEV(MONTHS, SALES, 0)
```

Use this formula to calculate the standard deviation of sales over 5000 for each month:

```
STDEV(PEOPLE, SALES, 0, SALES > 5000)
```

SYD

Syntax

SYD*(Cost, Salvage, Life, Period)*

Description

The SYD function returns the depreciation on an asset using the Sum-of-the-Years'-Digits method, an accelerated depreciation method. The SYD function takes these arguments:

Parameters

Parameter	***Description***
Cost	The cost of the asset.
Salvage	The worth of the asset at the end of its useful life.
Life	The number of periods in the asset's useful life.
Period	The period for which you wish to determine the depreciation.

Example

Suppose that you purchase a machine for 6000 USD , and you plan to sell it for 500 USD after five years. You can calculate the depreciation for each year as follows:

- `SYD(6000, 500, 5, 1) = 1833`
- `SYD(6000, 500, 5, 2) = 1467`
- `SYD(6000, 500, 5, 3) = 1100`
- `SYD(6000, 500, 5, 4) = 733`

- `SYD(6000, 500, 5, 5) = 367`

TAN

Syntax

TAN*(Data)*

Description

The TAN function returns the tangent of *Data,* where *Data* represents an angle in radians.

To convert from degrees to radians, multiply by PI( ) / 180. (The PI function returns the value of PI.)

Returns

Returns the tangent of *Data,* where *Data* represents an angle in radians.

Example

These examples employ the TAN built-in function:

- `TAN(PI( ))` returns *0* (tangent of p radians).
- `TAN(PI( ) / 4)` returns *1* (tangent of p / 4 radians).
- `TAN(45 * PI( ) / 180)` returns *1* (tangent of 45 degrees).

Related Links

PI

TEXT2MBR

Syntax

TEXT2MBR*(Dimension, Text)*

Description

Converts text to the member with that name in *Dimension.* If there is no member with that name, returns *0.*

Note: This is essentially a linear lookup function, so be careful when using it with dimensions that have a lot of members.

Example

`TEXT2MBR(MONTHS, "January")` returns a new member, January, in the MONTHS dimension.

TEXT2NUM

Syntax

TEXT2NUM *(Text)*

Description

Converts *Text* to a number. This performs a forgiving conversion. For example, dollar signs and commas are ignored, parentheses or a minus sign make the number negative, and % causes the number to be converted as a percentage. If there is no number in the text, the function returns *0*.

Example

These examples employ the TEXT2NUM built-in function:

- `TEXT2NUM("TEN")` returns *10*.
- `TEXT2NUM("$TEN")` returns *10*.
- `TEXT2NUM("-TEN")` returns *-10*.
- `TEXT2NUM("100,000")` returns *100000*.
- `TEXT2NUM("10%")` returns *10%*.

THIS

Syntax

THIS*(Expression)*

Description

The THIS function returns the value of *Expression* for the members being calculated, even if *Expression* is used in a database function. This function enables you to perform complex calculations that relate other members in a dimension to the member being calculated.

To understand the THIS function, you need to understand how database functions work. A database function scans the members in a dimension to calculate a result. For example, suppose that you define the following formula:

```
DAVG(PRODUCTS, ADVERTISING, SALES > 50000)
```

This formula calculates the average advertising for products with sales over 50000 USD. The DAVG function scans the PRODUCTS dimension and evaluates the condition SALES > 50000 for each product. If the condition is true, the DAVG function includes the Advertising for that product in the average. The important point here is that the DAVG function evaluates the condition (SALES > 50000) and the expression (Advertising) for the product being scanned.

Now suppose that you want to calculate the following result for each product: the average advertising for products whose sales are greater than the product being calculated.

Create a data cube called AVG_ADVERTISING_FOR_BETTER_PERFORMERS that uses the PRODUCTS dimension. Its formula should look similar to:

```
DAVG(PRODUCTS, ADVERTISING, SALES > "Sales for the product being calculated")
```

To get the *sales for the product being calculated,* remember that the DAVG function uses the sales for the product being scanned. The solution is to use the THIS function:

```
DAVG(PRODUCTS, ADVERTISING, SALES > THIS(SALES))
```

The THIS function forces the DAVG function to use the sales for the product being calculated. Thus, the condition compares the sales for the product being scanned to the sales for the product being calculated. If the condition is true, the DAVG function includes the Advertising for the product being scanned.

Returns

The value of *Expression* for the members being calculated, even if *Expression* is used in a database function.

THISCUBE

Syntax

THISCUBE*()*

Description

The THISCUBE function returns a reference to the current calculating data cube in a user function.

Returns

A reference to the current calculating data cube in a user function.

Example

```
CHANGE(MONTHS, THISCUBE())
```

The user function in this example calculates the monthly change for each data cube and is used inside an aggregate override user function that affects the SALES, COST_OF_GOODS, and GROSS_MARGIN data cubes.

In this example, the analytic calculation engine performs the same as if you entered these three functions:

- `CHANGE(MONTHS, SALES)`
- `CHANGE(MONTHS, COST_OF_GOODS)`
- `CHANGE(MONTHS, GROSS_MARGIN)`

TRUNC

Syntax

TRUNC*(Data)*

Description

The TRUNC function returns the value of *Data* with the decimals truncated.

Returns

The value of *Data* with the decimals truncated.

Example

The following examples employ the TRUNC built-in function:

- `TRUNC(14)` returns *14*.
- `TRUNC(14.3)` returns *14*.
- `TRUNC(14.7)` returns *14*.

UPPER

Syntax

UPPER*(Text)*

Description

The UPPER function returns *Text* converted to uppercase.

Returns

Text converted to upper case.

Example

`UPPER("StringFun")` returns *STRINGFUN*.

VAR

Syntax

VAR*(Dimension, Values, {Type}, {Condition})*

Description

The VAR function returns the variance of *Values*. If *Type* is zero or omitted, VAR calculates a population variance; otherwise, VAR calculates a sample variance. If *Condition* is omitted, the function uses all *Values*. If *Condition* is included, the function uses only those *Values* where *Condition* is true.

Example

Suppose that an analytic model contains a data cube called SCORES that uses dimensions called STUDENTS and TESTS.

Use the following formula to calculate the variance of the tests for each student:

```
VAR(TESTS, SCORES)
```

Use the following formula to calculate the variance of scores over 75 percent for each test:

```
VAR(STUDENTS, SCORES, 0, SCORES > 0.75)
```

WHILE

Syntax

WHILE*(Condition, Expression)*

Description

The WHILE function supports looping and takes two arguments: a condition that determines whether to continue looping and an expression to evaluate for each iteration.

Example

```
WHILE(&Balance < TARGET_BALANCE .AND. &Month < NUMMEMBERS(MONTHS),
 INC(&Month);
 INC(&Balance, AT(MONTHS, &Month, CA)));
IF(&Month <= NUMMEMBERS(MONTHS), &Month, #N/A)
```

This formula calculates the number of months required to accumulate a target balance.

The IF function returns the value of *&Month,* or an error code if the target balance is not achieved. Notice that it is not necessary to initialize &Balance and &Month because they are initialized to zero before the formula is evaluated.

YEAR

Syntax

YEAR*({Date})*

Description

The YEAR function returns the year of *Date*. If *Date* is omitted, the function returns the year of the calculation date.

Example

Suppose that an analytic model contains a data cube called YEAR_EXAMPLE that uses a dimension called MONTHS, and has the following formula: `YEAR( )`. Because the argument is omitted, YEAR returns the year for each date in the MONTHS dimension.

Now suppose that you plan to build a new building in 2006, and you want to spread the building costs over the quarters of that year. On the other hand, you do not want to allocate the building costs to any other years. If the year and building costs are stored in data cubes called BUILDING_YEAR and TOTAL_BUILDING_COSTS, you can calculate the QTRLY_BUILDING_COSTS data cube as follows:

```
IF(YEAR( ) = BUILDING_YEAR, TOTAL_BUILDING_COSTS / 4, 0)
```

Related Links

IF

Chapter 12

Understanding the Relationship of Analytic Types to Analytic Models

Purpose of Analytic Type Definitions

You create analytic type definitions for use with PeopleSoft Optimization Framework and Analytic Calculation Engine. In PeopleSoft Optimization Framework, you create analytic type definitions to group optimization records, optimization transactions, and optimization plug-ins together as one entity.

See “Creating Analytic Type Definitions” (Optimization Framework).

In Analytic Calculation Engine, you create analytic type definitions to group records and an analytic model together as one entity. You follow the same procedure to create analytic type definitions for both PeopleSoft Optimization Framework and Analytic Calculation Engine.

When creating a new analytic model definition, you create the analytic type definition in this developmental sequence:

1. Create and save an analytic model definition.

   See Understanding the Analytic Model Definition Creation Process.

2. Create an analytic type definition and define records.

   See Example: Working with an Analytic Type and an Analytic Model, “Viewing Record Definitions” (Application Designer Developer’s Guide), “Saving Record Definitions” (Application Designer Developer’s Guide), “Creating New Record Definitions” (Application Designer Developer’s Guide), “Setting Record Properties” (Application Designer Developer’s Guide).

3. Associate the analytic model with the analytic type.

   See “Creating Analytic Type Definitions” (Optimization Framework).

4. In the cube collection properties, map a main record in the analytic type to the cube collection.

   You can also map an aggregate record in the analytic type to the cube collection.

   See Mapping a Cube Collection to Main and Aggregate Records.

5. In the cube collection properties, map the fields in the record to data cubes and dimensions.

   See Mapping Data Cubes and Dimensions to Fields.

When updating an analytic model definition, create an analytic type definition during this developmental sequence:

1. Update the records in the analytic type definition.

See “Configuring Analytic Type Records” (Optimization Framework).

2. Create a new cube collection in the analytic model definition.

   See Understanding Cube Collections, Understanding Types of Cube Collections, Creating Cube Collections, Defining Cube Collection Properties.

3. In the cube collection properties, map one of the updated records to the cube collection.

   See Mapping a Cube Collection to Main and Aggregate Records.

4. In the cube collection properties, map fields of the updated record to data cubes and dimensions.

   See Mapping Data Cubes and Dimensions to Fields.

Example: Working with an Analytic Type and an Analytic Model

The following example illustrates the typical process for creating an analytic type definition to be used with a new analytic model. In this example, you create an analytic type definition called QE_ACE_DGCPROB and define the records that are used in the analytic model. You insert all of these records (except derived/work records) into the analytic type definition and set the attributes of the records.

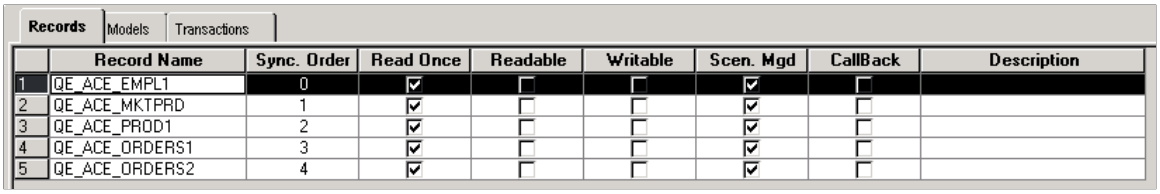
Note: For simplicity, this example maps only one cube collection to a main record, although the analytic model definition in this example contains several records which would be used with several cube collections. The record used in this example is mapped to a read/write cube collection for loading data from the database, receiving end user input, and persisting data back to the database.

See Understanding Cube Collections.

To create an analytic type definition to be used with a new analytic model:

1. Define the records within the QE_ACE_DBCPROB analytic type definition.

   Example of defining the records within the QE_ACE_DBCPROB analytic type definition.



	Record Name	Sync. Order	Read Once	Readable	Writable	Scen. Mgd	CallBack	Description
1	QE_ACE_EMPL1	0	☑	☐	☐	☑	☐	
2	QE_ACE_MKTPRD	1	☑	☐	☐	☑	☐	
3	QE_ACE_PROD1	2	☑	☐	☐	☑	☐	
4	QE_ACE_ORDERS1	3	☑	☐	☐	☑	☐	
5	QE_ACE_ORDERS2	4	☑	☐	☐	☑	☐	

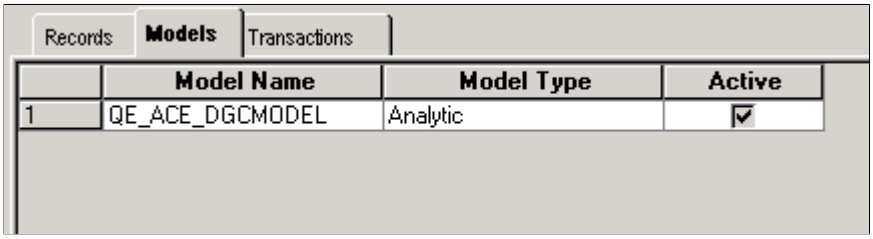
2. Using the information in the following table, you create an analytic model definition called QE_ACE_DGCMODEL, with data cubes and dimensions that are related in this manner.

Data Cube	*Attached Dimensions*
SALARY input data cube	These dimensions are attached to the SALARY data cube: • BUSINESS_UNIT • DEPTID • EMPLID • JOBCODE
EXPENSE input data cube	These dimensions are attached to the EXPENSE data cube: • BUSINESS_UNIT • DEPTID • EMPLID • JOBCODE
BONUS_PERCENT input data cube	These dimensions are attached to the BONUS_PERCENT data cube: • BUSINESS_UNIT • DEPTID • EMPLID • JOBCODE

Note: At this step in the process, you do not create the analytic model definition's cube collections.

3. On the Models tab, you associate the QE_ACE_DBCPROB analytic type to the QE_ACE_DGCMODEL analytic model.

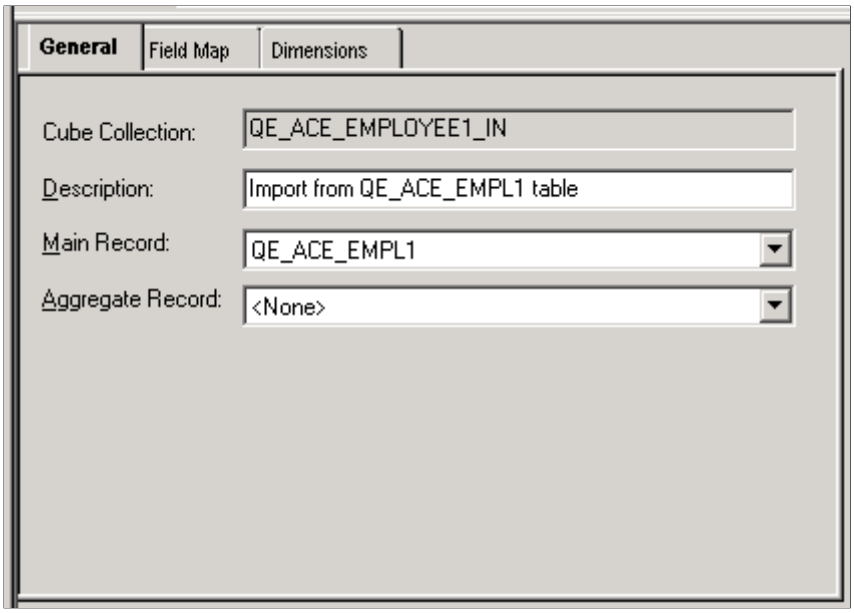
   Example of associating the QE_ACE_DBCPROB analytic type to the QE_ACE_DGCMODEL analytic model.



4. You open the analytic model definition and create a read/write cube collection called QE_ACE_EMPLOYEE1_IN.

5. On the General tab of the cube collection's properties, you map the cube collection to the QE_ACE_EMPL1 main record.

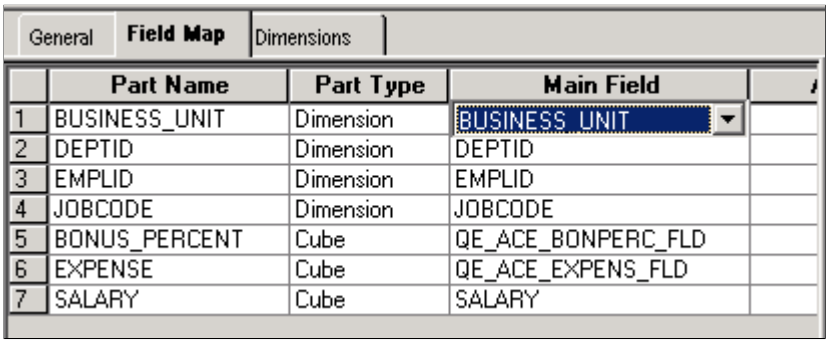
Example of mapping the QE_ACE_EMPLOYEE1_IN cube collection to the QE_ACE_EMPL1 main record.



Note: This cube collection does not use an aggregate record.

6. On the Field Map tab, you map the fields in the QE_ACE_EMPL1 record to the data cubes and dimensions.

   Example of mapping data cubes and dimensions to the fields of the QE_ACE_EMPL1 record.



	Part Name	Part Type	Main Field
1	BUSINESS_UNIT	Dimension	BUSINESS_UNIT
2	DEPTID	Dimension	DEPTID
3	EMPLID	Dimension	EMPLID
4	JOBCODE	Dimension	JOBCODE
5	BONUS_PERCENT	Cube	QE_ACE_BONPERC_FLD
6	EXPENSE	Cube	QE_ACE_EXPENS_FLD
7	SALARY	Cube	SALARY

Note: When mapping dimensions and data cubes, you may want to map dimensions to the key fields in the main record and data cubes to the non-key fields in the main record. The PeopleSoft system, however, does enable you to map dimensions to non-key and data cubes to key fields. To perform the most appropriate mapping, you must have a deeper understanding of the relationship between data cubes and dimensions.

See Data Cubes and Dimensions.

Related Links

“Understanding Analytic Type Definitions” (Optimization Framework)

“Creating Analytic Type Definitions” (Optimization Framework)

Relationship of Record Attributes to Data Caching Behavior

When you create an analytic type definition, how you set the record attributes determines the caching behavior of the data that is used in the analytic model. This topic describes analytic type definition record attributes and their effects on data caching.

Read Once

When you map a cube collection to a record that contains a Read Once attribute in the analytic type, the application data is read only once during analytic model load time. Map cube collections to Read Once records to load data that the user should not change during the analytic model's life cycle. You can specify the Read Once attribute for these record types:

- SQL tables.
- SQL views.
- Dynamic views.
- Query views.

Note: Data cubes that exist in a cube collection that is mapped to a main record with a Read Once attribute cannot exist in any other cube collection that is mapped to a main record with the Read Once attribute.

Readable

When you map a cube collection to a record that contains a Readable attribute in the analytic type, the application data is read during analytic instance load time and is updated with new data after:

- Each analytic model recalculation.

  You recalculate an analytic model by using the AnalyticModel class Recalculate method. The AnalyticModel class is one of the Analytic Calculation Engine classes.

  See “Recalculate” (PeopleCode API Reference).

- Each Save action that is triggered by a PeopleSoft Pure Internet Architecture page with an analytic grid.
- Each time data is updated using the CubeCollection class SetData method.

  The CubeCollection class is one of the Analytic Calculation Engine classes.

  See “SetData” (PeopleCode API Reference).

Map cube collections to Readable records to load data that should be refreshed more than once during the analytic model's life cycle.

You can specify the Readable attribute for the SQL table record type.

Note: Data cubes that exist in a cube collection that is mapped to a main record with the Readable attribute cannot exist in any other cube collection that is mapped to a main record with the Readable or Read Once attributes.
When a cube collection is mapped to either a Writable-only record or a record with the Readable and Writable attributes, all data cubes in the cube collection should share the same set of dimensions.

Writable

When you map a cube collection to a record that contains a Writable attribute in the analytic type, the data in the cube collection is written back to the application database after:

- Each analytic model recalculation.

  You recalculate an analytic model by using the AnalyticModel class Recalculate method. The AnalyticModel class is one of the Analytic Calculation Engine classes.

  See “Recalculate” (PeopleCode API Reference).

- Each Save action that is triggered by a PeopleSoft Pure Internet Architecture page with an analytic grid.

- Each time data is updated using the CubeCollection class SetData method.

  The CubeCollection class is one of the Analytic Calculation Engine classes.

  See “SetData” (PeopleCode API Reference).

You can specify the Read Once attribute for the SQL table record type.

If the analytic type contains a writable-only record that uses a primary key field, you must set up the application to clear the data in the database for the writable record before recalculating the analytic instance.

Note: After the data is written back to the database, the data cubes that are mapped to the writable-only record are cleared from the analytic instance, resulting in 0 or empty values in the analytic grid.
When a cube collection is mapped to either a Writable-only record or a record with the Readable and Writable attributes, all data cubes in the cube collection should share the same set of dimensions.

Scen. Mgd (Scenario Managed)

Use the Scenario Managed attribute to indicate that the record pertains to multiple analytic instances. A scenario managed record is read from and written back to the database according to the other attributes that are specified for the record.

Typically, one user views and edits one analytic instance, although Analytic Calculation Engine supports multiple users per analytic instance.

Records that contain the scenario managed attribute must have a PROBINST key field. The PROBINST key field is used to segment the data of scenario managed records, resulting in a different data set loaded for each analytic instance.

The following table provides an example of a record with a PROBINST key field.

PROBINST key field	*ACCT field*	*TRANS_DATE field*	*REGION field*
BUDGET01	100	January	EUROPE
BUDGET01	100	Feb	EUROPE
BUDGET02	110	Feb	ASIA
BUDGET02	110	March	ASIA
BUDGET03	120	March	USA

In this example:

- The users of the BUDGET01 analytic instance can access only the first and second rows of this record.
- The users of the BUDGET02 analytic instance can access only the third and fourth rows of this record.
- The users of the BUDGET03 analytic instance can access only the fifth row of this record.

Note: Data cubes that exist in a cube collection that is mapped to a main record with the Scenario Managed attribute cannot exist in any other cube collection that is mapped to a derived/work main record.

See “Scenario Management” (Optimization Framework).

Records based on dynamic views can be Scenario Managed. For these records, the associated SQL must contain a meta string for qualifying the analytic instance.

The following example shows a Dynamic View record:

```
SELECT PROBINST, QE_BAM_MONTH_FLD, QE_BAM_REGION_FLD,
     QE_BAM_PRODUCT_FLD, QE_BAM_UNIT_FLD,
     QE_BAM_SALES_FLD, QE_BAM_PRDSALES_FL
FROM PS_QE_BAM_FACT_TBL
WHERE PROBINST = %ProbInst
```

If a Union clause is present the `WHERE PROBINST= %ProbInst` must be added to the individual clauses making up the SQL Union. In addition all the fields that are part of the dynamic view must be selected in the analytic type definition. This is enforced by PeopleSoft Application Designer.

Related Links

“Using the Analytic Type Classes” (PeopleCode API Reference)
“How to Create an Analytic Type Class Object” (PeopleCode API Reference)
“How to Import the Analytic Type Classes” (PeopleCode API Reference)
“Configuring Analytic Type Records” (Optimization Framework)

Synchronization Order

In Analytic Calculation Engine, the synchronization order indicates the order in which the analytic calculation engine reads the records in the analytic type definition. Records that are used as aggregate records should have a higher synchronization order than records that are used as main records.

Chapter 13

Creating Analytic Grids

Understanding Analytic Grid Design

The analytic grid retrieves data from the analytic server and displays it in a grid format on a PeopleSoft Pure Internet Architecture page. This grid is the centerpiece for the Analytic Calculation Engine user interface, enabling end users to view, edit, and drag and drop data from an analytic model's cube collection.

Note: Within an application, a PeopleSoft Pure Internet Architecture page that contains an analytic grid may be referred to as an *interactive report*. Interactive reports are typically read only, but in some cases may also be editable.

Constructing a PeopleSoft Pure Internet Architecture pages that contain an analytic grid consists of these basic steps:

1. In PeopleSoft Application Designer, create an analytic model.

   See Understanding the Analytic Model Definition Creation Process.

2. Use PeopleSoft Application Designer to design the page that contains the analytic grid.

   See “Understanding PeopleSoft Application Designer” (Application Designer Developer’s Guide), “Using PeopleSoft Application Designer” (Application Designer Developer’s Guide).

3. Insert the analytic grid into the page and associate the grid with the analytic model by using the Analytics tab in the Analytic Grid Properties dialog box.

   See Setting Analytic Grid Analytic Properties.

4. Define the initial layout and characteristics of the analytic grid.

   Producing an analytic grid involves many of the same tasks as generating a regular grid. These include inserting and resizing grid controls, inserting and manipulating grid columns, and setting column properties. In addition, you set certain analytic grid properties by using the Analytics tab, Use tab, Label tab, and General tab in the Analytic Grid Properties dialog box.

   See Setting Column Properties for Analytic Grids, Setting Analytic Grid Label Properties, Setting Analytic Grid Use Properties, and Setting Analytic Grid General Properties.

   You can also control the analytic grid layout programmatically using analytic grid APIs.

   See “GetLayout” (PeopleCode API Reference), and “SetLayout” (PeopleCode API Reference).

You can populate the grid with data in two ways: use the PeopleCode analytic grid classes or have the system populate the analytic grid with data automatically.

See “Understanding the Analytic Calculation Engine Classes” (PeopleCode API Reference), “Using the Analytic Grid in PeopleCode” (PeopleCode API Reference).

To populate the analytic grid data automatically:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, create or open a page definition.
3. Insert into the page any control from which you can obtain analytic instance values—for example, an edit box.
4. Set this control's properties to the analytic instance (select the appropriate record and field name on the Record tab).

   This will be the analytic instance field you select in the Analytics tab of the Analytic Grid Properties dialog box.
5. Drag the appropriate record onto the grid.

Although the developer determines the initial layout of the analytic grid using PeopleSoft Application Designer, one of the primary advantages of the analytic grid is that end users can also modify the layout of the grid at runtime. Among other things, end users can use the analytic grid to:

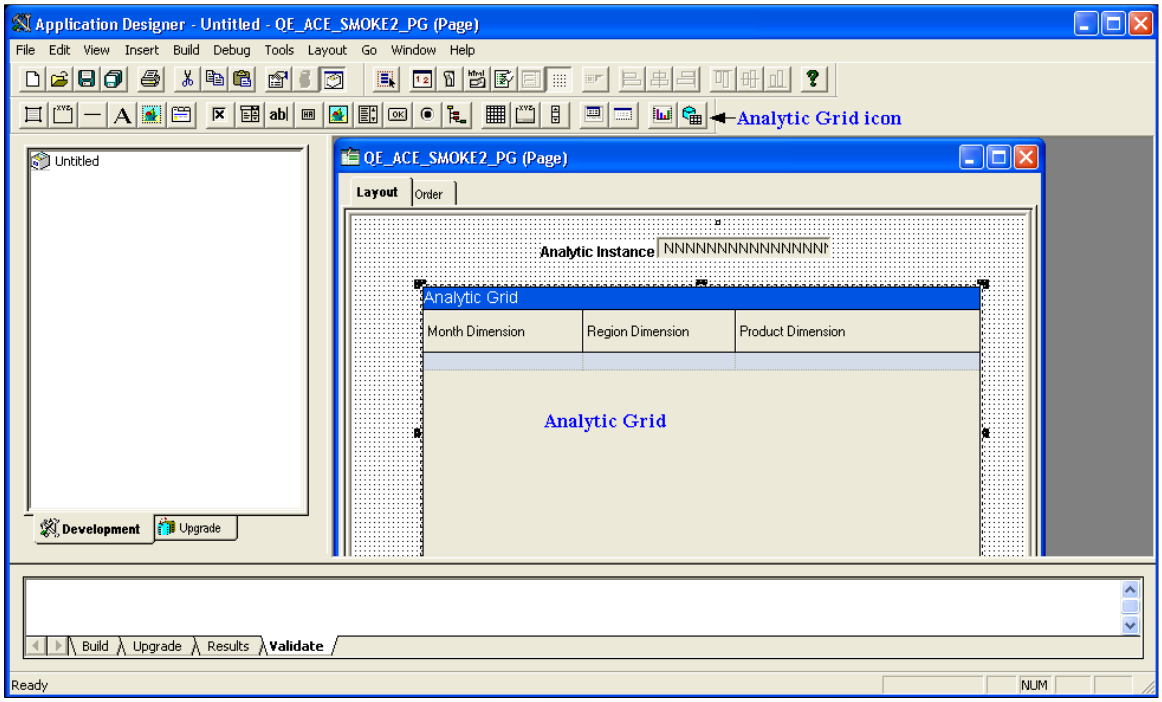
- Pivot data—for example, swap row and column orientations.
- Perform hierarchy-related actions such as expanding, collapsing, drilling in, and drilling out in the data.
- Slice data, for example, view a subset of a multidimensional array.
- Hide non-dimensional fields.
- Modify data.
- Save the current view settings.
- Restore the defaults as specified by the application.

See “Working with Analytic Grids” (Applications User’s Guide).

Components for Working with Analytic Grids

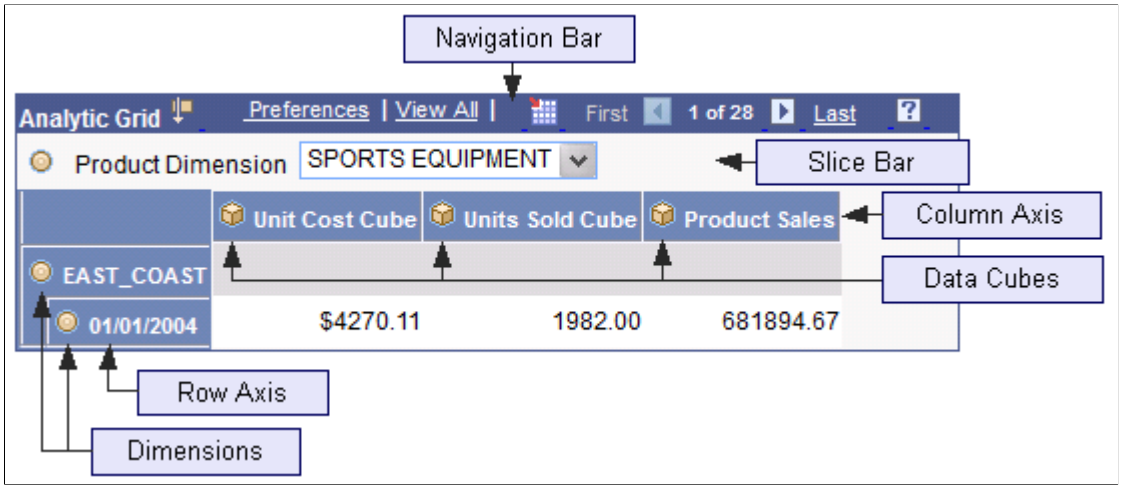
You design analytic grids using PeopleSoft Application Designer. In addition, you may need to work with analytic grids in the runtime environment.

This example shows the PeopleSoft Application Designer areas used to create analytic grids.



Term	*Definition*
Analytic Grid button	Select and then drag to insert an analytic grid into the page.
Analytic grid	Contains the grid itself. You can modify the grid's properties by double-clicking to display the Analytic Grid Properties dialog box. (Click anywhere except the column headings.) You can also double-click a grid column to display the properties box for modifying the column. In addition, you can change the order of columns on either the Layout or Order tab. See Inserting and Manipulating Analytic Grid Columns.

This example shows the areas in the Analytic Grid section used to modify analytic grids at runtime.



Term	*Definition*
Navigation Bar	Enables end users to navigate through the displayed data set. Also contains a link to drag and drop instructional text.
Slice Bar	Enables end users to view selected portions, or slices, of the data, for instance, the sales of one category of product or the sales from a single region.
Column Axis	Displays the designated cubes or dimensions across the top of the analytic grid. Also contains icons for expanding or collapsing items. **Note:** Dimension on Column Axis can be expanded up to only four levels.
Row Axis	Displays the designated cubes or dimensions along the left-hand side of the analytic grid.
Data Set	Displays the data from the loaded analytic instance.

See “Understanding Analytic Grids” (Applications User’s Guide).

Inserting and Resizing Analytic Grid Controls

To insert an analytic grid on a page:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition.
3. Select **Insert** > **Analytic Grid.**
4. Drag to place the grid on the page.
5. If you need to adjust the grid width, drag the horizontal or vertical control handles.

   The grid width should be roughly equivalent to the columns that you insert into the grid. Otherwise, the grid might appear too wide or narrow at runtime.

   Note: The grid height depends on the data contained in the grid.

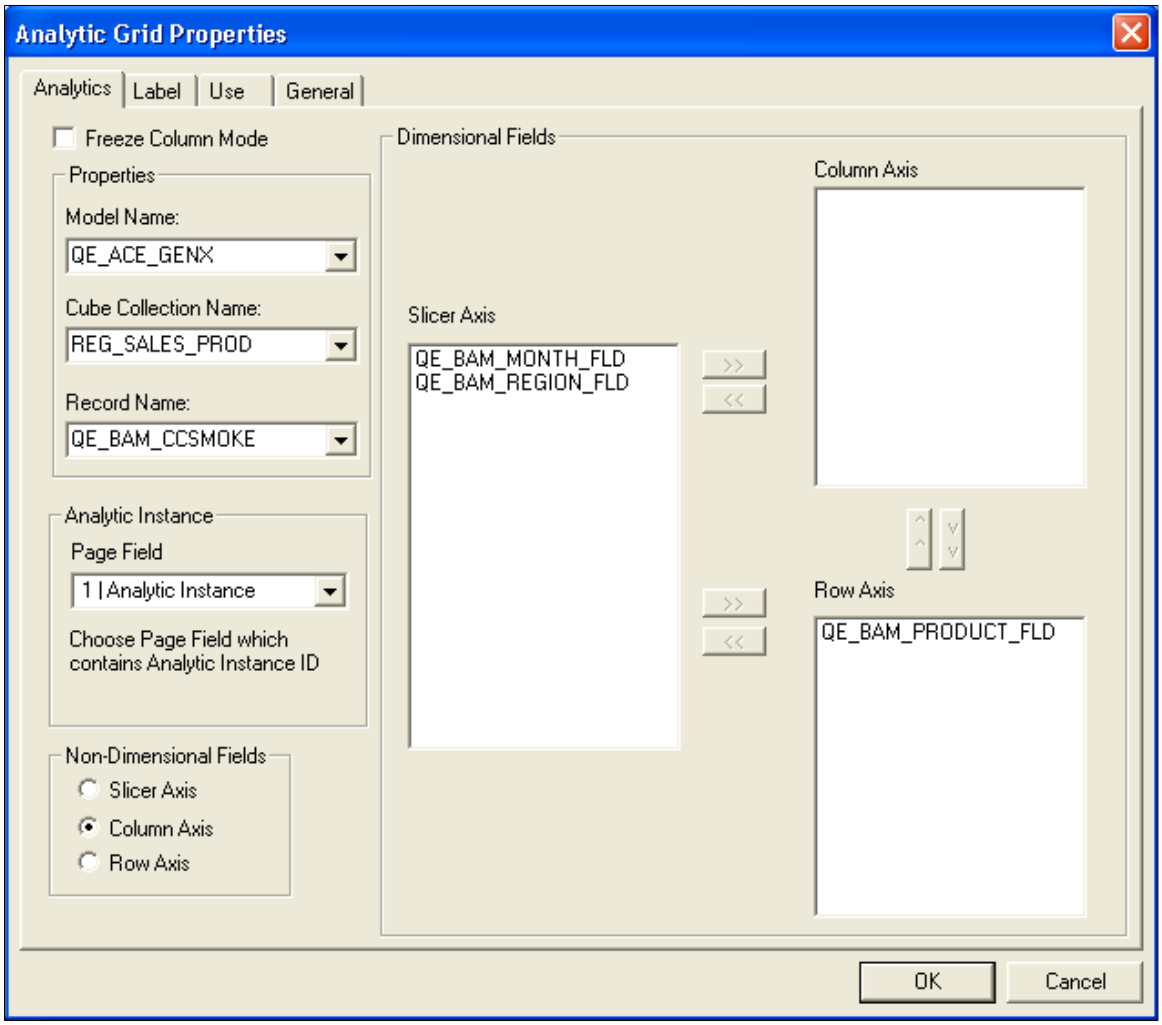
Note: These steps insert an analytic grid control on the page, but so far you have not associated this analytic grid with the relevant model or record definition. You establish this association by means of the Analytic Grid Properties dialog box.

Setting Analytic Grid Analytic Properties

To set analytic grid Analytic properties:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition.
3. Select **Insert** > **Analytic Grid** to add a analytic grid.
4. Access the Analytic Grid Properties dialog box by double-clicking anywhere on the analytic grid other than on the column headings.
5. Use the Analytics tab to set analytic model association and axis display properties.

This example illustrates the fields and controls on the Analytic Grid Properties dialog box - Analytics tab. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Freeze Column Mode	Select this check box if you want to freeze the columns of the grid when it's displayed to the end user. If you select this check box, the only enabled field in the dialog box is the **Record Name**field.
Model Name	Select the analytic model that you want to associate with the current analytic grid. You can select from any of the analytic models in the database.
Cube Collection Name	Select a cube collection from the analytic model. **Note:** This drop-down list box only displays presentation cube collections, which have work/derived records associated with them. Any other cube collections do not appear. See Presentation Cube Collections.
Record Name	Select either a main record or an aggregate record from the cube collection. The aggregate record is selected by default, if applicable. If there's no aggregate record, this field is populated with the main record. If you've selected the **Freeze Column Mode** check box, only work/derived records are displayed.
Analytic Instance	Specify the page field containing the analytic instance ID—that is, the instance of the analytic model that is displayed in this analytic grid.

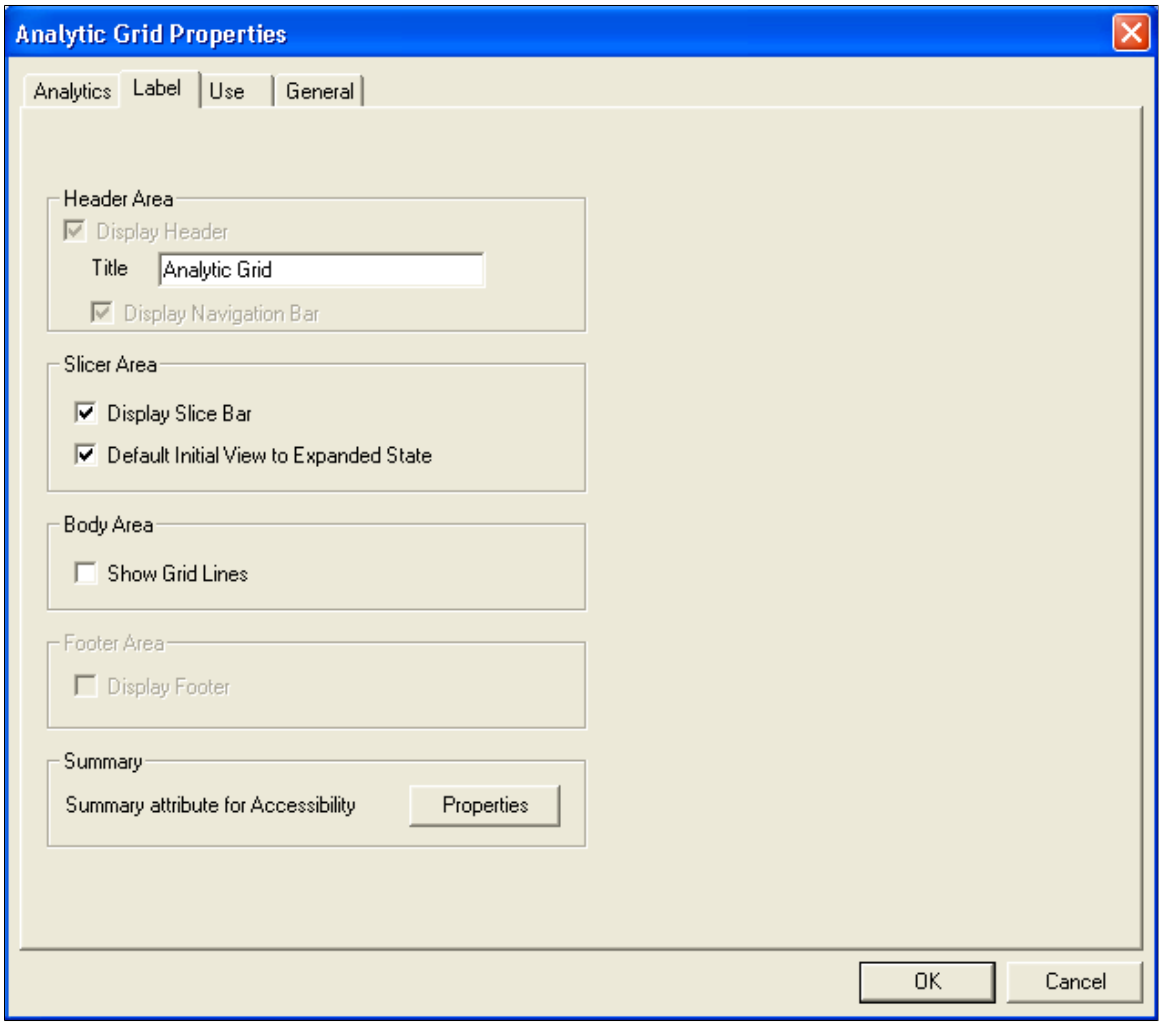
Field or Control	*Description*
Non-Dimensional Fields	Select *Slicer Axis* to set the non-dimensional fields axis to the slicer axis. Select *Column Axis* to set the non-dimensional fields axis to the column axis. Select *Row Axis* to set the non-dimensional fields axis to the row axis. ***Warning!*** All fields that are mapped to dimensions are considered dimensional fields. All fields that are mapped to data cubes are considered non-dimensional fields. The non-dimensional fields referred to within this dialog box are mapped to data cubes. However, any field that is not mapped to anything is also considered a non-dimensional field. If the application developer wants to include such non-dimensional fields (those not mapped to anything) with the analytic grid, he or she must populate them using the RowInit method or their value will be zero. **Note:** If a field is designated as invisible, that property is sometimes honored and sometimes not honored within the analytic grid. If the field is a dimension on the slicer axis, the property is honored. If the field is a dimension on the row or column axis, the invisible property is not honored. If the field is a cube, the invisible property is honored on the column but not the row axis.
Dimensional Fields	Use the **Slicer Axis** option to set which dimensional fields are used for the slice bar. Dimensions that have filter functions applied will behave differently depending on whether they reside on the column axis/row axis or the slice bar. See the following for details. See Filter User Functions. Use the **Column Axis** option to set which dimensional fields are used for the column axis. Use the **Row Axis** option to set which dimensional fields are used for the row axis. You can select any dimensional field and move it from one list box to another by using the appropriate arrow keys. (The arrow keys are grayed out if you do not have a field selected.) The order of the fields on any particular axis—as end users will see it—is determined by the order in which their columns appear in the analytic grid, not by their order in the analytic model. You can change the column order by using drag and drop. By default, all but the last dimension appear on the slice bar axis; the last dimension appears on the row axis, and the data cubes appear on the column axis. See Inserting and Manipulating Analytic Grid Columns.

Setting Analytic Grid Label Properties

To set analytic grid label properties:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition.
3. Select **Insert** > **Analytic Grid** to add an analytic grid.
4. Access the Analytic Grid Properties dialog box by double-clicking anywhere on the analytic grid other than on the column headings.
5. From the Analytic Grid Properties - Analytics tab, select the Label tab.

This example illustrates the fields and controls on the Analytic Grid Properties dialog box - Label tab. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Display Header	Select if you want to display preferences and the link to download to Excel to the end user in the header.
Title	Enter a title that displays in the upper-left corner of the analytic grid. **Note:** You can also modify this title at runtime by using the AnalyticGrid classes.
Display Navigation Bar	Select if you want to display navigational elements for the grid in the header, such as *First, Last, View All,* and so on. Preferences and the link to download to Excel still display to the end user.
Display Slice Bar	Select for the slice bar to appear in the analytic grid. By default, this check box is selected. This item is not available if the **Freeze Column Mode** check box is selected in the Analytics tab.
Default Initial View to Expanded State	Select to have the slice bar appear expanded to the end user initially. Clear to have the slice bar initially appear collapsed to the end user. This item is not available if the **Freeze Column Mode** check box is selected in the Analytics tab. **Note:** The slice bar is expanded by default.
Show Grid Lines	Select to display grid lines to the end user.
Display Footer	Select if you want to display preferences and the link to download to Excel to the end user in the footer. No navigational elements are displayed in the footer.
Summary	Enables you to provide a brief description of the functionality and content of the grid area. This property is pertinent for users who access the application by using screen readers.

Analytic grid label properties such as the label text, grid lines, slicer, and summary text can also be set through PeopleCode.

See “AnalyticGrid Class Properties” (PeopleCode API Reference).

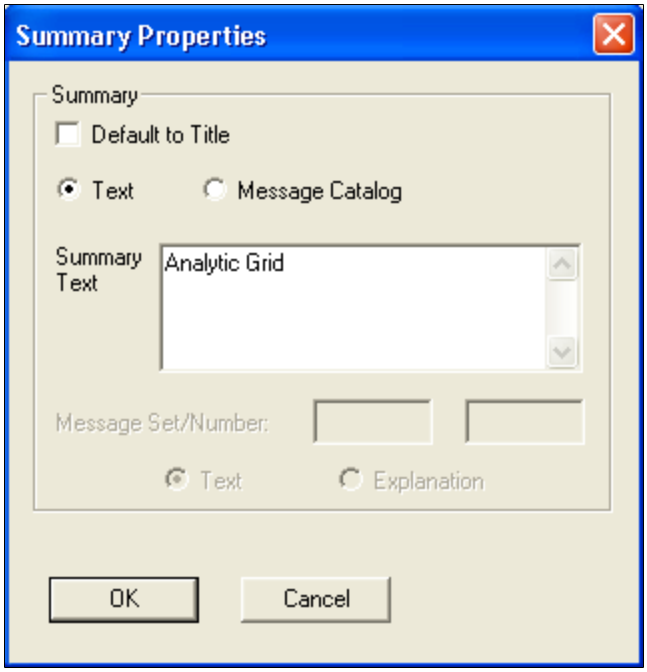
Setting Analytic Grid Label Properties

You use the Summary Properties dialog box to set the analytic grid label properties.

Navigation:

Click the **Properties** button on the Analytic Grid Properties - Label dialog box.

This example illustrates the fields and controls on the Summary Properties dialog box. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Default to Title	Select this option to have the summary property the same as the grid title. Clear this option to activate the **Text** and **Message Catalog** options.
Text	Select this option to enter up to 254 characters in the **Summary Text** field. Selecting this option disables all **Message Catalog** option related fields.
Message Catalog	Select this option to choose a message stored in the **Message Catalog.** Selecting this option disables all **Text** option related fields. Select one of these two options: • *Text*: Select this option to use only the message text from the message catalog. • *Explanation*: Select this option to use only the message explanation from the message catalog.
Summary Text	The default summary text value is the same as the Title of the grid area. You can also enter static text or use the Message Catalog to store the summary information.

To change the summary properties:

1. Open the grid area.
2. Access the Label tab.
3. Click the **Properties** button located in the Summary group box.

   The Summary Properties dialog box appears.
4. Clear the **Default to Title** option to activate the other **Summary** options.
5. Enter static text or enter a **Message Set** and **Number** to retrieve information from the Message Catalog.

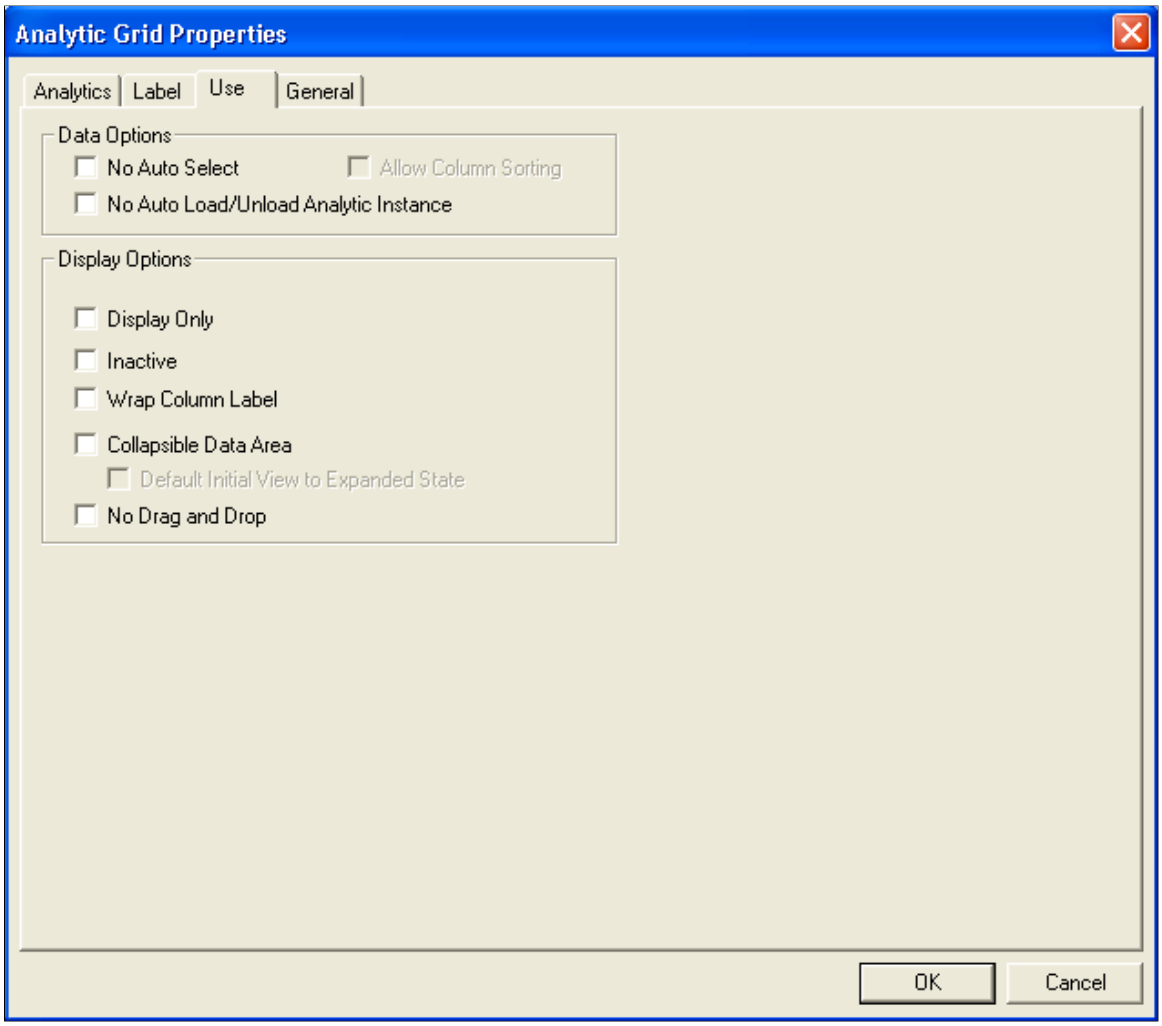
Setting Analytic Grid Use Properties

To set analytic grid Use properties:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition.
3. Select **Insert** > **Analytic Grid** to add a analytic grid.
4. Access the Analytic Grid Properties dialog box by double-clicking anywhere on the analytic grid other than on the column headings.
5. Select the Use tab.

   The Analytic Grid Properties - Use tab appears.

This example illustrates the fields and controls on the Analytic Grid Properties dialog box - Use tab. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
No Auto Select	Select to suppress the system from automatically retrieving data from the analytic calculation engine. If you select the **No Auto Select** check box, you must use the LoadData method to load the analytic grid with data. See “LoadData” (PeopleCode API Reference). This item is not available if the **Freeze Column Mode** check box is selected in the Analytics tab.

Field or Control	*Description*
No Auto Load/Unload Analytic Instance	Determines whether and how you load the analytic instance for the analytic grid. • If you do not select this option, you can supply an analytic instance ID on the Analytics tab to have the analytic grid automatically load that analytic instance. If the analytic grid auto loads the analytic instance, it uses the default timeout setting and also recalculates the model. The analytic instance is unloaded when the user navigates out of the component. Another option for supplying the analytic instance ID is to use the Analytic Grid Class SetAnalyticInstance method. See “SetAnalyticInstance” (PeopleCode API Reference). • If you select this option, the analytic instance is not loaded automatically. In this case, use the PeopleCode analytic instance classes to load the analytic instance into the grid. If you do not use a PeopleCode program to load the data, no analytic instance is loaded. This item is not available if the **Freeze Column Mode** check box is selected in the Analytics tab.
Sort Order	This check box is enabled only if the **Freeze Column Mode** check box is selected in the Analytics tab. Selecting this option enables the end user to sort the data in the frozen columns.
Display Only	In some cases, you might design grids that enable end users to view but not change information. Select this check box if you do not want the end user to enter data into the fields in any of the rows. End users can still manipulate the grid to display a new view of their data, but they cannot update the actual data displayed in the analytic grid. If the grid is display only, obviously all the fields within the grid are display only. However, if the grid is not display only, there are several possibilities: • If fields are display only, the analytic grid honors that. • If fields are not display only, they are editable as long as they are nonaggregate values. • If fields are dimensions, they are display only unless the dimension is on the slicer axis.
Inactive	The analytic grid does not display to end users and no data is loaded into the analytic grid data, thus no data is available to the application developer using PeopleCode.

Field or Control	*Description*
Wrap Column Label	Select whether column labels wrap if they are too long to fit within the column at its current width. If you do not select this option, columns widen as needed to accommodate long column labels.
Collapsible Data Area	Select this option so that the data area for your analytic grid can be collapsed into a header bar with an icon that the end user must click to expand it. Selecting the **Collapsible Data Area** option activates the *Default Initial View to Expanded State* check box.
Default Initial View to Expanded State	Select whether the initial view of the grid is expanded or collapsed. It is expanded by default. **Note:** This check box is available only if the **Collapsible Data Area** option is selected.
No Drag and Drop	Specify whether the end user can drag and drop cubes, dimensions, and so on at runtime. This item is not available if the **Freeze Column Mode** check box is selected in the Analytics tab.

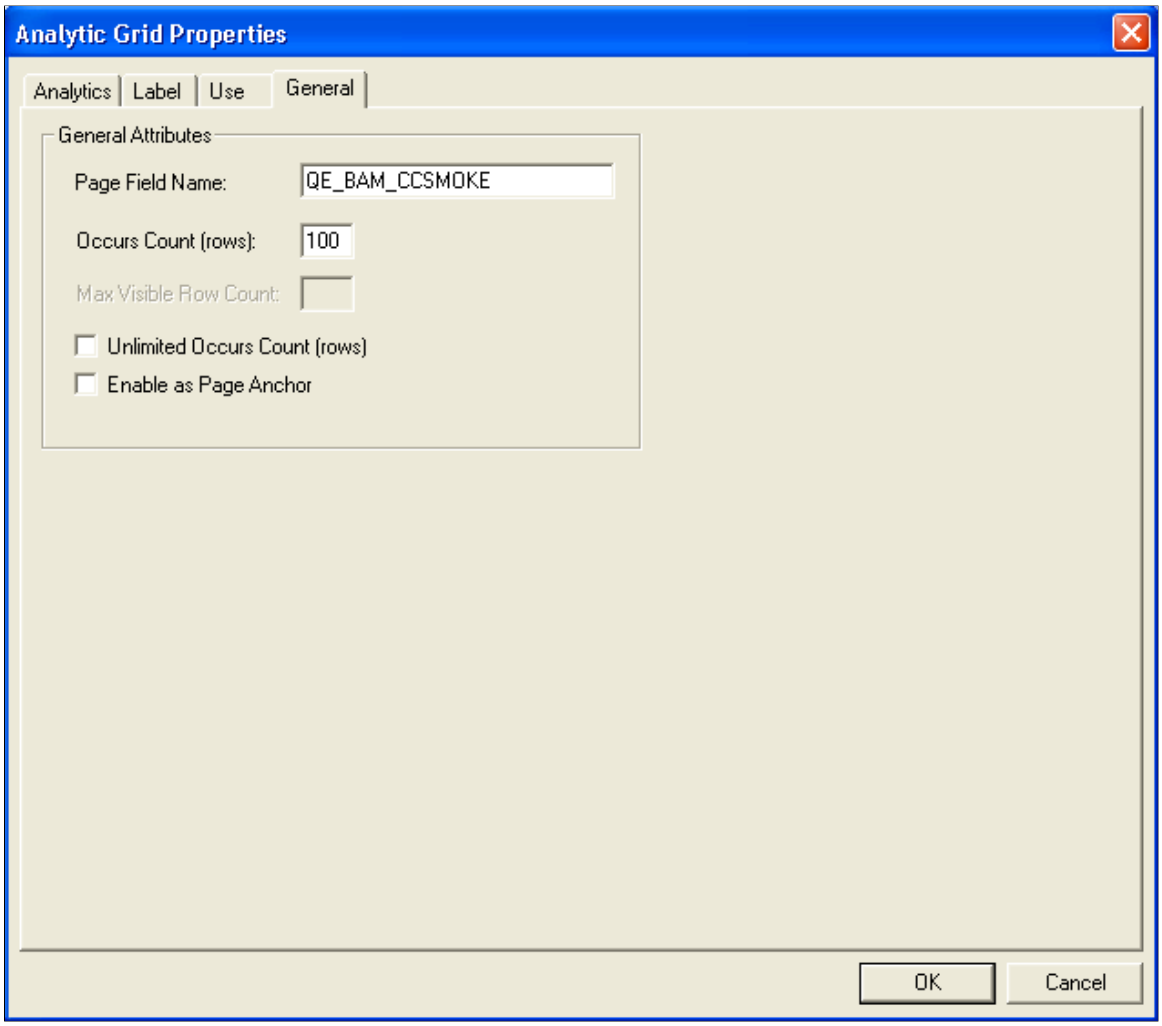
Setting Analytic Grid General Properties

To set analytic grid General properties:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition.
3. Select **Insert** > **Analytic Grid** to add a analytic grid.
4. Access the Analytic Grid Properties dialog box by double-clicking anywhere on the analytic grid other than on the column headings.
5. Select the General tab.

   The Analytic Grid Properties - General tab appears.

This example illustrates the fields and controls on the Analytic Grid Properties dialog box - General tab. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Page Field Name	Specify a grid name consisting of any combination of uppercase letters, digits, and the symbols #, $, @, and _. The default is the name of the main record for the analytic grid; however, you can rename the grid as long as you use a unique name for the page or component. This name is used by the PeopleCode GetAnalyticGrid function to create a grid definition. See “GetAnalyticGrid” (PeopleCode Language Reference).
Occurs Count (rows)	Determines the vertical page size—that is, how many rows of data are displayed initially at runtime. The occurs count is set to 1 by default. If you set the occurs count to 30 rows, for example, the end user sees 30 rows of data at a time.

Field or Control	*Description*
Max Visible Row Count (maximum visible row count)	Specify the maximum number of visible rows. This item is only available if the **Freeze Column Mode** option is selected in the Analytics tab.
Unlimited Occurs Count (rows)	Sets the occurs count to unlimited, which means that the end user sees all rows of data. Selecting this check box disables the **Occurs Count** option because it is no longer applicable. In addition to setting an occurs count, the developer can set a threshold by using ACEGRDROWS in PeopleSoft Personalizations to limit how many rows of data are displayed in the grid. (The analytic grid supports a minimum of 2 rows: one for column axis and one for data; and it supports a maximum of 101 rows: one for column axis and 100 for data). This threshold works with the **View All** link in the grid. If the number of rows of data returned is less than the threshold, this link reads View All and, when the end user clicks it, all records appear. However, if the number of rows of data returned is more than the threshold, the **View All** link changes to View X, where *X* is the value of the threshold. (This link is a toggle: clicking it switches between displaying the occurs count specified in the properties dialog box and the threshold specified in user personalizations). If the occurs count is greater than or equal to the threshold, the threshold takes precedence. See “Working with System Personalization Options” (Security Administration).
Enable as Page Anchor	Select to apply an anchor tag to serve as a jump destination on the page. See “Specifying Type Properties for Push Buttons or Links” (Application Designer Developer’s Guide), and “Setting General Properties” (Application Designer Developer’s Guide).

Inserting and Manipulating Analytic Grid Columns

This topic discusses how to insert and manipulate analytic grid columns.

Note: The analytic grid supports a minimum of two columns: one for row axis and one for data; and it also supports a maximum of 101 columns: one for column axis and 100 for data. The default maximum number of columns is 41. You can also use the ACEGRDCOLS option in PeopleSoft Personalizations to set the number of columns displayed in the analytic grid. If necessary, the analytic grid provides a scroll bar that enables end users to scroll through all columns.
When the **Freeze Column Mode** check box is selected in the Analytic Grid Properties - Analytics tab, all columns in the analytic grid are displayed. No threshold is placed on the maximum number of columns. The user profile setting ACEGRDCOLS, has no affect if the **Freeze Column Mode** option is selected. The analytic grid provides a horizontal scroll bar to scroll through all the columns in the grid. The size of the analytic grid at runtime is a factor of the design time size of the analytic grid and the actual browser page width. The browser width is used only when the page is generated, so the size of the grid does not change as the user resizes the browser window. It does change on the next server trip when the page is regenerated.
The order of the dimensions in the analytic grid does not need to match the order of dimensions in the analytic model. The order in the model is for the purposes of calculation, whereas the order of columns in the analytic grid determines the order that displays to the end user.

Inserting Analytic Grid Columns

Use one of the following methods to insert an analytic grid column:

- Select a page control from the Insert menu and click the analytic grid.

  Note: You can insert these page controls within analytic grids: edit boxes, long edit boxes, images, push buttons, and check boxes.
  A cube formatted as dimension member should have field type of edit box.

- Drag a page field from within the current page, or from another page, into the analytic grid.
- Copy and paste a page field or record field.
- Drag a definition (such as a record field definition, a page field definition, or an entire record definition) from the project workspace to the analytic grid.

The Analytic grid columns should all be bound to the same record definition to which the underlying cube collection is attached; the only exception is the columns used for related display/related edit. All the fields in the record definition that are mapped to either a dimension or a field should have a representative column in the analytic grid.

Deleting Analytic Grid Columns

To delete an analytic grid column:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition.
3. Select a column by clicking the column heading of the analytic grid.

Be sure that you select only the column and not the analytic grid as a whole; otherwise, you might delete the entire grid instead of just the column. The column is selected when it turns black. The whole analytic grid is selected when control boxes appear around the edges of the grid.

4. Press the **Delete** key.

Moving Analytic Grid Columns on the Layout Tab

To move analytic grid columns on the Layout tab:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition and access the Layout tab.
3. Click to select a column heading of the analytic grid.
4. Drag the column to its new location.
5. Release the mouse button over the column that is to the left of the new location.

Note: The order of columns here determines the order in which they display to end users. However, to determine the axis on which fields appear, you use the Analytics tab in the Analytic Grid Properties dialog box. All non-dimensional fields can appear on one axis only.

Moving Analytic Grid Columns on the Order Tab

To move analytic grid columns on the Order tab:

1. Select **Start** > **Programs** > **PeopleTools 8.5x** > **Application Designer** to access PeopleSoft Application Designer.
2. After signing in to the PeopleSoft Application Designer, open a page definition and access the Order tab.
3. Select the column row by clicking the row number.

   The analytic grid is identified as such in the Type column and appears in green. All columns in the analytic grid are directly below this analytic grid row and appear in a lighter green.
4. Drag the row to the new position in the grid.

   A red line indicates the new position of the column before you release it.

Note: You cannot move a column outside of the grid when working on the Order tab. Similarly, you cannot move an existing page control from elsewhere on the page into the grid. You can perform both of these operations on the Layout tab.

Resizing Analytic Grids

You can resize analytic grids in PeopleSoft Application Designer by dragging the right border of the grid. The size of individual columns is determined by the data they contain. The height of the analytic grid is determined by the number of rows it contains. If the number of columns extends beyond the maximum width of the page, a scroll bar is introduced to enable end users to scroll through the columns.

Note: When designing analytic grids, keep in mind that the row header, which you do not see in PeopleSoft Application Designer, takes up some of the width of the analytic grid that is displayed to end users.

Setting Column Properties for Analytic Grids

After you insert the page control or field into your grid, you can set the properties for that field as you would set properties for any other page control. Access the field properties by double-clicking the column heading. This properties dialog box behaves much as it does for ordinary grids.

See “Setting Page Field Properties for Controls” (Application Designer Developer’s Guide).

Note: Settings that you select in the properties dialog, which comes up when you click an individual column in the analytic grid, override the settings that you select in the Analytic Grid Properties dialog box.

In addition, related display fields and related edit fields behave the same for analytic grids as they do for ordinary grids.

See “Creating Display Control and Related Fields” (Application Designer Developer’s Guide).

Manipulating the Analytic Grid at Runtime

Your job as an application developer is not finished at design time. You can perform several tasks at runtime to ensure that the analytic grid works as desired:

- All data for the analytic grid can be accessed using the PeopleCode RowSet class, as with the regular grid.

  You can write business logic to manipulate this data.

  See “Understanding Data Buffer Access” (PeopleCode Developer’s Guide), “Understanding Data Buffer Classes Examples” (PeopleCode Developer’s Guide), “Instantiating Rowsets Using Non-Component Buffer Data” (PeopleCode Developer’s Guide), “Using the Analytic Grid in PeopleCode” (PeopleCode API Reference).

- If necessary, use the RowInit event to populate data for application data fields.

  The record definition associated with a cube collection—and, therefore, with the analytic grid—can have fields that are not mapped to the cube collection's cubes or dimensions. These extra application data fields are treated as non-dimensional fields. They are not populated automatically by the Analytic Calculation Engine. The RowInit event is fired for each row as it is retrieved from the database, and

provides the opportunity for the application to populate these application fields with the appropriate data.

See “RowInit Event” (PeopleCode Developer’s Guide).

- Use PeopleCode to manipulate the analytic model and analytic grid data, as well as change the display of the analytic grid.

  See “Understanding the Analytic Calculation Engine Classes” (PeopleCode API Reference), “Using the Analytic Grid in PeopleCode” (PeopleCode API Reference).

Chapter 14

Viewing and Debugging Analytic Models

Understanding the Analytic Model Viewer

The Analytic Model Viewer is a debugging tool with which you can view intermediate results from calculations and modify data when testing calculations. This tool is provided in the runtime environment through the PeopleSoft Pure Internet Architecture (PeopleSoft PIA). Using the Analytic Model Viewer, you can view metadata (such as virtual data cubes) that may not appear to the end user, and edit analytic model data to see how your results would then change.

Even though you create analytic model definitions in PeopleSoft Application Designer, you need to view an analytic instance of the analytic model during runtime to determine whether the analytic calculation engine is performing its calculations as intended. Simply viewing an analytic instance within the application during runtime is not sufficient, because some parts and data of the analytic model may not be visible to end users. Using the Analytic Model Viewer you can view, analyze, and debug all cube collections in the model. The Analytic Model Viewer operates during runtime because it depends on the analytic calculation engine (for calculation) and the analytic server (for data transformation). In the Analytic Model Viewer, you view a specific analytic instance of the analytic model.

Although you can modify data from within the Analytic Model Viewer, you cannot change metadata, including rules, from this environment. Conversely, from within the analytic model in PeopleSoft Application Designer, you can change metadata but cannot change the data itself. The ideal solution is to simultaneously view the design time analytic model, in PeopleSoft Application Designer, and the runtime analytic model, in the Analytic Model Viewer. This arrangement enables you to compare calculation results, and to change either data or metadata, based upon your needs.

After you load an analytic instance for use in the Analytic Model Viewer, closing the viewer does not automatically unload the analytic instance. You must do that manually from the Analytic Instance Load/Unload page.

Note: You can view causes, effects, and inputs in both PeopleSoft Application Designer and the Analytic Model Viewer. PeopleSoft Application Designer displays the causes, effects, and inputs of data cubes and cube collections. The Analytic Model Viewer displays not only causes, effects, inputs, but also displays overrides (which are determined during runtime), thus providing a more detailed display of causes, effects, and inputs.
The Analytic Model Viewer utility is available only to system administrators. You cannot change its security settings to make it available to other users.

Viewing Analytic Model Properties

This topic provides an overview of the Analytic Model Viewer.

Understanding Analytic Model Properties

Use the Analytic Model Viewer page (PTATSRCHPG2) to search for existing analytic models.

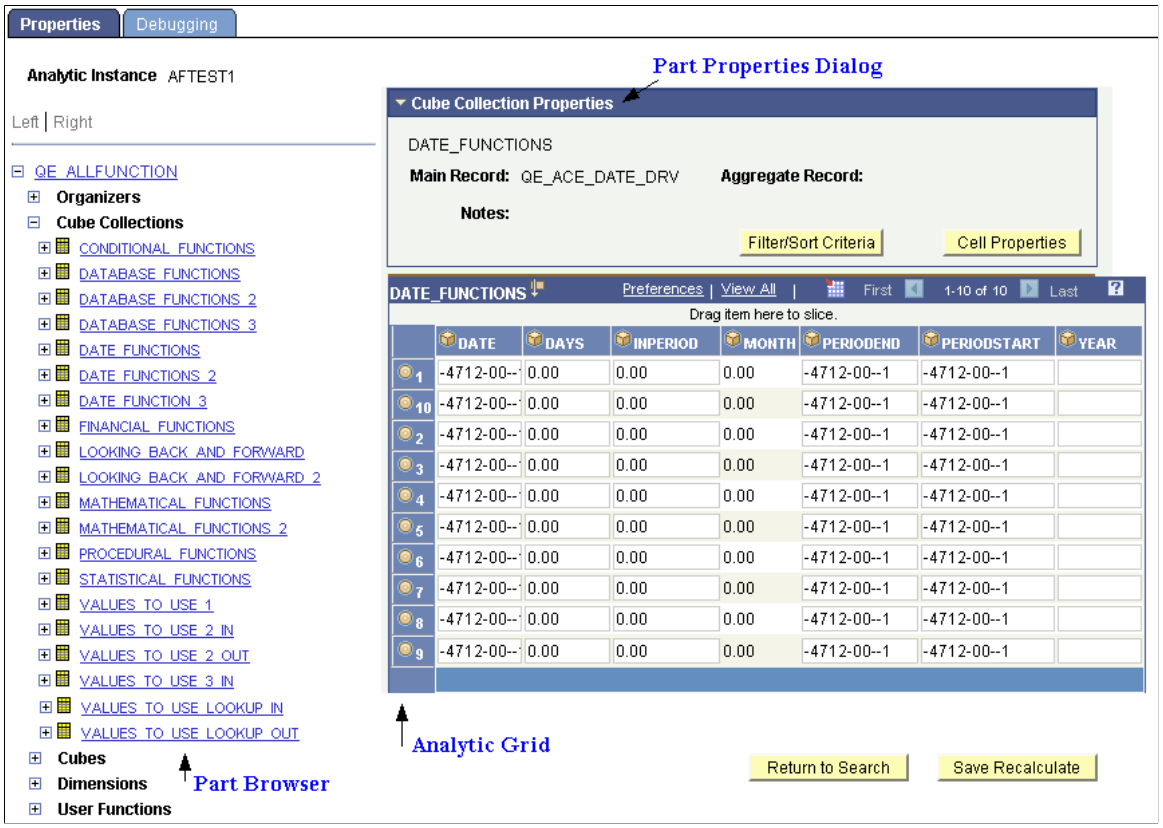
If you have already created an analytic instance and do not need to change any of its settings, you can use the Analytic Model Viewer to search for, load, and open the analytic instance. (If the instance is not loaded, clicking its name both loads and opens it; if it is loaded, clicking its name opens the instance.) When you open an analytic instance, the Analytic Model Viewer opens to a Properties tab, which displays the parts, properties, and data of the current analytic instance.

See Creating, Deleting, and Copying Analytic Instances, Loading and Unloading Analytic Instances.

Navigation:

PeopleTools > Utilities > Administration > Analytic Model Viewer

This example illustrates the fields and controls on the Analytic Model Viewer - Properties tab. Definitions for the fields and controls appear following the example.



Term	*Definition*
Part browser	Contains hierarchies that you use to view and debug the parts of the current analytic instance. This part browser is similar to the one in PeopleSoft Application Designer. For example, if you drill into a cube collection it expands to display the specific cube collections in the current analytic instance. You can then expand each cube collection further into data cubes and dimensions. When you click the name of any part, its associated properties appear on the right side of the page, in the Part Property dialog. See PeopleSoft Application Designer Window Components for Creating Analytic Model Definitions.
Part Properties dialog	Displays the properties of the selected part of the current analytic instance.
Analytic grid	Displays the data for the part selected in the Part browser. For example, if you select the Product Sales data cube, an analytic grid displaying Product Sales data appears. You can update the data in the analytic grid and recalculate the analytic instance. This analytic grid looks and feels like the analytic grid that you add to PeopleSoft pages to display data from the analytic server. You can use it to view, edit, and drag and drop data from an analytic model. See Understanding Analytic Grid Design, “Working with Analytic Grids” (Applications User’s Guide).
Debugging tab	Contains options that enable you to audit the selected data cube, or from one to three data cubes from the selected cube collection. You can use this feature to view All Causes, All Effects, All Inputs, Circular System, Direct Causes, and Direct Effects. Like the Properties tab, the Debugging tab includes a Part browser, a Part Property dialog, and an analytic grid. In addition, it displays an audit grid that displays the audited data when you click the **View** button.

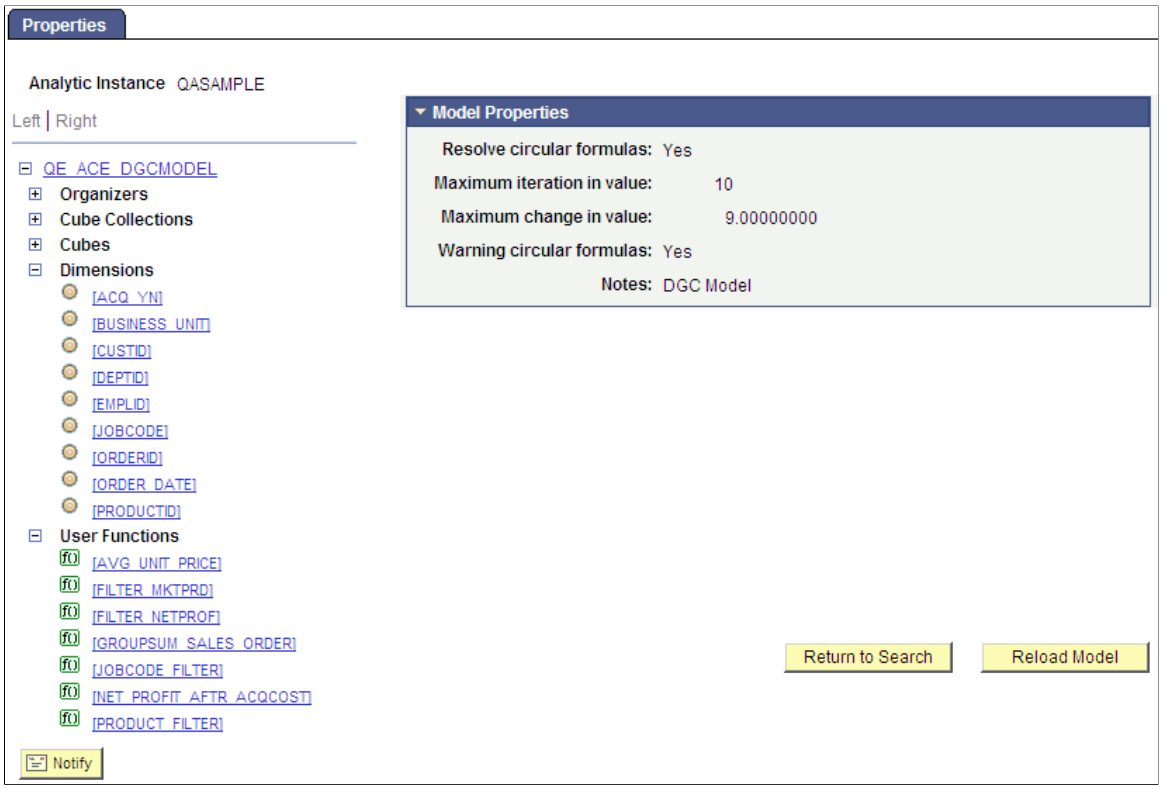
Viewing Analytic Models

Use the Analytic Model Viewer - Properties page (PTACEMDLVWR) to view the properties and data of the selected part through the PeopleSoft Pure Internet Architecture.

Navigation:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select the name of an analytic instance.

This example illustrates the fields and controls on the Analytic Model Viewer - Properties page. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Resolve circular formulas	Indicates whether circular formulas will be resolved. See Working with Circular Formulas and Circular Systems.
Maximum iteration in value	Indicates maximum iteration in value.
Maximum change in value	Indicates maximum change in value.
Warning circular formulas	Specifies whether the model contains circular formulas.
Notes	Lists notes related to this analytic model that the developer entered when creating the model. See Entering Notes for an Analytic Model Definition's Parts.
Reload Model	Reloads the analytic model. PeopleSoft recommends that you reload the analytic model after you update the analytic model definition so that you can view the resulting changes.

Viewing and Debugging Cube Collection Properties

This topic discusses how to:

- View cube collections and cube collection properties.
- Debug cube collections.
- Sort and filter cube collections.

See Purpose of Analytic Type Definitions, Relationship of Record Attributes to Data Caching Behavior, Synchronization Order.

Viewing Cube Collections and Cube Collection Properties

To view cube collections and cube collection properties:

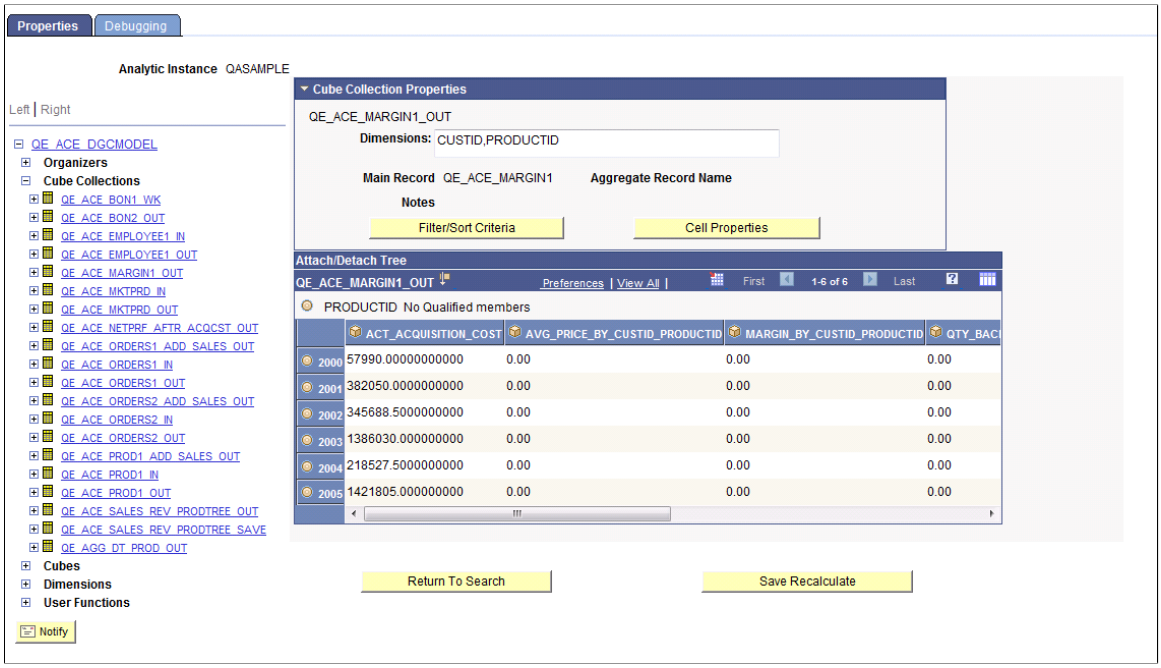
1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a cube collection whose properties you want to view.

   A Cube Collection Properties panel appears showing the properties of the selected cube collection.

   Note: The analytic grid underneath the Cube Collection Properties panel displays the cube collection data itself. You can drag and drop data cubes and dimensions within this grid; or view selected slices of your data by choosing from the slice bar.

See Understanding Analytic Grid Design, Creating a New Analytic Model Definition.

This is an example of the Cube Collection Properties panel within the Analytic Model Viewer - Properties page.



Field or Control	*Description*
Dimensions	Lists all of the dimensions that are attached to the data cubes within the cube collection.
Main Record	Lists the main record to which the cube collection is mapped.
Aggregate Record	Lists the aggregate record that stores the cube collection's aggregate data, if applicable.
Notes	Lists notes related to this cube collection that the developer entered when creating the model. See Entering Notes for an Analytic Model Definition's Parts.
Filter/Sort Criteria	Click to displays a secondary window enabling you to filter and/or sort a selected dimension within the cube collection.
Cell Properties	Click to displays a secondary window from which you can view the properties of a specified cell. You can view cell properties for data cubes and cube collections. See Viewing Cell Properties.
Save Recalculate	Click to recalculates the results of your analytic instance, if you modified it.

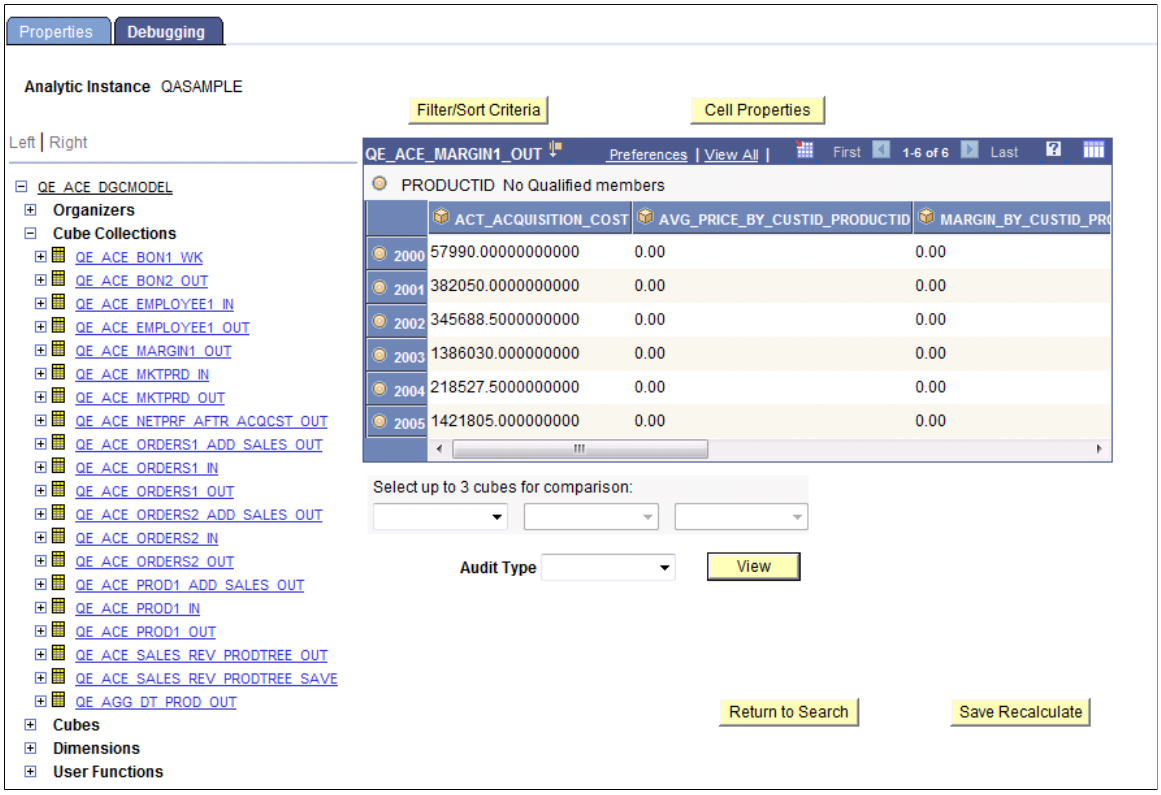
Debugging Cube Collections

Use the Analytic Model Viewer - Debugging page (PTACEMDLVWRDBG) to perform debugging tasks related to the selected data cube or cube collection.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Analytic Model Viewer** > **Debugging tab**

This example illustrates the fields and controls on the Analytic Model Viewer - Debugging page, Cube Collections. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Select up to 3 cubes for comparison	Select from one to three cubes in the cube collection to audit.
Audit Type	Select from one of the audit types: all causes, all effects, all inputs, circular system, direct causes, direct effects.
View	Click to displays the results of the audit in an audit grid below the analytic grid.

To debug a cube collection:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a cube collection whose properties you want to view.
3. Select the Debugging tab.

   The part browser, filtering and sorting features, and main grid on the Debugging tab work like those features on the Properties tab. However, this tab also enables you to audit the data.

See Auditing Data Cubes at Design Time.

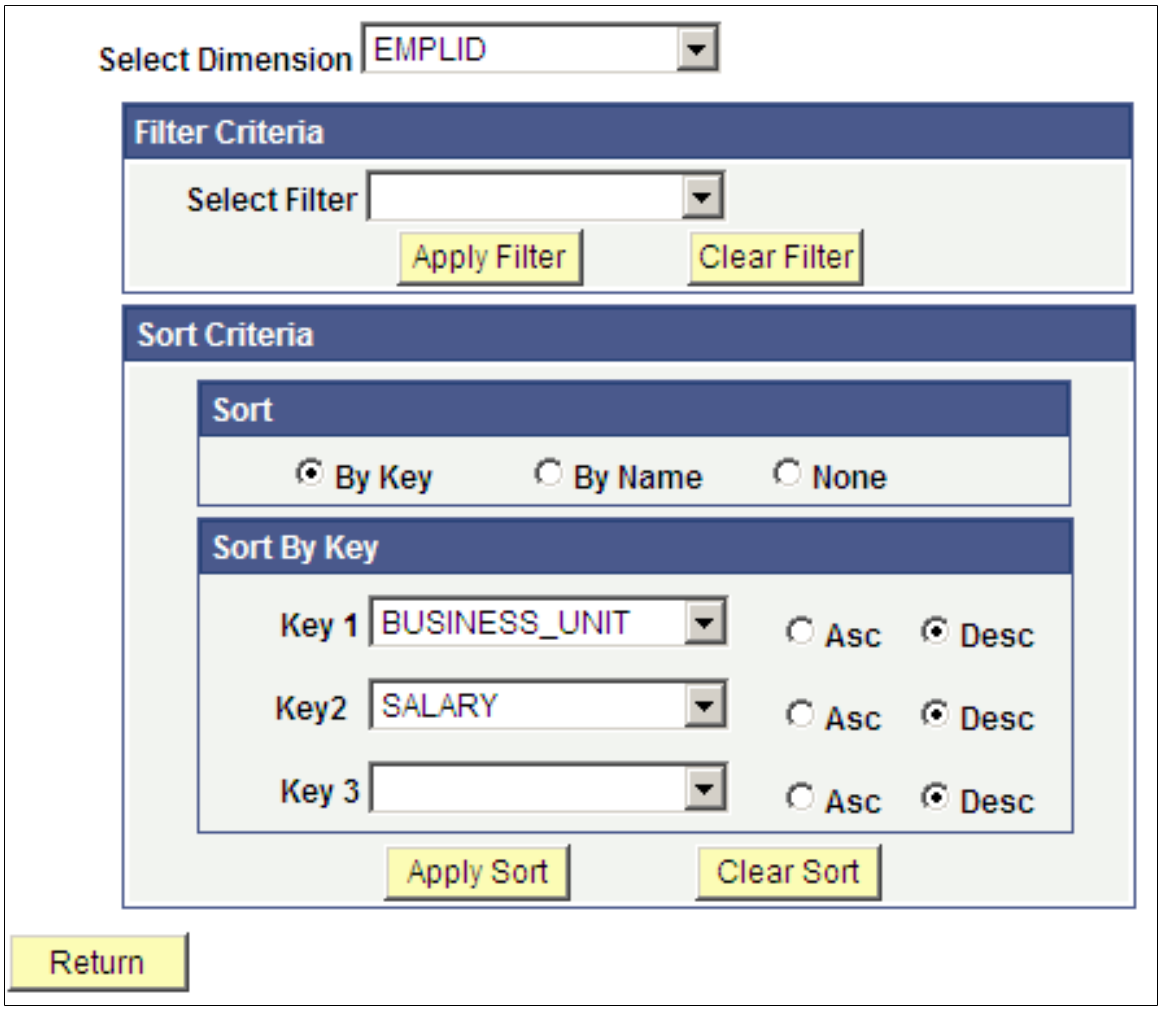
Sorting and Filtering Cube Collections

Use the Filter/Sort Criteria page (PTACEDIMPROP_SEC) to filter and sort the contents of the cube collection based on a specified dimension.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Analytic Model Viewer** > **Filter/Sort Criteria**

This example illustrates the fields and controls on the Analytic Model Viewer - Filter/Sort Criteria page. Definitions for the fields and controls appear following the example.



Note: Members are only filtered when the filter condition is met. In addition, if an aggregate member is filtered, all its children are also filtered.

Field or Control	*Description*
Select Dimension	Select the dimension upon which you want to base the filter or sort.

Field or Control	*Description*
Select Filter	Select the filter.
Apply Filter	Click to apply the selected filter.
Clear Filter	Click to clear the selected filter.
Sort	Select whether to sort by key, by name, or by neither. When you elect to sort by key, you can choose from one to three keys upon which to sort, and can choose to sort each one of those keys either ascending or descending. When you sort by name, you can choose to sort either ascending or descending.
Apply Sort	Click to apply the selected sort.
Clear Sort	Click to clear the selected sort.
Return	Click to go back to the main page.

To sort and filter cube collections:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a cube collection whose properties you want to view.
3. Click the **Filter/Sort Criteria** button.

Viewing and Debugging Data Cube Properties

This topic discusses how to:

- View data cubes and data cube properties.
- Debug data cubes.

See Understanding Data Cubes, Creating Calculation Data Cubes, Defining Data Cube Properties.

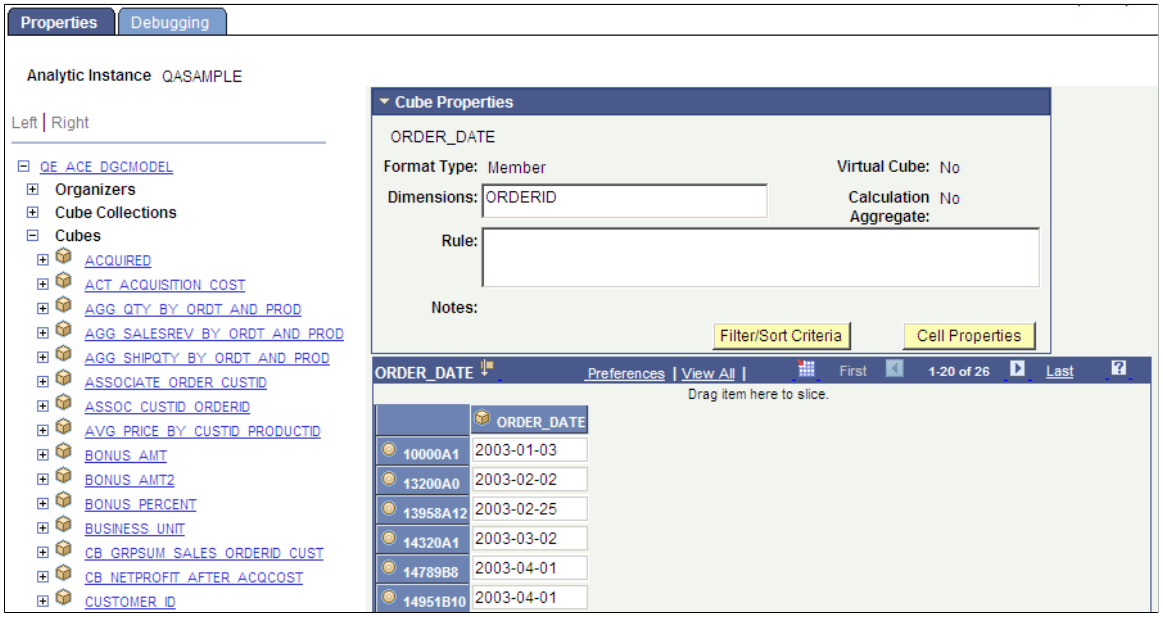
Viewing Data Cubes and Data Cube Properties

To view data cubes and data cube properties:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a data cube whose properties you want to view.

The analytic grid underneath the Cube Properties panel appears showing the data cube's values. As with cube collections, you can use drag and drop to manipulate the analytic grid. You can also view slices of your data by choosing from the drop-down lists of dimensions.

This example illustrates the fields and controls on the Analytic Model Viewer - Properties page, Cube Properties panel. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Format Type	Displays the format type of the data cube—such as Number, Char, Date, and so on. These format types are attributes of the cube and are defined within PeopleSoft Application Designer. See Understanding the Relationship Between Field Definition Attributes and Data Cube Formats.
Virtual Cube	Indicates whether the selected data cube is a virtual cube. See Virtual Data Cubes.
Dimensions	Lists the dimensions that are attached to the selected data cube.
Calculation Aggregate	Indicates whether the analytic calculation engine calculates aggregates for the data cube.
Rule	Displays any rules that the analytic calculation engine uses to calculate the current data cube.

Field or Control	*Description*
Notes	Lists notes related to this data cube that the developer entered when creating the model. See Entering Notes for an Analytic Model Definition's Parts.
Cell Properties	Click to displays a secondary window from which you can view the properties of a specified cell. You can view cell properties for data cubes and cube collections. See Viewing Cell Properties.
Save Recalculate	Click to recalculates the results of your analytic instance if you modified it.

Debugging Data Cubes

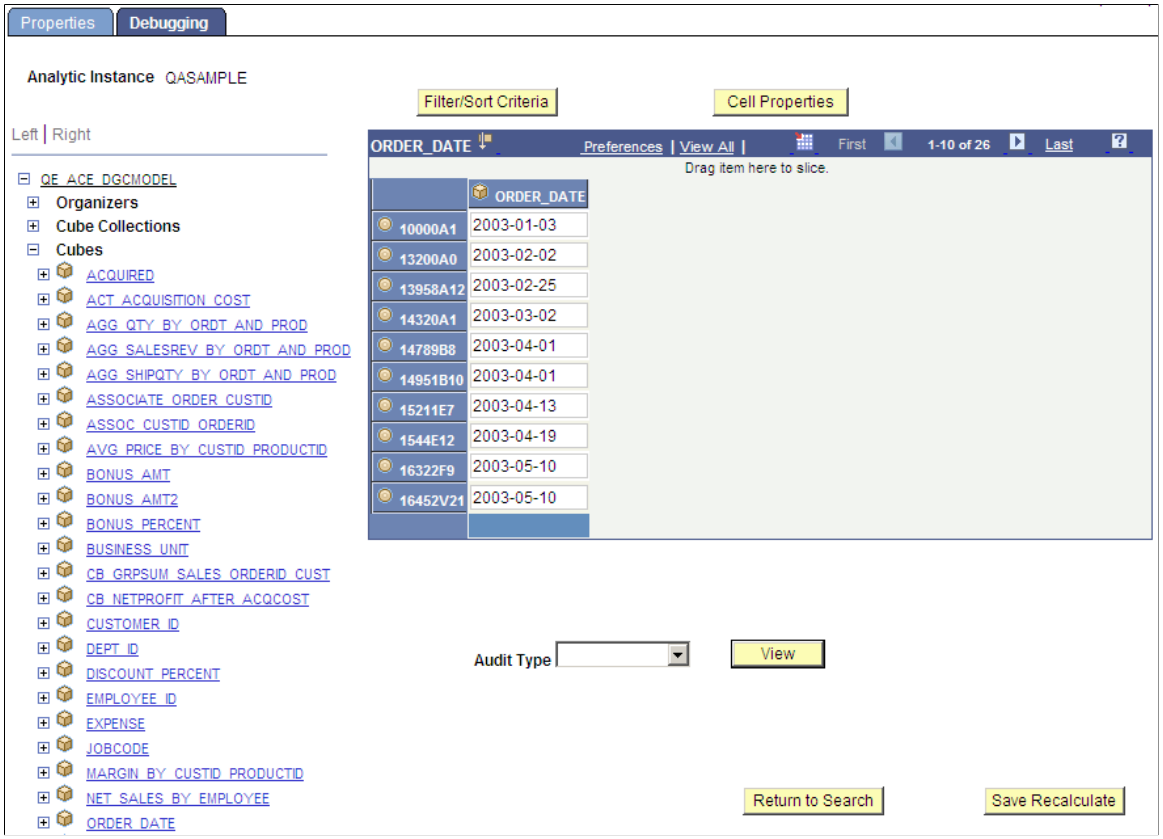
To debug a data cube:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a data cube whose properties you want to view.
3. Select the Debugging tab.

Note: You can also audit data cubes at design time.

See Auditing Data Cubes at Design Time.

This example illustrates the fields and controls on the Analytic Model Viewer - Debugging page, Data Cube panel. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Cell Properties	Click to displays a secondary window from which you can view the properties of a specified cell. You can view cell properties for data cubes and cube collections.
Audit Type	Select from one of the audit types: all causes, all effects, all inputs, circular system, direct causes, direct effects.
View	Displays the results of the audit in an audit grid below the analytic grid.

See Auditing Data Cubes at Design Time.

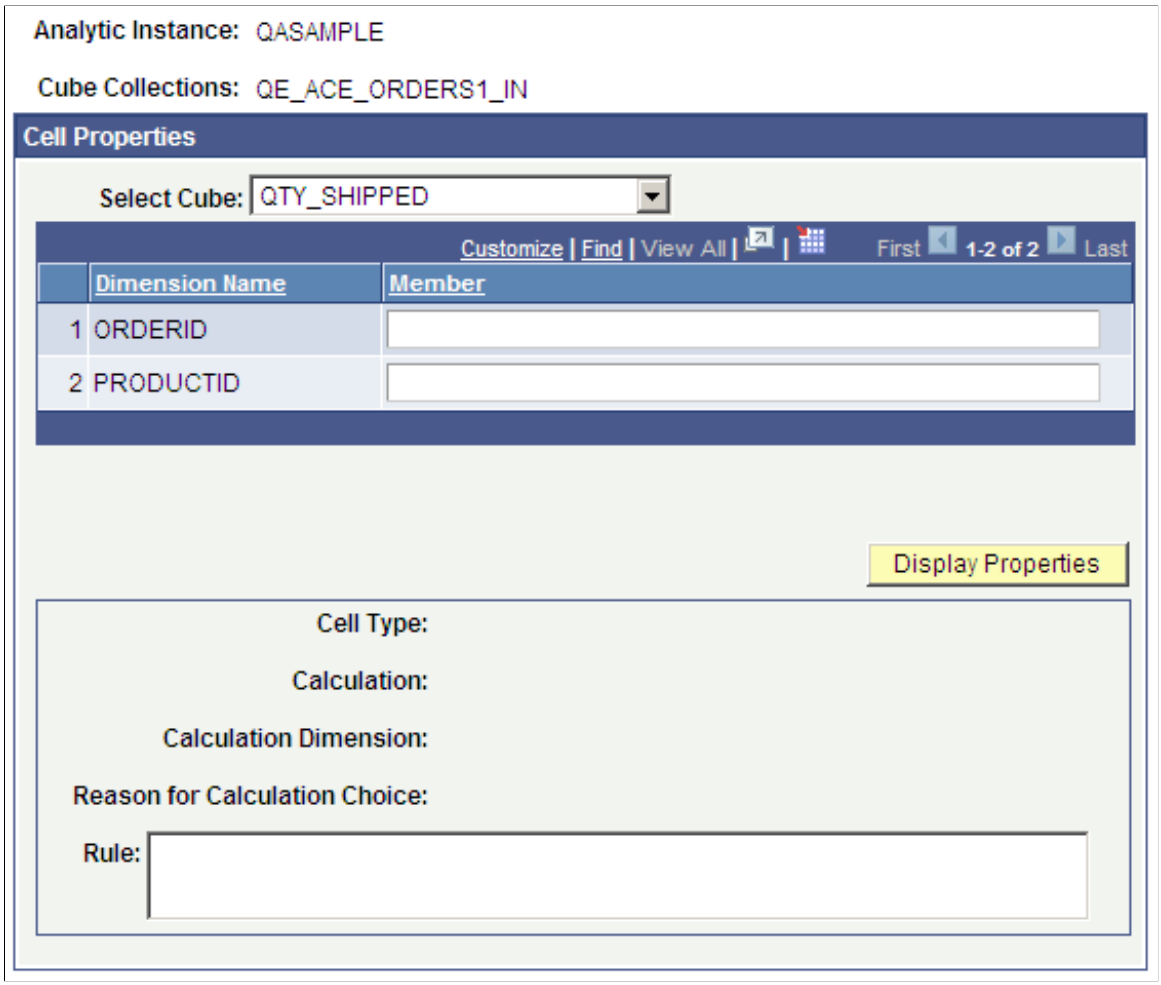
Viewing Cell Properties

Use the Cell Properties page (PTACECELLPROP_SEC) to view cell properties for designated member of the selected dimension.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Analytic Model Viewer** > **Cell Properties**

This example illustrates the fields and controls on the Analytic Model Viewer - Cell Properties page. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Select Cube	Choose which data cube in the selected cube collection you want to view. **Note:** The **Select Cube** option is enabled only if you activated the Analytic Model Viewer - Cell Properties page while viewing a cube collection.
Dimension Name	Displays the names of dimensions attached to the selected cube.
Member	Enter the member in the selected dimension whose properties you want to view.
Display Properties	Click to displays the properties—including cell type, calculation, calculation dimensions, reason for calculation choice, and rule—of the selected cell.

Field or Control	*Description*
Return	Click to go back to the main page.

To view cell properties:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a data cube whose properties you want to view.
3. Select the Debugging tab.
4. Click the **Cell Properties** button.

Viewing Dimension Properties

This topic discusses how to view dimension properties.

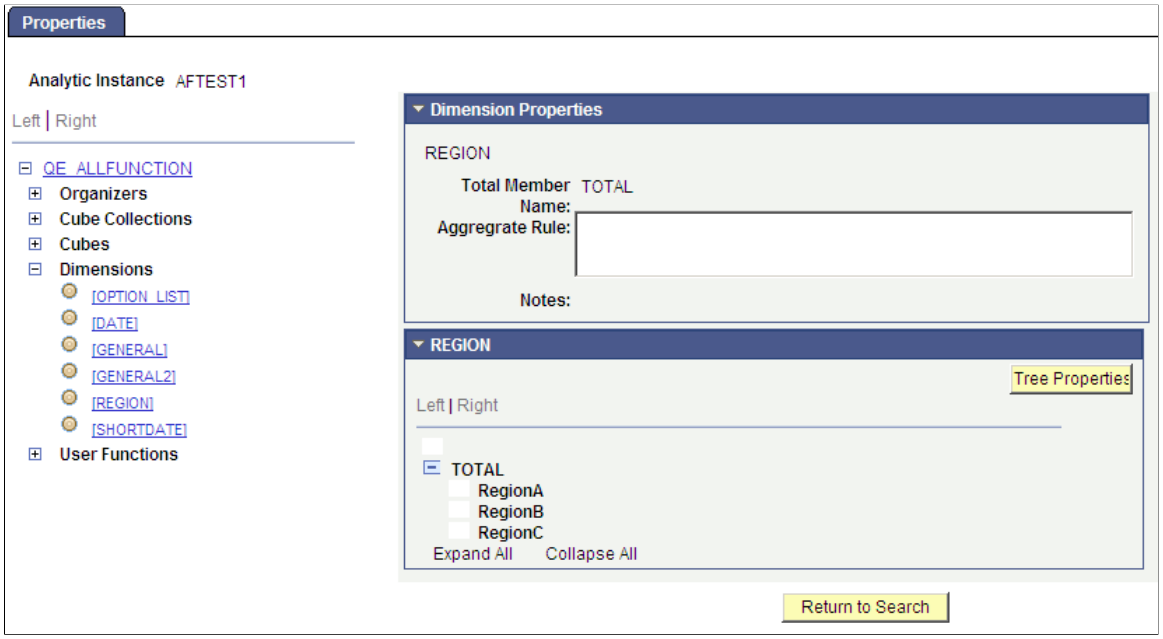
See Understanding Dimensions, Creating a New Dimension.

Use the Tree Properties page (PTACETREE_SEC) to view the properties of the tree attached to the selected dimension.

Navigation:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer**
2. To view a dimension with a tree attached, select Tree Properties.

This example illustrates the fields and controls on the Analytic Model Viewer - Properties page, Dimension Properties panel. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Total Member Name	Displays a different value depending on a fairly complex set of factors. See Defining Dimension Properties.
Aggregate Rule	Displays the user function that calculates the dimension's aggregate fields, if applicable. See Defining Dimension Properties, Working with Overrides.
Notes	Lists notes related to this dimension that the developer entered when creating the model. See Entering Notes for an Analytic Model Definition's Parts.
Dimension Members	Underneath the Dimension Properties panel are the dimension members. In many cases, you see a simple list of dimension members. In some cases, however, a hierarchy has been created for the dimension and you see a tree-like structure. In this case, you open and close each branch and leaf, and see each member of this hierarchy.
Tree Properties	Click to display a secondary page that displays additional properties of the selected tree, including: • Tree name • Node name • Start level • Discard level • SetID • Record name • Effective date

To view dimension properties:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a dimension whose properties you want to view.

   After you select a dimension in the part browser, a Dimension Properties panel appears showing the properties of the selected dimension.

Viewing User Function Properties

This topic discusses how to view user functions.

See Understanding Rules, Formulas, and User Functions, Defining and Editing User Functions.

To view user function properties:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select a user function whose properties you want to view.

   The Analytic Model Viewer - Properties page, User Function Properties panel appears.

This example illustrates the fields and controls on the Analytic Model Viewer - Properties page, User Function Properties panel. Definitions for the fields and controls appear following the example.

Field or Control	*Description*
Rule	Displays the user function's rule.
Notes	Lists notes related to this user function that the developer entered when creating the model. See Entering Notes for an Analytic Model Definition's Parts.

Viewing Organizer Properties

This topic discusses how to view organizer properties.

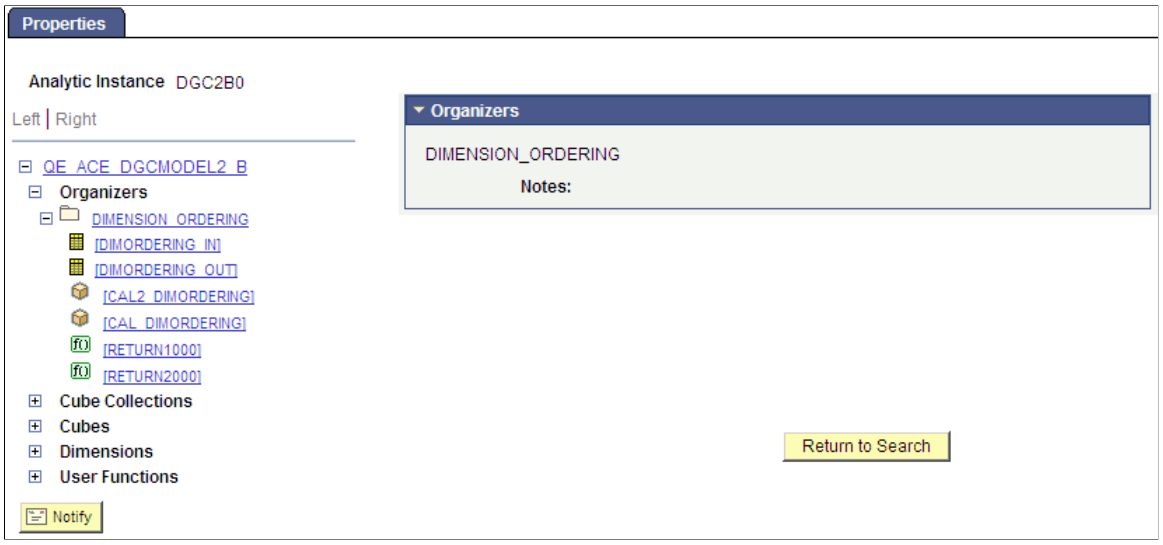
See Organizers, Creating Organizers.

To view organizer properties:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Analytic Model Viewer.**
2. Select an organizer whose properties you want to view.

The Analytic Model Viewer - Properties page, Organizer Properties panel appears.

This example illustrates the fields and controls on the Analytic Model Viewer - Properties page, Organizer Properties panel.



Using Analytic Model Viewer Alongside PeopleSoft Application Designer

It can be profitable to use the Analytic Model Viewer side by side with PeopleSoft Application Designer. This approach enables you to update the analytic model within PeopleSoft Application Designer and then quickly see the results of those updates by reloading the analytic instance within the Analytic Model Viewer. This approach enables you to change both the data and metadata for your model at the same time.

To use the Analytic Model Viewer alongside PeopleSoft Application Designer:

1. Create an analytic model.

   See Understanding the Analytic Model Definition Creation Process, Creating a New Analytic Model Definition, Opening an Analytic Model Definition.

2. Specify what analytic model works with what analytic type definition.

   See “Defining an Analytic Type” (Optimization Framework).

3. Access the Create Analytic Instance page, and create an analytic instance based upon the analytic type definition.

   See Creating, Deleting, and Copying Analytic Instances.

4. Access the Analytic Model Viewer, and open the analytic instance you created.

   See Viewing Analytic Model Properties.

5. From within PeopleSoft Application Designer, modify the analytic model.

6. From within the Analytic Model Viewer, click the **Reload Model** button.

Using the Application Log Fence

In addition to the model viewer, you can also use the application log fence settings to cause error messages created on the analytic server to be written to the analytic server log file.

If you set the application log fence to 3 or above, all the detailed messages created on the analytic server to be sent back to the application server are also logged in the analytic server log file.

In addition, if you set the application log fence to 4 or above, all tracing information is logged to the analytic server log file.

Related Links

“Using Application Logging” (PeopleCode Developer’s Guide)

Chapter 15

Capturing Analytic Instances

Understanding the Analytic Instance Capture Utility

When customers report a problem in an application that uses Analytic Calculation Engine or PeopleSoft Optimization Framework, they often need to send the relevant data and metadata to PeopleSoft engineers who can then review the problem. Using the Analytic Instance Capture Utility, customers can package the data and metadata in a form that they can send to PeopleSoft for analysis and debugging. PeopleSoft developers then employ the Analytic Instance Capture Utility to unpackage (import) the data for analysis.

The Analytic Instance Capture Utility works across platforms. For example, there should be no problem if a customer exports data from an Oracle database and then PeopleSoft user support imports it into Microsoft SQL Server. The same is true if the data is exported, for instance, from a UNIX platform and imported into Windows.

Note: The machines being used to import and export data must be on identical versions of PeopleTools.

The Analytic Instance Capture Utility is not intended to handle major problems, such as crashes. Instead, it focuses on data problems—for example, when customers discover incorrectly calculated application data and want PeopleSoft developers to help determine the source of these calculations errors.

Note: Before using the Analytic Instance Capture Utility to communicate with PeopleSoft support, you should attempt to diagnose the problem by using the Analytic Model Viewer.

See Understanding the Analytic Model Viewer, Viewing Analytic Model Properties.

Capturing Analytic Instance Data

You need to capture the relevant data and metadata before sending it to PeopleSoft support for diagnosis. You can capture the data by loading an analytic instance and then exporting it with the Analytic Instance Capture Utility.

By default, the Analytic Instance Capture Utility exports the analytic instance to a directory that it creates entitled *<PS_HOME>\appserv\<domain>\LOGS\<analytic instance><timestamp>*. For example, if you export an analytic instance named ACEINST, the resulting export directory is named something like *<PS_HOME>\appserv\<domain>\LOGS\ACEINST_20041113_015912.* A valid export directory will by default include the following files. If not, the export was not successful:

- The utility registration file items.reg.
- One or more data cache files named data_1.bin, data_2.bin, and so forth.
- Two .txt files, exportResults.txt and importDirections.txt.

The exportResults.txt file contains explicit details on how to export your metadata to a project. The importDirections.txt file contains details on how to import this particular analytic instance.

See Loading and Unloading Analytic Instances.

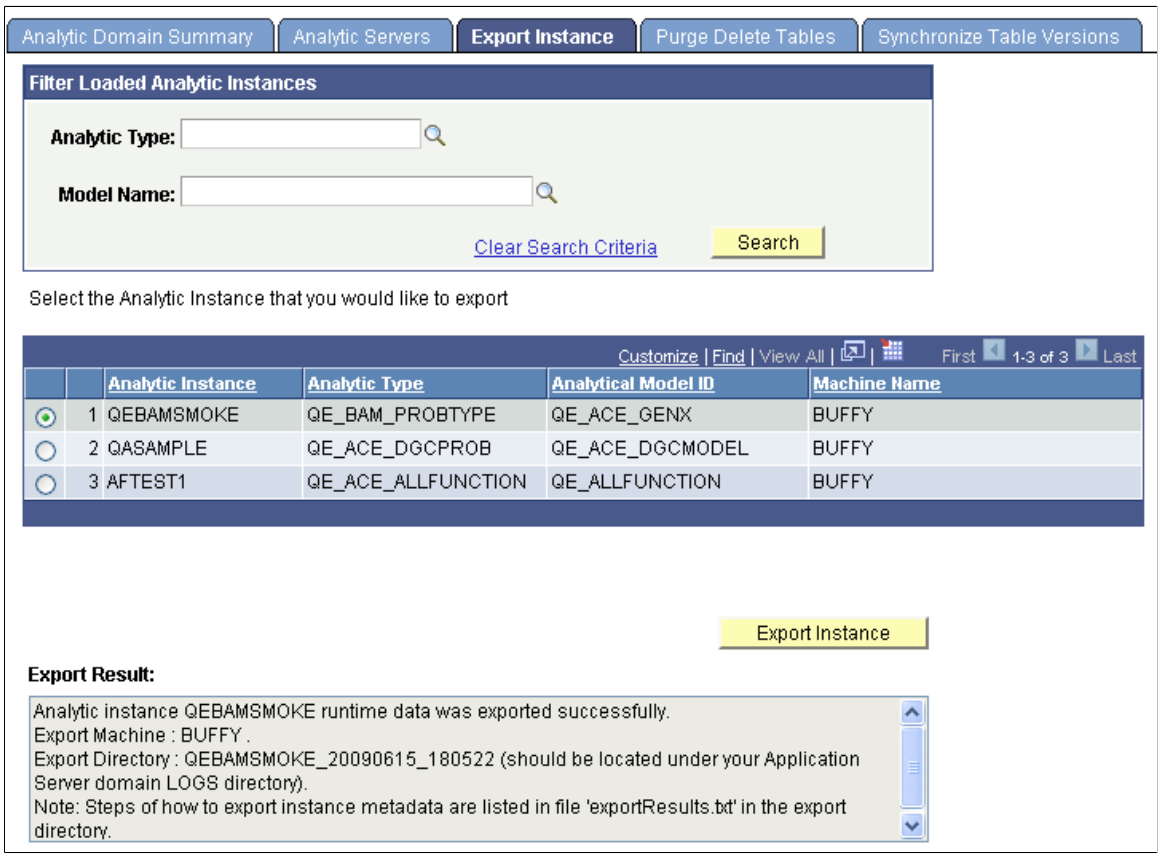
Exporting Analytic Instances

Use the Export Instance page (PTATEXPORT) to export an analytic instance.

Navigation:

PeopleTools > Utilities > Administration > Analytic Server Administration > Export Instance tab

This example illustrates the fields and controls on the Export Instance page.



To export an analytic instance:

1. Load the analytic instance.

   See Loading and Unloading Analytic Instances.

2. In PeopleSoft Pure Internet Architecture, select **PeopleTools > Utilities > Administration > Analytic Server Administration.**

3. Select the Export Instance tab.

4. (Optional) Select whether to filter the loaded analytic instances, either by analytic type or by model name.
5. Click the **Search** button to display the designated loaded analytic instances.
6. Click the option button to the left of the analytic instance that you want to export.

   Although you can load multiple analytic instances, you can export only one at a time.
7. Click the **Export Instance** button.

   The **Export Result** text box displays the status of the export operation. This message lists:

   - The instance name and whether it was exported successfully.
   - The export machine.
   - The export directory.
   - A message about the exportResults.txt file.
8. Retrieve the exportResults.txt file for specific details on how to export this analytic instance.
9. In PeopleSoft Application Designer, create a project that has the same name as the export directory that was created during the export procedure.

   Remember, the export process creates a directory whose name consists of the analytic instance name followed by the date and timestamp. For example, if the analytic instance is named ACEINST, the directory might be named ACEINST_20041113_015912.

   See “Understanding Projects” (Application Designer Developer’s Guide), “Creating and Saving Projects” (Application Designer Developer’s Guide).
10. Select **Insert** > **Definitions into Project** and insert the items listed in the exportResults.txt file into the project.
11. Save the project.
12. Select **Tools** > **Copy Project** > **To File.**
13. Select <*PS_HOME*>*appserv*\<*domain*>*LOGS*\<*analytic instance*><*timestamp*> as the export directory and then click the **Copy** button.
14. Zip the contents of <*PS_HOME*>*appserv*\<*domain*>*LOGS*\<*analytic instance*><*timestamp*> and send it to PeopleSoft user support.

Importing Analytic Instance Data

After the customer packages the analytic instance and sends it to PeopleSoft user support, user support must import that data to diagnose the customer issue.

Importing Analytic Instances

Use the Create Analytic Instance page (PTACECRTINST) to create an analytic instance.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Maintain Analytic Instances**

Use the Analytic Instance Load/Unload page (PTACEMDLLOAD) to load or unload an analytic instance.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Load/Unload Analytic Instances**

To import an analytic instance:

1. Unzip the packaged analytic instance into the directory of your choice—for example, *c:\TEMP*.
2. Read the importDirections.txt file for explicit details about how to import this analytic instance.
3. In PeopleSoft Application Designer, select **Tools** > **Copy Project** > **From File.**
4. Search for the project named <*analytic instance*><*timestamp*>.

   This project, which is the result of the export functionality, should be located in the *c:\TEMP\<analytic instance>* directory. For example, if the analytic instance is named ACEINST, the directory would be named something like *c:\TEMP\ACEINST_20041113_015912*.
5. Access the Create Analytic Instance page and create an analytic instance named <analytic instance>—for example, ACEINST.

   See Creating, Deleting, and Copying Analytic Instances.
6. Access the Analytic Instance Load/Unload page and perform these steps:

   See Loading and Unloading Analytic Instances.

   a. Select the name of the directory you just created.

   b. Select the **Import From File** check box.

   c. Enter the import directory name in the **File Directory** text box.

   d. Click the **Load Analytic Instance** button.

Chapter 16

Converting BAM 8.8 Models to Analytic Models

Understanding the Conversion Process

PeopleSoft Business Analysis Modeler (BAM) is a standalone application that enables developers to create multidimensional models for the purposes of reporting and analysis. BAM models are not integrated into the PeopleTools framework. Integrating BAM models with PeopleSoft applications takes many steps and a significant amount of time.

Analytic Calculation Engine analytic models are integrated into the PeopleTools framework and include much of the same functionality and many of the same parts as BAM models. You may want to convert existing BAM 8.8 models into Analytic Calculation Engine analytic models to reduce the extra steps and time needed to integrate these models with PeopleSoft applications. You use the PTAEACECONV Application Engine program for this purpose.

Because PTAEACECONV does not convert all BAM 8.8 model parts, you must manually complete the analytic model after conversion.

Note: The PTAEACECONV Application Engine program does not convert application data. Application developers are responsible for converting application data.

Converting BAM 8.8 models into analytic models involves these steps:

1. Analyze the BAM 8.8 model.

   If the existing model contains rules that use the TEXT2MBR function, you must hard-code the function's second argument (Text) if the use of the function meets both of these conditions:

   - The TEXT2MBR function's second argument (Text) is not hard-coded.
   - The second argument matches the name of any dimension or data cube in the model.

     Note: If the second argument matches the name of a dimension member reference—for example, [COUNTRY:Belgium]—you do not need to hard-code the second argument of this instance of the TEXT2MBR function.

     ---

2. Export the BAM 8.8 model.

   See Exporting BAM 8.8 Models.

3. Run the PTAEACECONV Application Engine program.

   See Running the PTAEACECONV Application Engine Program.

4. Examine the Application Engine log file.

   See Examining the PTAEACECONV Log File.

5. Map the new analytic model to main and aggregate records that hold the application and aggregation data.

   See Mapping a Cube Collection to Main and Aggregate Records.

6. Map data cubes and dimensions to fields in the main and aggregate records.

   See Mapping Data Cubes and Dimensions to Fields.

7. Create PeopleSoft Pure Internet Architecture pages with analytic grids.

   See Understanding Analytic Grid Design.

8. View the new analytic model in PeopleSoft Pure Internet Architecture.

   See Understanding the Analytic Model Viewer.

The PTAEACECONV Application Engine program converts most parts and circular formula options into analytic models.

BAM 8.8 Parts That Can Be Converted

These BAM 8.8 parts can be converted into analytic models:

- Data cubes
- Dimensions
- User functions
- Expression modifiers
- Table views
- Import maps
- Organizers

BAM 8.8 Parts That Cannot Be Converted

These BAM 8.8 parts cannot be converted into analytic models:

- Prefix modifiers
- Styles
- Option lists

Note: References to option lists within the code of data cube rules; user functions and expression modifiers, however, are converted.

See Understanding Part Conversion Details.

- Timelines
- Roles
- Chart views

BAM 8.8 Circular Formula Options That Can Be Converted

These BAM 8.8 circular formula options can be converted into analytic models:

- Resolve circular formulas through iteration.
- Maximum number of iterations.
- Maximum change in values.
- Warn about circular formulas every time a circular formula is defined.

BAM 8.8 Circular Formula Options That Cannot Be Converted

The *Only if iteration is not enabled* BAM 8.8 circular formula option cannot be converted into an analytic model.

Related Links

Understanding Part Conversion Details

Understanding Part Conversion Details

This topic provides more detailed information about the conversion of BAM 8.8 parts.

Part Names Conversion Method

PTAEACECONV Application Engine program converts the names for each part that will be included in the analytic model.

The following table describes BAM 8.8 part name attributes and the changes that PTAEACECONV makes to these attributes.

BAM 8.8 Part Name Attribute	***Attribute Change Upon Conversion***
Lower case alphanumeric characters	All lower case alphanumeric characters are converted to upper case alphanumeric characters. For example: *Products* converts to *PRODUCTS*.
White spaces	All white spaces are converted to underscores. For example: *Actual Sales* converts to *ACTUAL_SALES*.

BAM 8.8 Part Name Attribute	*Attribute Change Upon Conversion*
Hyphens	All hyphens are converted to underscore characters. For example: *Gross-Margin* converts to *GROSS_MARGIN*.
Non alphanumeric characters	Non alphanumeric characters are removed from the part name. For example: *Cost$ of delivery* converts to *COST_OF_DELIVERY*.
Part names with more than 27 characters	Characters exceeding the 27 character limit are truncated. For example: *Moving STD by Country and Group* converts to *MOVING_STD_BY_COUNTRY_AND_G*
Duplicate part names	Numeric values are appended to duplicate part names. For example, if the BAM 8.8 model contains the *Actual Sales* and *Actual_Sales* part names, PTAEACECONV creates two new part names: *ACTUAL_SALES* and *ACTUAL_SALES1*.

Note: Converted names of expression modifiers include the prefix EXP_.
Additionally, for each part that is converted, the original part name is converted into the new part's description.
Filter user functions that are referenced by dimensions that exist in table views are converted to user functions. The user function names include the prefix DR_ plus the converted dimension name.

Code in Data Cube Rules, User Functions, and Expression Modifiers

PTAEACECONV uses the following order of execution when converting code in data cube rules, user functions, and expression modifiers:

1. Replace all references to part names with resolved part names.

   During conversion, PTAEACECONV searches through the code in data cube rules, user functions, and expression modifiers for all part names and replaces these part names with new, converted part names. For example, the EmployeeNetMonthlyIncome user function contains the following code:

   ```
   Monthly Salary - Monthly Deductions
   ```

   PTAEACECONV changes the user function's code to:

   ```
   MONTHLY_SALARY - MONTHLY_DEDUCTION
   ```

2. Replace all references to option lists with a literal string.

Because analytic models do not support option lists, PTAEACECONV converts references to option lists within the code of data cube rules, user functions, and expression modifiers. For example, the RevenueMethod option list exists in the following user function code:

```
&RevenueMethod := GetRevenueMethod( );
CASE(
&RevenueMethod = \Revenue Method\Data Entry\ :
  Do_Something;
&RevenueMethod = \Revenue Method\Repeat Value\ :
  Do_Something_ELSE;
)
```

The PTAEACECONV Application Engine program converts the user function's code to:

```
&RevenueMethod := GetRevenueMethod( );
CASE(&RevenueMethod = "Data Entry":
  Do_Something;
    &RevenueMethod = "Repeat Value":
  Do_Something_ELSE;
)
```

3. Replace all references to original dimension names with converted dimension names.
4. Replace all references to original data cube names with converted data cube names.

   Note: If a dimension name and data cube name share the same name in the original model and one or both names contain more than 30 characters, the dimension name retains the original part name in the converted analytic model. If the BAM model contains rules or user functions that reference data cubes that share the same names as dimensions, the converted rules and user functions reference the dimensions instead. Developers must resolve these issues. The PTAEACECONV conversion log file indicates all data cubes and dimensions that fall into this category.

5. Replace all references to the original user function names with converted user function names.

Data Cubes

The PTAEACECONV Application Engine program converts all data cubes and most data cube attributes.

Note: Data cube values are not converted. Application developers are responsible for converting data cube values.

The following data cube attributes are unaffected by the conversion:

- These data cube formats:
  - Text

    Note: The Width property is not converted.

  - Number

    Note: Digit and Decimal properties are not converted.

  - Member
  - Date

Note: The Dimension Name property is not converted.

- These virtual data cube properties:
  - Is virtual
  - Is not virtual
  - Note
  - Attachments to dimensions

The following data cube attributes are changed during conversion:

- These data cube formats:
  - General

    The General format is converted to the Text format.
  - Currency

    The Currency format is converted to the Number format.
  - Option List

    The Option List format is converted to the Text format.
  - Percent

    The Percent format is converted to the Number format.

    Note: Digit and Decimal properties are not converted.
  - Yes/No

    The Yes/No format is converted to Text format.
- Code in data cube rules.

  For more information, see the Expression Modifiers section below.
- Data cube names:
  - Original data cube names are converted into new data cube names using the part names conversion method.

    For more information, see the
  - Original data cube names are also converted into new data cube descriptions.

These data cube attributes are not converted:

- All methods for combining periods, including:
  - Summing Values
  - Averaging Values
  - Last in Period
  - Using Formula
  - Blank
- All methods for splitting periods, including:
  - Dividing Value
  - Interpolating
  - Repeating Value
  - Using Formula
  - Blank
- All methods for justification, including:
  - Default
  - Left
  - Center
  - Right
- Formatting function names.

Dimensions

PTAEACECONV converts all dimensions and most dimension attributes.

The conversion does not change the notes for dimensions.

These dimension attributes are converted but are changed during the conversion process:

- Dimension names:
  - Original dimension names are converted into new dimension names using the part names conversion method.

    For more information, see the Part Names Conversion Method section.
  - Original dimension names are also converted into new dimension descriptions.
- Total member names.

If a dimension contains a Total member, the name of the Total member is converted to an alias of the root node used in the analytic model.

These dimension attributes are not converted:

- Dimension members.
- Default Alias Function property.

User Functions

PTAEACECONV converts all user functions.

These user function attributes are unaffected by conversion:

- Rules that have been defined to use within user functions.
- References to user functions from other parts.

  For more information, see the Expression Modifiers section below.

These user function attributes are changed during the conversion process:

- User function names:
  - Original user function names are converted into new user function names using the part names conversion method.

    For more information, see the Part Names Conversion Method section.
  - Original user function names are also converted into new user function descriptions.
- Code used in user functions.

  For more information, see the Expression Modifiers section.

PTAEACECONV does not convert these user function categories:

- Calculation Function.
- Alias Function.
- Formatting Function.

Expression Modifiers

PTAEACECONV converts all expression modifiers. Converted expression modifiers exist as user functions in the analytic model.

PTAEACECONV does not affect rules that are defined for expression modifiers.

These expression modifier attributes are converted but are changed during the conversion process.

- Expression modifier names:

    - Original expression modifier names are converted into new expression modifier names using the part names conversion method.

      For more information, see the Part Names Conversion Method section.

    - Converted names of expression modifiers include the prefix EXP_.

    - Original expression modifier names are also converted into new expression modifier descriptions.

- Code used in expression modifiers.

  For more information, see the Expression Modifiers section.

PTAEACECONV does not convert references to the original expression modifiers.

Table Views

PTAEACECONV converts all table views into cube collections. Note that BAM 8.8 table views lack important information needed to complete cube collections, including:

- Main and aggregate records.
- Field mapping between dimensions and data cubes to fields in the main and aggregate records.

  See Mapping Data Cubes and Dimensions to Fields.

For this reason, you must provide this information in the converted analytic model.

These table view attributes are unaffected by conversion:

- All references to data cubes.
- All references to dimensions.
- Notes.

PTAEACECONV changes table view names in the following manner:

- Original table view names are converted into new cube collection names using the part names conversion method.

  For more information, see the Part Names Conversion Method section.

- Original table view names are also converted into new cube collection descriptions.

These table view attributes are not converted:

- All references to timelines
- All references to expression modifiers
- Prefix modifiers
- All table view-related properties, including:

    - Coordinates

    - Positions of dimensions in table views
    - Table header cells
    - Sections of table views

- Table data

Import Maps

PTAEACECONV converts all import maps into cube collections. Note that BAM 8.8 import maps lack important information needed to complete cube collections, including:

- Mapping between cube collections to main and aggregate records.
- Field mapping between dimensions and data cubes to fields in the main and aggregate records.

  See Mapping Data Cubes and Dimensions to Fields.

For this reason, it is necessary for application developers to provide this information in the converted analytic model.

These import map attributes are unaffected by conversion:

- All references to dimensions
- All references to data cubes
- Notes

PTAEACECONV converts import map names but changes the names in the following manner:

- Original import map names are converted into new cube collection names using the part names conversion method.

  For more information, see the Part Names Conversion Method section.

- Original import map names are also converted into new cube collection descriptions.

PTAEACECONV does not convert all table view-related properties, including:

- Coordinates.
- Positions of dimensions in import maps.

Organizers

PTAEACECONV converts all organizers.

These organizer attributes are unaffected by conversion:

- Hierarchies within organizers (for example, folders within folders).
- Notes.
- References to all parts except:

- Expression modifiers
- Prefix modifiers
- Styles

PTAEACECONV converts organizer names but changes them in the following manner:

- Original organizer names are converted into new organizer names using the part names conversion method.

  For more information, see the Part Names Conversion Method section.

- Original organizer names are also converted into new organizer descriptions.

PTAEACECONV does not convert organizer references to these parts:

- Expression modifiers
- Prefix modifiers
- Styles
- Chart views

Exporting BAM 8.8 Models

When you export a BAM 8.8 model, you create an XML file of the model. To export the BAM 8.8 model:

1. Launch PeopleSoft 9.1 Business Analysis Modeler.
2. Select **File** > **Open to open an analytic model.**

   The Open dialog box appears.
3. Select the model that you want to convert.
4. Click the **Open** button.

   The model appears in the Model Designer.
5. Select **File** > **XML** > **Export Schema.**

   The Export XML dialog box appears.
6. Select the location to which you want to export the model schema.
7. Enter a filename for the schema.
8. Click the **Save** button.

Running the PTAEACECONV Application Engine Program

This topic discusses how to run the PTAEACECONV Application Engine program.

Related Links

“Using the Command Line to Invoke Application Engine Programs” (Application Engine)

“Starting Programs with the Application Engine Process Request Page” (Application Engine)

Running PTAEACECONV from PeopleSoft Application Designer

Before running PTAEACECONV from PeopleSoft Application Designer, you must customize the program to find the location and file name of the correct XML file. In PeopleSoft Application Designer, open the PTAEACECONV Application Engine program definition and view the PeopleCode in Step01. Scroll down to the following PeopleCode:

```
If (&modelName = "") Then
   &modelName = "TEST";
End-If;

If (&xmlFilePath = "") Then
   &xmlFilePath = "C:\PeopleSoft\text.xml";
End-If;
```

Replace the *TEST* variable with the XML filename of the exported BAM 8.8 model.

Replace the *C:\PeopleSoft\text.xml* variable with the full path to the XML file of the exported BAM 8.8 model.

Note: The path must include the full name and extension of the XML file, for example: *C:\User \employment\employment.xml*.

When completing the run request, select to output a log to a file. Either use the default path*c:\temp \PTAEACECONV.log* or create your own path.

Related Links

“Using PeopleCode to Invoke Application Engine Programs” (Application Engine)

Running PTAEACECONV from a PeopleSoft Pure Internet Architecture Page

You use PeopleSoft Application Designer to create a PeopleSoft Pure Internet Architecture page that can run the PTAEACECONV Application Engine program. This page must contain the following PeopleCode event:

```
Local Record &staterec = CreateRecord(Record.PTACECONV_AET);
&staterec.ACEXMLFILEPATH.Value = <ACEXMLFILEPATH>;
&staterec.ACEMODELID.Value = <model_name>;
CallAppEngine("PTAEACECONV", &staterec);
```

Both the <*ACEXMLFILEPATH*> and <*model_name*> variables should be replaced by user input.

For example, you would enter *C:\User\employment\employment.xml* for the <*ACEXMLFILEPATH*> variable, and *Employment* for the <*model_name*> variable.

Examining the PTAEACECONV Log File

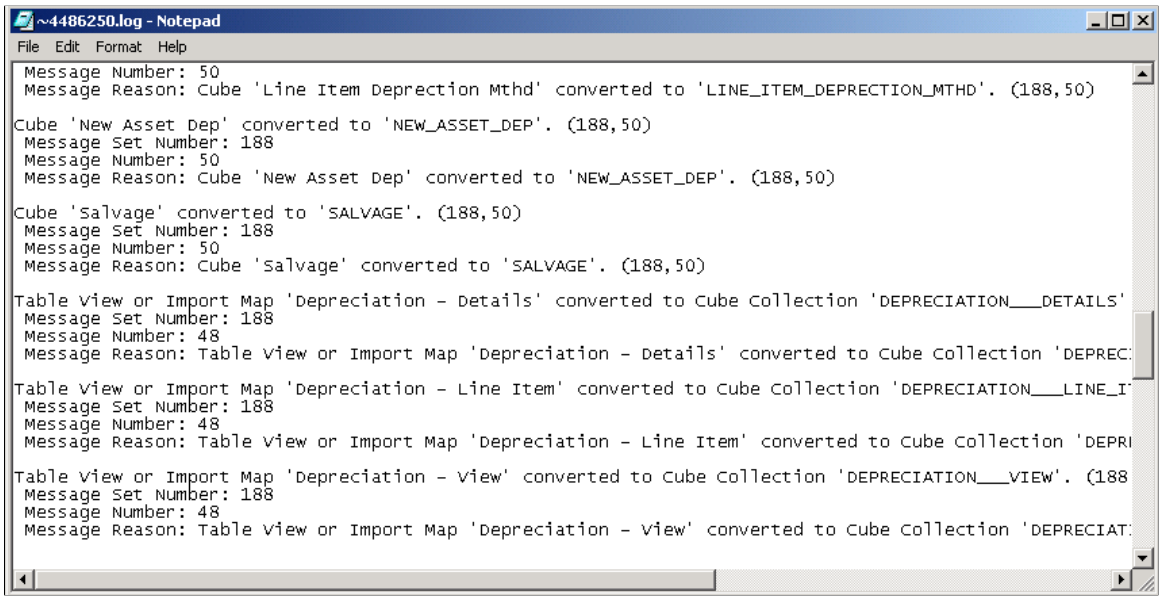
Use the log file to determine whether the BAM model successfully converted to an analytic model, or whether there are conversion errors that you must resolve.

If the BAM model converted successfully to an analytic model, the message *Application Engine program PTAEACECONV ended normally* appears at the bottom of the PTAEACECONV log file.

The PTAEACECONV log file contains detailed information about:

- All parts that were successfully converted.
- All parts that were not converted for either of these reasons:
  - Conversion failure.
  - Parts were not available in Analytic Calculation Engine.
- All part names that were changed using the part name conversion method.
- All user functions and rules that contained changed part names.
- All expression modifiers that were converted to user functions.
- All part name conflicts, such as shared names between dimensions and data cubes.

This example shows the PTAEACECONV log file.



Chapter 17

Managing Analytic Servers

Understanding the Analytic Server Framework

This topic provides an overview of Analytic Server Framework present in PeopleSoft.

Analytic Server Framework Overview

When a program doesn't “maintain state” or when the infrastructure of a system prevents a program from maintaining state, it's known as a *stateless* program or system. It can't take information about the last session into the next session, such as settings the user makes or conditions that arise during processing.

For example, the HTTP protocol is stateless. Additional schemes, such as cookies, are necessary to maintain state in the HTTP (web) environment.

PeopleTools is architected primarily around a stateless model of client/server connectivity. This model enables users' application sessions to be preserved even if servers are shut down or rebooted. All session state is maintained by the client and is transferred to the server with each request. As long as an application server is up and running, a user’s session remains active and functional, and any application server can perform requested transactions.

However, with some products, such as Analytic Calculation Engine or PeopleSoft Optimization Framework, running a calculation on a multi-dimensional model is likely to produce far more data than is reasonable to shuttle between a client and server to maintain a stateless connection. For performance reasons, the calculations are performed completely in memory. If these calculations were to be synchronized and stored in the database so that a stateless connection could be maintained, performance would suffer significantly.

The *analytic server framework* provided by PeopleSoft is a general server infrastructure designed to meet the needs of PeopleSoft products that process large amounts of data in memory. It provides a *stateful* model of client/server connectivity that these products require to be part of the PeopleTools system, by keeping track of configuration settings, transaction information, and other data for a session.

For example, client software could request that an analytic model or optimization model be recalculated in one transaction, then retrieve the results of the calculation on that model at a later time. A server process handles these requests, and maintains the model state and calculated data in memory between the requests. Additional transactions can then modify the model and perform recalculations on it without shuffling all of the data between the client and the server or dumping all the data to a database, thus preserving in-memory performance.

A large model might take a long time to load. In the event that a user’s session times out and is terminated, the loading and calculation of the model continues, and enables the user to return to the model at a later time in a new session.

The elements of the analytic server framework are:

- PSANALYTICSRV server.

  PSANALYTICSRV is a Tuxedo managed PeopleSoft application server process, like PSAPPSRV. It contains both the analytic calculation engine and the optimization engine. Multiple instances of PSANALYTICSRV can run in an application server domain. The current condition of each PSANALYTICSRV instance is tracked in system tables.

- Analytic server administration pages.

  The Analytic Domain Summary page provides current information about the application server domains with PSANALYTICSRV running that are attached to the current database.

  The Analytic Servers page enables you to inspect the individual analytic server instances within the running domains, with information about their analytic types and analytic instances, operations, and timeout intervals. You can also halt processes individually on this page.

- Analytic table administration pages.

  The Purge Delete Tables page displays the names of delete tables relevant to an analytic type or analytic instance, and enables you to clear the data from the tables.

  The Synchronize Table Versions page enables you to resynchronize versions of analytic type or analytic instance data and the PSOPTSYNC table that are out of synchronization after you use PeopleSoft Data Mover to move data from one database to another.

- Pages for creating, deleting, and copying analytic instances.

  The Create Analytic Instance, Delete Analytic Instance, and Copy Analytic Instance pages enable you to define and manage analytic instances that you can then load to inspect and debug your analytic models.

- The Analytic Instance Load/Unload page.

  The Analytic Instance Load/Unload page enables you to load analytic instances so you can view them within the Analytic Model Viewer, then unload the analytic instances that you no longer need.

- Various supporting enhancements in several PeopleTools products.

  These products include Analytic Calculation Engine, PeopleSoft Optimization Framework, PeopleSoft Performance Monitor, PeopleSoft Process Scheduler, PeopleSoft Application Engine, PSADMIN, and PeopleCode.

  Note: Information about the role that the analytic server framework plays in these products can be found in the documentation for each product.

Analytic Server Terms

The following terms are useful in understanding analytic server technology.

Term	*Definition*
Analytic type	A description of a data set to be loaded and the calculations to be performed on the data set in the analytic server framework. Multiple calculation engines such as the analytic calculation engine or the optimization engine can be associated with an analytic type.
Analytic instance	One instance of an analytic type. You can create multiple instances of the same analytic type.
Analytic server	The primary functional element of the analytic server framework, called PSANALYTICSRV. Each application server domain can include zero or more analytic servers.
Analytic server instance	One running instance of an analytic server. You can run multiple instances of PSANALYTICSRV for a given domain. Each running analytic server instance can hold one analytic instance.
Analytic engine	The portion of the analytic server framework that's responsible for managing analytic instances.
Analytic engine type	Select one of the following options: • Analytic Server. • Application Engine Server. • Application Engine.

Analytic Server Features

The analytic server framework has the following features:

- It's dedicated to the storage and management of large models.
- It's supported on all PeopleTools application server platforms.
- It runs PSANALYTICSRV as a Tuxedo managed server.
- You can configure the minimum and maximum number of analytic server instances per domain that are running at one time.
- You can specify a timeout for a loaded analytic instance. If the analytic instance isn't referenced within the timeout interval, it's discarded.

- Multiple domain environments are supported, in which an analytic instance can be loaded in one Tuxedo domain, and a user can access the analytic instance from another domain.
- You install, configure, and administer analytic servers using the same facilities as with other servers.
- You can shut down an analytic server and discard any loaded analytic instances.
- You use the standard PeopleTools mechanisms to troubleshoot, trace, log and debug analytic servers.

Related Links

"Using the Quick-Configure Menu" (System and Server Administration)
"Application Servers" (System and Server Administration)
"Domain Settings" (System and Server Administration)
"PeopleSoft Optimization Framework Overview" (Optimization Framework)
"Performance Monitor Overview" (Performance Monitor)

Analytic Server Process Flow and Behavior

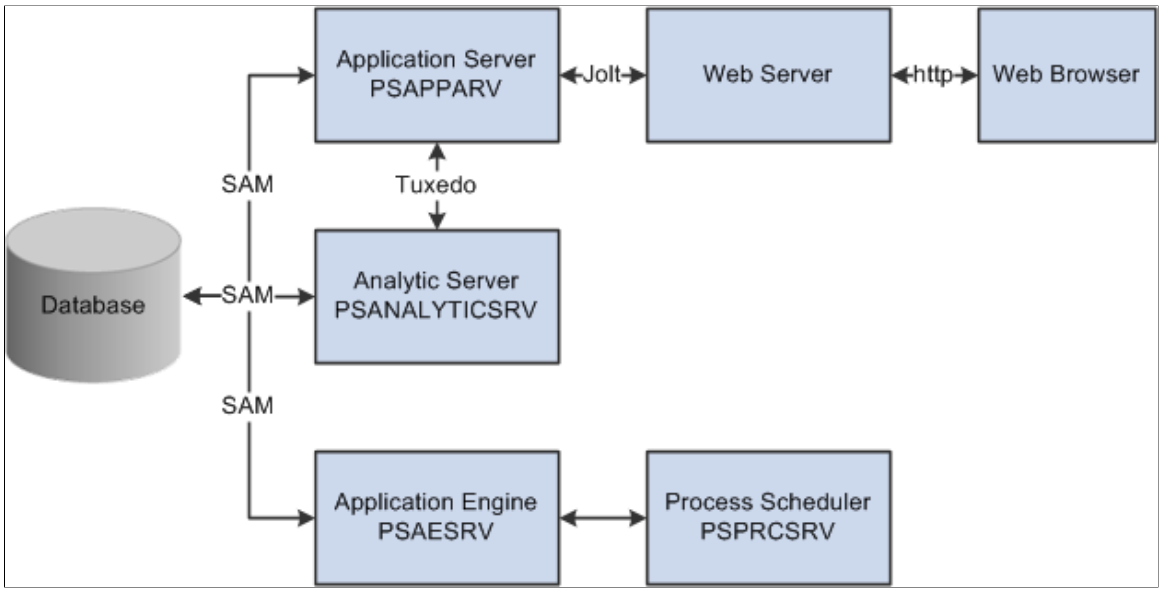
PeopleSoft session activity, such as a user action, a component interface operation, or a message subscription, launches PeopleCode that requires the application server to invoke the analytic calculation engine or the optimization engine to process an analytic instance.

The database maintains a list of all the available PSANALYTICSRV (analytic server) instances, their status, and any analytic instances currently loaded, so it can properly select analytic server instances for new analytic instances, and direct subsequent requests to the proper analytic server instance. When an analytic server instance starts, the database is updated.

When a running program requests the creation of an analytic instance, the analytic server framework considers all available PSANALYTICSRV instances in the same application server domain and allocates one of them from the pool of idle server instances to handle this particular analytic instance. Any further load or recalculate operations requested by the application for this analytic instance are directed back to the same analytic server instance for processing.

Note: If there are no idle server instances, the analytic server framework can spawn additional server instances up to a maximum limit that you can define in the application server domain configuration. If this maximum is reached, the system attempts to allocate a server instance from a running analytic server in another domain.

This example illustrates the analytic server architecture, including the database, application server, web server, web browser, analytic server, application engine server, and process scheduler server.



After an analytic server instance takes over processing the analytic instance, that processing becomes independent of the status of the application server. The core functionality provided by the analytic server framework is the ability to host analytic instances for an indeterminate amount of time in an environment where that data can persist across multiple sessions, and where that data can be accessed without requiring its entire content to be transferred from server to client and back.

Analytic Instance Access

Access to the analytic instances maintained by the analytic server environments is supported only through PeopleCode programs. The environments in which PeopleCode can run include:

- The application server (PSAPPSRV).
- PeopleSoft Application Engine accessed by PeopleSoft Process Scheduler (PSAESRV or psae)
- PeopleSoft Application Engine run from the command line (psae).

An allocated server instance doesn't need to be running in the same Tuxedo domain or on the same server machine as the application server. Once it's allocated, the initiating user is redirected to an application server that's running on the same domain and server machine as the analytic server.

PeopleCode that's running in any PSAPPSRV process can access analytic instances that were loaded by any other PSAPPSRV process, regardless of the Tuxedo domain in which it's running. When the program requests access to such an analytic instance, the running PeopleCode program is restarted and the web server is notified to redirect the request back to an application server within the Tuxedo domain where the analytic instance is loaded. This application server is then able to directly contact the PSANALYTICSRV server with the loaded analytic instance.

Note: PeopleCode that's running in a given Application Engine environment can access only analytic instances that were loaded in the same process.

Secondary Database Connection

A secondary database connection is used to prevent unexpected table locks when you run an analytic calculation. The secondary connection isn't opened until an analytic instance is referenced in a PeopleCode program. A secondary connection is required regardless of whether the analytic calculation is run by an application engine program as a batch process or directly by an online application.

By default, the secondary connection is persistent for improved performance. If you find that the persistent connection imposes too much overhead, you can change it to an on-demand connection by setting bit eight of the DbFlags application server and process scheduler domain parameter.

Note: A non-persistent connection can significantly affect system performance, so consider this setting carefully.
You can use DbFlags bit four to disable the secondary connection altogether, but analytic instance processing requires it, so ensure that DbFlags does not have bit four set.

Errors and Abnormal Process Termination

Any errors that occur while processing an analytic server request result in the PeopleCode program returning an error code or throwing a PeopleCode exception.

If an analytic server instance that's hosting an analytic instance terminates unexpectedly, the loaded analytic instance is lost and unrecoverable. However, the analytic instance status still appears on the Analytic Server Administration pages. The domain monitor (PSMONITORSRV) discovers the unexpected termination and cleans up the status information.

Related Links

“DbFlags” (System and Server Administration)
“Load” (PeopleCode API Reference)
“PeopleSoft Process Scheduler Overview” (Process Scheduler)

Understanding Batch Processing of Analytic Instances

The analytic server framework integrates with and works with PeopleSoft Process Scheduler using PeopleSoft Application Engine, because PeopleSoft Application Engine can access the analytic calculation engine and the optimization engine directly.

When PeopleSoft Process Scheduler launches an Application Engine job to process an analytic instance, PeopleSoft Application Engine handles the entire job directly by loading the analytic engine within its own process rather than using a server.

This is true whether PeopleSoft Process Scheduler submits the job to the PSAESRV process, or launches PeopleSoft Application Engine using the *psae* command.

Related Links

“PeopleSoft Optimization Framework Overview” (Optimization Framework)

Configuring and Starting Analytic Servers

This topic discusses how to configure and start analytic servers.

You can specify whether an application server domain includes the PSANALYTICSRV Tuxedo managed application server process, and specify the maximum number of analytic server instances that you want the domain to support. You use the Quick-Configure menu of the PSADMIN utility to enable, configure, and start analytic server instances.

Enabling PSANALYTICSRV

You access the Quick-Configure menu of PSADMIN by selecting *Configure This Domain* from the Domain Administration menu.

On the Quick-Configure menu, enter the menu item number for *Analytic Servers* to toggle the setting for that entry to *Yes*, so the domain will include instances of PSANALYTICSRV when it boots.

Related Links

"Using the Quick-Configure Menu" (System and Server Administration)

Specifying Analytic Server Instance Quantities

Before you boot the application server domain, specify the appropriate minimum and maximum number of allowed analytic server instances. The values you specify depend on your assessment of how many users you expect to be using applications that process analytic instances.

Consider the typical number of analytic instances in a domain that are being processed at any given moment as your minimum, and the possible total number of analytic instances that might simultaneously require processing as your maximum. The most appropriate values produce the fastest system response without unused server instances consuming memory unnecessarily.

To specify analytic server instances:

1. On the Quick-Configure menu for the domain, enter the menu number for *Custom Configuration*.

   The Custom Configuration environment launches, and prompts you to indicate whether you want to change any configuration values.

2. Enter *y* to indicate that you want to change configuration values.

   Custom Configuration prompts you to decide whether to change any values for each configuration item in turn.

3. Press **Enter** to accept the default answer for each item presented, until the following entry appears:

   ```
   Values for config section - PSANALYTICSRV
   ```

4. Enter *y* to change the values for PSANALYTICSRV.

   You're prompted for each value in turn.

5. Specify the minimum number of instances.

This defines the number of analytic server instances that start when you boot the application server domain. There are always at least this number of instances running. The default value of this parameter is *3*.

6. Specify the maximum number of instances.

   This defines the maximum number of analytic server instances that can result from spawning new processes. The default value of this parameter is *3*.

7. Press **Enter** to accept the default answer for each subsequent item presented. When you respond to the last item, PSADMIN loads the new configuration and the PeopleSoft Domain Administration Menu appears.

   You now can boot the domain normally.

Notice that one analytical server can load only one analytic instance. If no analytical server is available to access analytical instance, you may experience one of these two scenarios:

- Only one tuxedo domain is available.

  This scenario happens when analytical instance accesses `'n+1' th` analytical server where `n` is the maximum number of analytical servers configured. In this scenario, a message appears to alert you that no analytical server is currently available.

- More than one tuxedo domains are available.

  This scenario happens when analytical instance accesses `'n+1' th` analytical server. Analytical instance checks for available tuxedo domains and redirects the request to the domain where analytical server is available. If no analytical server is available in these tuxedo domains, a message appears to alert you that no analytical server is currently available.

Related Links

“Using the Quick-Configure Menu” (System and Server Administration)
“PSANALYTICSRV Options” (System and Server Administration)

Starting PSANALYTICSRV

If you enabled analytic servers on the Quick-Configure menu, when you boot the application server domain, the PSANALYTICSRV process starts with the minimum number of instances that you specified.

When an application running under this domain requests an analytic instance, the analytic server framework allocates an available idle analytic server instance for that analytic instance. If no idle server instance is available, the framework spawns and allocates an additional server instance, up to the maximum that you defined.

Administering Analytic Servers

This topic discusses how to administer analytic server domains and instances.

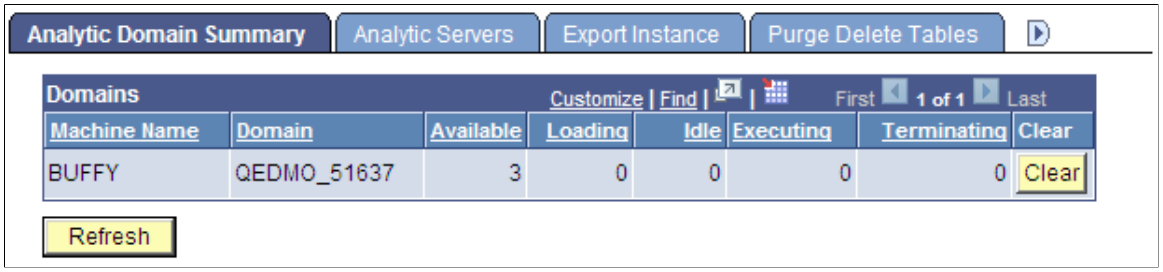
Administering Analytic Server Domains

Use the Analytic Server Administration - Analytic Domain Summary page (PTATADMIN_SUMMARY) to view the current status of the application server domains, which are attached to the current database, running with PSANALYTICSRV. Each active domain is listed along with its machine name, domain name, and the total number of availability, loading, idle, executing, terminating, and clear

Navigation:

PeopleTools > **Utilities** > **Administration** > **Analytic Server Administration**

This example illustrates the fields and controls on the Analytic Server Administration - Analytic Domain Summary page. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Machine Name	Displays the network name of the computer on which the listed domain is running.
Domain	Displays the name of each active domain. **Note:** If a domain has been unexpectedly terminated, it might still be listed here. You can click the **Clear** button to remove the outdated information from the display.
Available	Displays the total number of analytic server instances running in the domain.
Loading	Displays the number of available analytic server instances in the domain that are currently being loaded.
Idle	Displays the number of available analytic server instances in the domain that are allocated to analytic instances, but aren't actively processing them.
Executing	Displays the number of available analytic server instances in the domain that are allocated to analytic instances, and are actively processing them.
Terminating	Displays the number of analytic server instances in the domain that are marked as terminated, but haven't yet been shut down.

Field or Control	*Description*
Clear	Click to remove the row from the display when the domain has been unexpectedly terminated. ***Warning!*** Use the **Clear** button with caution, and only if you're certain that the domain has been unexpectedly terminated. Clearing the domain information for servers that are still running produces errors in those servers.
Refresh	Click to retrieve information about any newly started domains that have PSANALYTICSRV running.

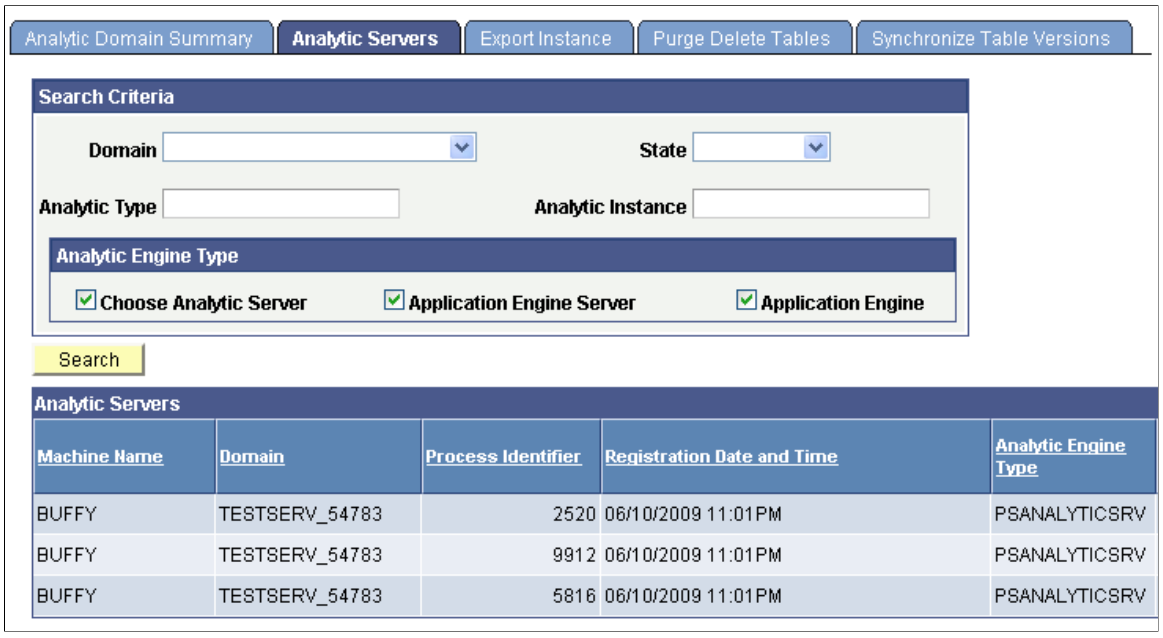
Administering Analytic Server Instances

Use the Analytic Server Administration - Analytic Servers page (PTATADMIN_SERVERS) to administer analytic server instances.

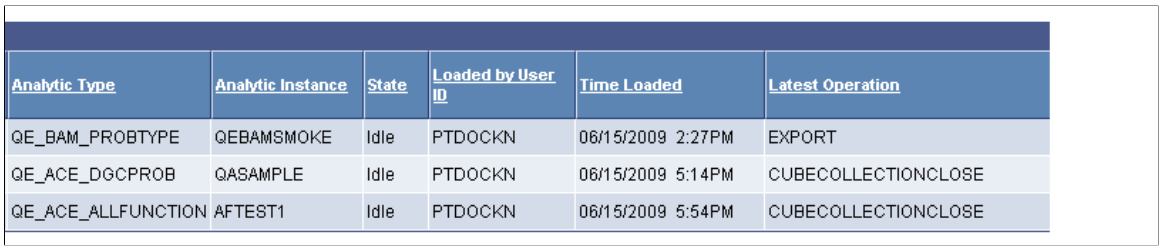
Navigation:

PeopleTools > **Utilities** > **Administration** > **Analytic Server Administration** > **Analytic Servers**

This example illustrates the fields and controls on the Analytic Server Administration - Analytic Servers page (1 of 3). Definitions for the fields and controls appear following the examples.

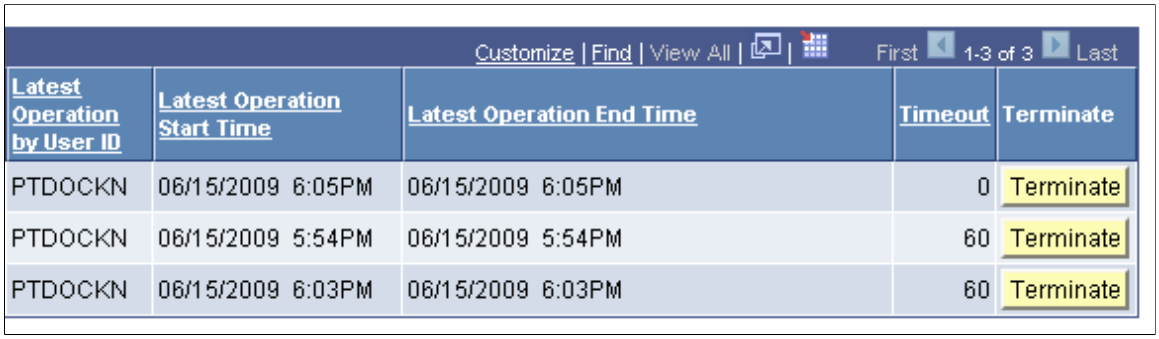


This example illustrates the fields and controls on the Analytic Server Administration - Analytic Servers page (2 of 3). Definitions for the fields and controls appear following the examples.



Analytic Type	Analytic Instance	State	Loaded by User ID	Time Loaded	Latest Operation
QE_BAM_PROBTYPE	QEBAMSMOKE	Idle	PTDOCKN	06/15/2009 2:27PM	EXPORT
QE_ACE_DGCPROB	QASAMPLE	Idle	PTDOCKN	06/15/2009 5:14PM	CUBECOLLECTIONCLOSE
QE_ACE_ALLFUNCTION	AFTEST1	Idle	PTDOCKN	06/15/2009 5:54PM	CUBECOLLECTIONCLOSE

This example illustrates the fields and controls on the Analytic Server Administration - Analytic Servers page (3 of 3). Definitions for the fields and controls appear following the example.



Customize | Find | View All | First 1-3 of 3 Last

Latest Operation by User ID	Latest Operation Start Time	Latest Operation End Time	Timeout	Terminate
PTDOCKN	06/15/2009 6:05PM	06/15/2009 6:05PM	0	Terminate
PTDOCKN	06/15/2009 5:54PM	06/15/2009 5:54PM	60	Terminate
PTDOCKN	06/15/2009 6:03PM	06/15/2009 6:03PM	60	Terminate

Search Criteria

Field or Control	*Description*
Search	Click to retrieve status information about all analytic server instances that are running in application server domains that are attached to the current database. You can use the Search Criteria section to limit the information returned based on various criteria.
Domain	Select the name of an active application server domain for which you want to retrieve information.

Field or Control	*Description*
State	Select a server state to limit the search to analytic server instances with the selected state. You can select from the following states: • *Available* • *Registered* • *Loading* • *Idle* • *Executing* • *Terminate*
Analytic Type	Select an analytic type from the set of analytic types defined in the current database. This limits the search to analytic server instances that have analytic instances of the selected analytic type loaded.
Analytic Instance	Select an analytic instance from the set of analytic instances defined in the current database. This limits the search to analytic server instances that have the selected analytic instance loaded.
Analytic Engine Type	Specify the process types for which you want to get status information. Select one or more of the following: • *Analytic Server* • *Application Engine Server* • *Application Engine* All three types are selected by default.

Analytic Servers

This section displays a row of status information retrieved for each analytic server instance that's returned by the search. In addition to the fields documented in the previous section, each row displays the following information.

Field or Control	*Description*
Machine Name	Displays the network name of the computer on which the listed analytic server instance is running.

Field or Control	*Description*
Process Identifier	Displays the operating system process ID for the listed analytic server instance.
Registration Date and Time	• Analytic server type: Displays the date and time that this analytic server booted. • Application Engine types: Displays the date and time that the application engine process loaded this analytic instance.
Loaded by User ID	Displays the user ID of the user whose activity resulted in the allocation of this analytic server instance.
Time Loaded	Displays the date and time that this analytic server instance loaded its analytic instance.
Latest Operation	Identifies the last operation that was applied to this analytic instance.
Latest Operation By User ID	Displays the user ID of the last user to access this analytic instance.
Latest Operation Start Time	Displays the date and time that the last operation on this analytic instance started.
Latest Operation End Time	Displays the date and time that the last operation on this analytic instance completed.
Timeout	Displays the timeout interval in minutes that's specified for this analytic instance. Timeout values are defined for analytic instances by the TimeOut parameter of the PeopleCode AnalyticInstance class Load method. A value of *0* indicates an unlimited lifespan for this analytic instance.
Terminate	Click to indicate that the server instance should be shut down.

Administering Analytic Tables

This topic discusses how to administer analytic tables.

Purging Delete Tables

Delete tables contain rows that have been deleted from analytic instance working data. These tables accumulate data when you use triggers for database level auditing, and they're not always cleaned up after the deletes have been completed.

Use the Analytic Server Administration - Purge Delete Tables page (PTATADMIN_PURGE) to accomplish the cleanup manually.

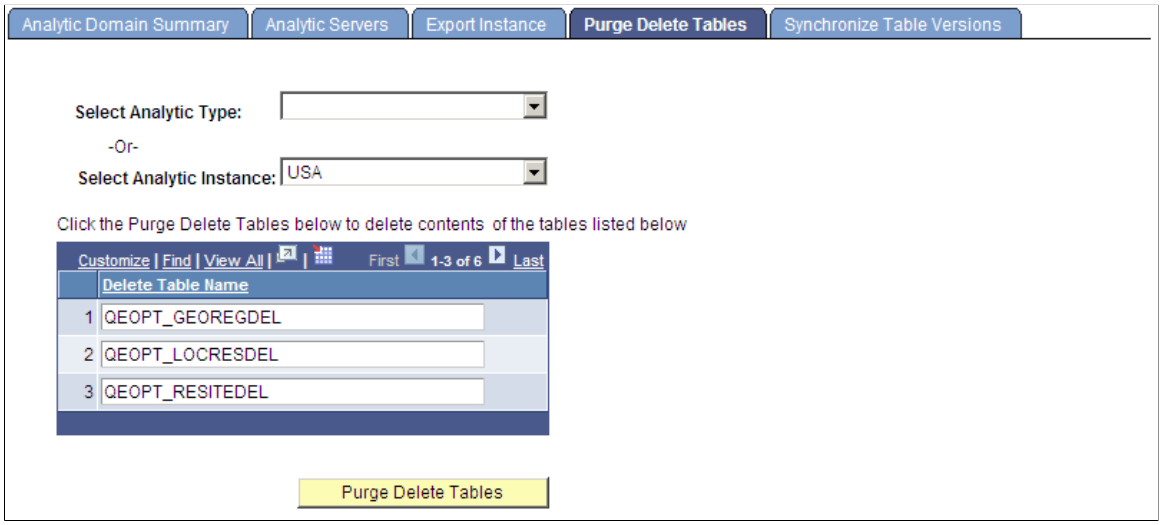
Navigation:

PeopleTools > Utilities > Administration > Analytic Server Administration > Purge Delete Tables

---

Note: Shut down all running analytic server processes before using this page.

This example illustrates the fields and controls on the Analytic Server Administration - Purge Delete Tables page. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Select Analytic Type -Or- **Select Analytic Instance**	These drop-down lists are mutually exclusive. Select either an analytic type or an analytic instance for which you want to purge delete tables.
Delete Table Name	Displays the names of the delete tables relevant to the analytic type or analytic instance that you selected.
Purge Delete Tables	Click to clear the data from the displayed delete tables.

Related Links

“Understanding Database Level Auditing” (Data Management)

“Creating Audit Record Definitions” (Data Management)

Synchronizing Table Versions

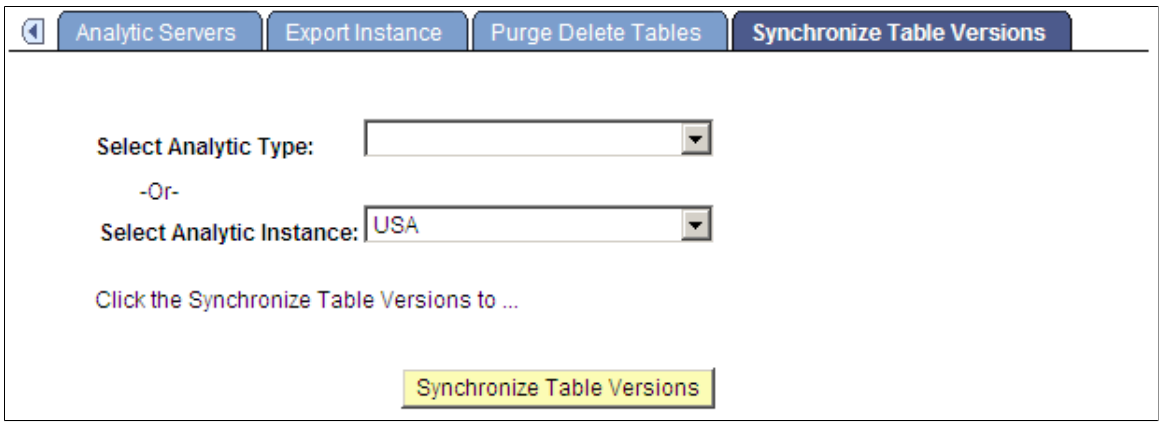
Some scenario-managed optimization tables used with an analytic type have a version number field. The analytic server framework maintains a list of the tables and their version numbers. After an upgrade, the version numbers in the upgraded tables might not match the version numbers on this list. You use the Analytic Server Administration - Synchronize Table Versions page (PTATADMIN_SYNCVER) to update the list so the version numbers match.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Analytic Server Administration** > **Synchronize Table Versions**

Note: Shut down all running analytic server processes before using this page.

This example illustrates the fields and controls on the Analytic Server Administration - Synchronize Table Versions page. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Select Analytic Type -Or- **Select Analytic Instance**	These drop-down lists are mutually exclusive. Select either an analytic type or an analytic instance for which you want to synchronize table versions.
Synchronize Table Versions	When you use PeopleSoft Data Mover to move data from one database to another, it's often the case that the versions of analytic type or analytic instance data and the PSOPTSYNC table are out of synchronization. Click this button to synchronize the PSOPTSYNC table with the analytic instance tables.

Creating, Deleting, and Copying Analytic Instances

This topic discusses how to create, delete and copy analytic instances.

Note: You can create, delete and copy analytic instances for use with both Analytic Calculation Engine and PeopleSoft Optimization Framework.

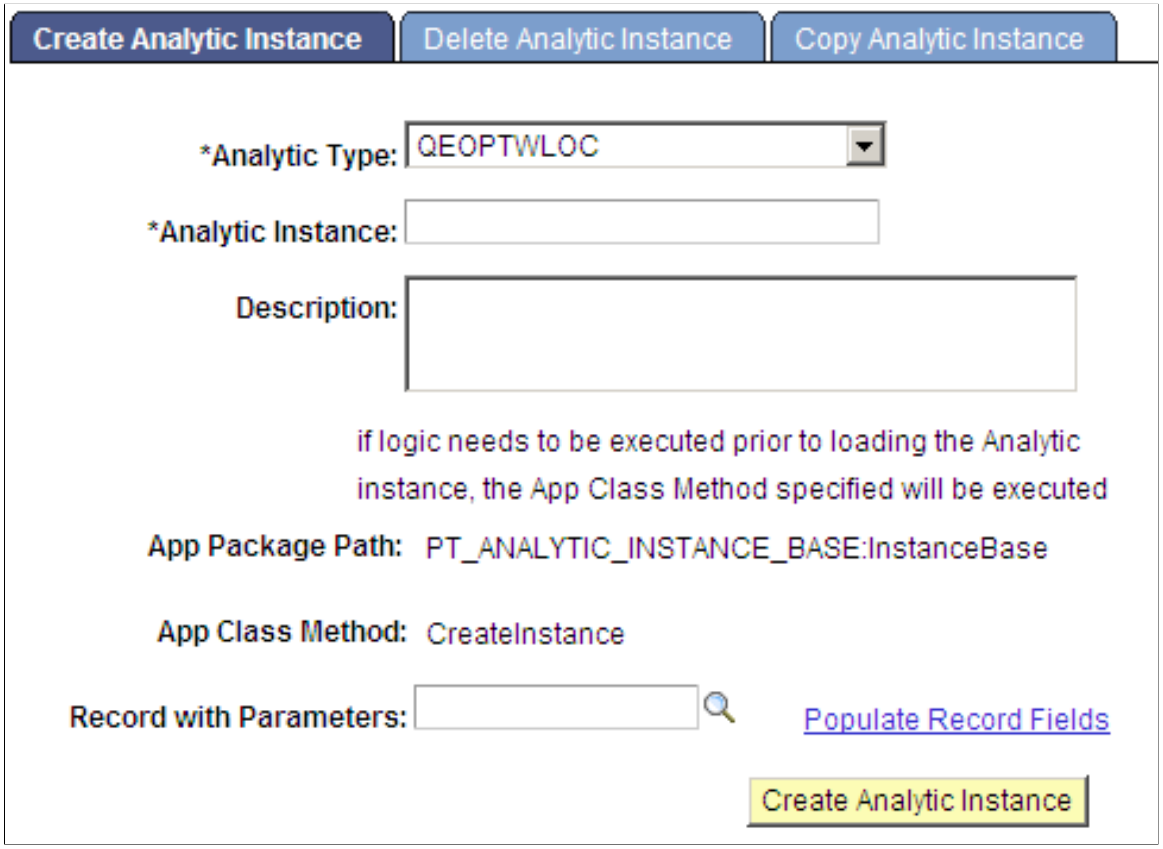
Creating Analytic Instances

Use the Create Analytic Instance page (PTACECRTINST) to create an analytic instance that you can then load and view within the Analytic Model Viewer to inspect and debug your analytic model.

Navigation:

PeopleTools > Utilities > Administration > Maintain Analytic Instances

This example illustrates the fields and controls on the Create Analytic Instance page. Definitions for the fields and controls appear following the example.



Field or Control	*Description*
Analytic Type	Select an analytic type from the drop-down list. See Purpose of Analytic Type Definitions, "Creating Analytic Type Definitions" (Optimization Framework).
Analytic Instance	Enter a name for the analytic instance. Analytic instance names should consist of alphanumeric characters, can be up to 20 characters long, and cannot include spaces.

Field or Control	*Description*
App Package Path and **App Class Method**	The **App Package Path** field displays the full name of an application class (application package name, subpackage names if applicable, and class name) that's used to execute logic before loading the analytic instance. The **App Class Method** field displays the name of the method in the displayed class that's called at creation time to populate the new analytic instance with data. You establish the application package class and method to use when you define the analytic type. See “CreateAnalyticInstance” (PeopleCode Language Reference).
Record with Parameters	Look up and select parameters to be passed to the application class method. Click the lookup button to display a list of records. Selecting any record generates a standalone record.
Populate Record Fields	Displays a secondary page that lets you populate the fields of the standalone record; the values of these fields will serve as parameters passed into the App Class Method.
Create Analytic Instance	Create the analytic instance. After the analytic instance has been successfully created, you receive a notification to that effect.

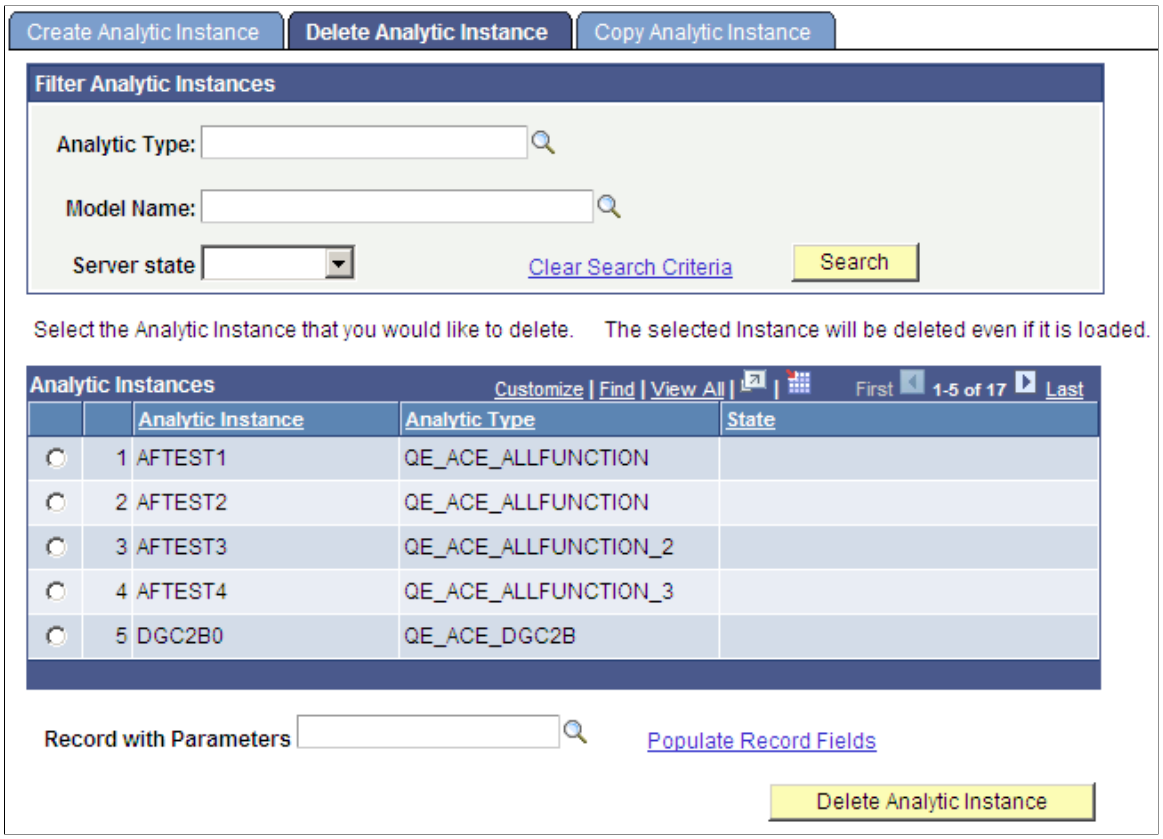
Deleting Analytic Instances

Use the Delete Analytic Instance page (PTACEDELINST) to specify search criteria to filter the display of returned analytic instances based on a combination of analytic type, model name, and server state; and then unload the selected analytic instance and delete the data associated with it.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Maintain Analytic Instances** > **Delete Analytic Instance**

This example illustrates the fields and controls on the Delete Analytic Instance page. Definitions for the fields and controls appear following the example.



Field or Control	***Description***
Analytic Type	Look up and select the analytic type upon which the analytic instance is based.
Model Name	Look up and select the analytic model upon which the analytic instance is based.
Server State	Select one of the following: • *Idle* • *Loading* • *Executing* • *Terminating* See Administering Analytic Servers.
Clear Search Criteria	Click to delete any current search criteria so you can begin a search from scratch.

Field or Control	*Description*
Search	Click to display all analytic instances that meet the specified search criteria. Select one of the displayed analytic instances to delete.
Record with Parameters	Look up and select parameters to be passed to the application class method. Click the lookup button to display a list of records. Select a record, the first row of which will consist of parameters that you want to pass to the application class method. Selecting any record generates a populated, standalone record.
Populate Record Fields	Displays a secondary page that lets you populate the fields of the standalone record; the values of these fields will serve as parameters passed into the App Class Method.
Delete Analytic Instance	Unload the selected analytic instance and delete the data associated with it.

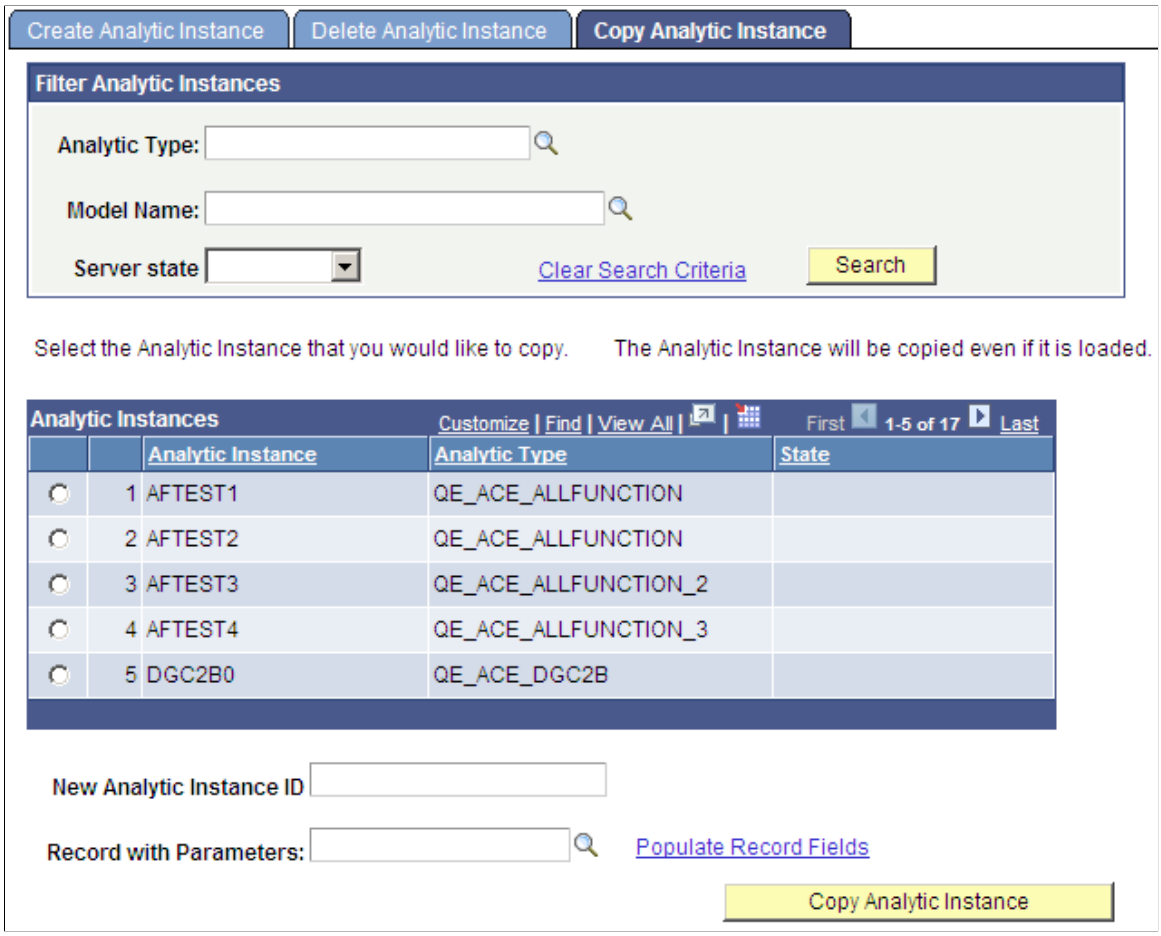
Copying Analytic Instances

You use the Copy Analytic Instance page (PTACECPYINST) to specify search criteria to filter the display of returned analytic instances based on a combination of analytic type, model name, and server state; and then copy the selected analytic instance and its associated data.

Navigation:

PeopleTools > **Utilities** > **Administration** > **Maintain Analytic Instances** > **Copy Analytic Instance**

This example illustrates the fields and controls on the Copy Analytic Instance page. Definitions for the fields and controls appear following the example.



Field or Control	***Description***
Analytic Type	Look up and select the analytic type upon which the analytic instance is based.
Model Name	Look up and select the analytic model upon which the analytic instance is based.
Server State	Select one of the following: • *Idle* • *Loading* • *Executing* • *Terminating* See Administering Analytic Servers.

Field or Control	*Description*
Clear Search Criteria	Click to delete any current search criteria so you can begin a search from scratch.
Search	Click to display all analytic instances that meet the specified search criteria. Select one of the displayed analytic instances as the source instance to copy.
New Analytic Instance ID	Enter a name for the new analytic instance; this analytic instance will be a copy of the selected source instance.
Record with Parameters	Look up and select parameters to be passed to the application class method that will copy the source analytic instance. Click the lookup button to display a list of records. Select a record, the first row of which will consist of parameters that you want to pass to the application class copy method. Selecting any record generates a populated standalone record.
Populate Record Fields	Displays a secondary page that lets you populate the fields of the standalone record; the values of these fields will serve as parameters passed into the application class copy method.
Copy Analytic Instance	Copy the selected analytic instance and its associated data. If a tree is attached to the selected analytic instance, all tree data is also copied to the new analytic instance. **Note:** The analytic instance data and tree data are copied only if the record with parameters that you specified is populated with the source analytic instance ID.

Loading and Unloading Analytic Instances

This topic discusses how to load and unload analytic instances.

Note: You can load and unload analytic instances for use with both the Analytic Calculation Engine and PeopleSoft Optimization Framework.

Loading and Unloading Analytic Instances

To use the Analytic Model Viewer, you must load an analytic instance of the analytic model that you want to view or debug. You load analytic instances by using:

- The Analytic Instance Load/Unload page.
- The Analytic Model Viewer.

You unload instances by using the Analytic Instance Load/Unload page (PTACEMDLLOAD).

It is quicker to load an analytic instance by going through the Analytic Model Viewer: This approach allows you to simultaneously load and view the analytic instance. However, you can use the Analytic Instance Load/Unload page if you need to modify the timeout value or attach or detach a tree. You need to attach a tree before loading an analytic instance if you want to see the tree structure while reviewing this analytic instance within the Analytic Model Viewer.

See Understanding Analytic Model Properties, Understanding the Relationship of PeopleSoft Trees to Analytic Models.

Note: You can only load one analytic instance per analytic server.

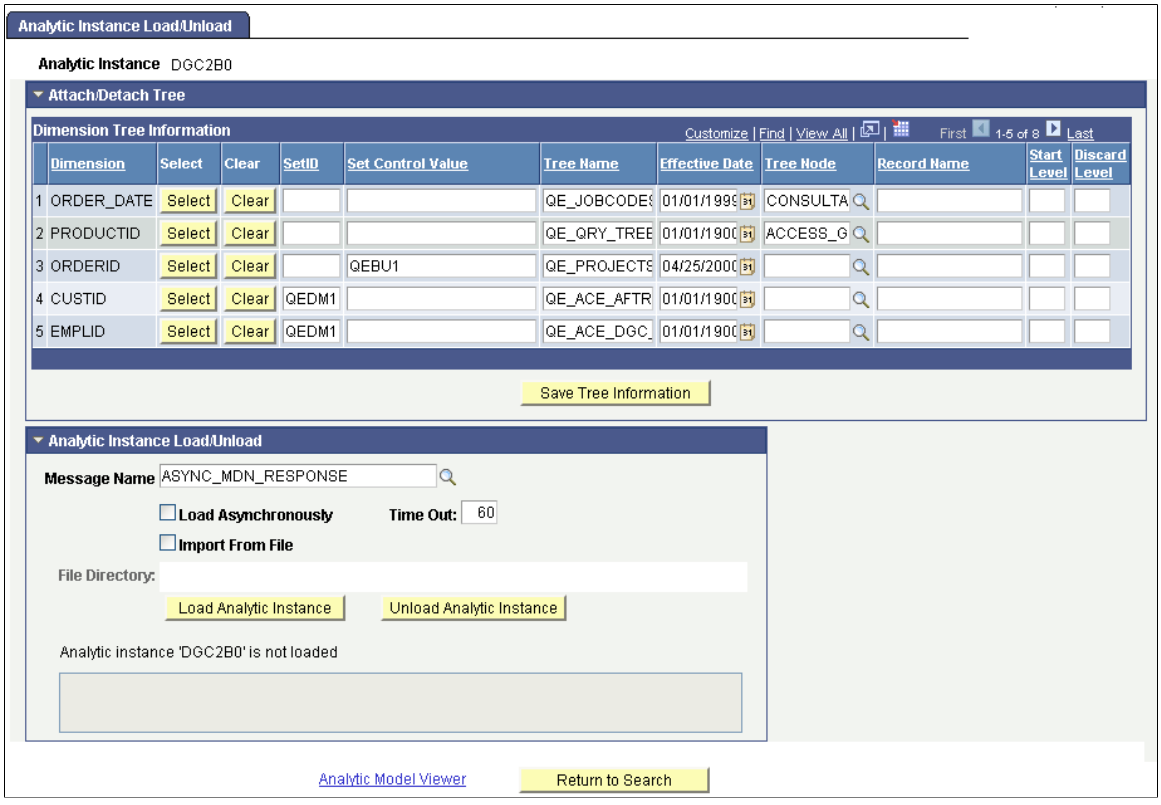
You can also load and unload analytic instances by means of PeopleCode, using the AnalyticInstance class Load or Unload methods.

See “AnalyticInstance Class Methods” (PeopleCode API Reference).

Navigation:

1. Select **PeopleTools** > **Utilities** > **Administration** > **Load/Unload Analytic Instances.**
2. Selecting the desired analytic instance on the search results page.

This example illustrates the fields and controls on the Analytic Instance Load/Unload page. Definitions for the fields and controls appear following the example.



Note: If the desired analytic instance is not displayed, you need to create it.

See Creating, Deleting, and Copying Analytic Instances.

Attach/Detach Tree

Note: If you selected a PeopleSoft Optimization Framework analytic instance, the Attach/Detach Tree section of the Analytic Instance Load/Unload page is not displayed.

Field or Control	*Description*
Dimension	Lists the dimensions in the selected analytic instance.
Select	Click to select a tree to attach to the dimension. A secondary page appears from which you can select a tree.
Clear	Click to disassociate a selected tree from the dimension.
SetID	Displays the setID associated with the tree, if applicable.
Set Control Value	Displays the Set Control Value associated with the tree, if applicable.
Tree Name	Displays the name of the selected tree.
Effective Date	Displays the effective date associated with the tree.
Tree Node	Specify a node from the selected tree.
Record Name	Displays the name of a record containing override rules or functions. See Understanding the Relationship of PeopleSoft Trees to Analytic Models.
Start Level	Enter a number to specify the type of dimension members that Analytic Calculation Engine creates out of the nodes and leaves of a tree. The default value of this field is *0*. The root level is 1. **Note:** If you specify a nonzero start level, you must specify the strictly enforced method to the tree in PeopleSoft Tree Manager. The strictly enforced method ensures that all members that are created out of one level are created as the same data type. See Purpose of Node Levels in Creating Hierarchies.

Field or Control	*Description*
Discard Level	Enter a number to specify the level from which Analytic Calculation Engine does not attach any more of the tree to the dimension. Analytic Calculation Engine does not create members out of nodes or leaves that are either at this level or lower than this level. You must specify a start level to every tree for which you want to specify a discard level. The default value of this field is *0*. If you specify any other value, then it must be at a lower level (a higher number) than the start level. See Purpose of Node Levels in Creating Hierarchies.
Save Tree Information	Click to save the dimension tree information that you've selected. The updated tree information takes effect the next time you load the analytic instance.

Analytic Instance Load/Unload

Field or Control	*Description*
Message Name	Specify an application message that should be sent if the analytic instance can't be unloaded successfully and is terminated prematurely. This can happen if the analytic server crashes while the analytic instance is loaded. **Note:** The message is sent when the analytic server process restarts itself after crashing. The long edit box in this section of the page displays the content of detailed messages.
Load Asynchronously	Select to indicate that the analytic instance should be run asynchronously.
Time Out	Enter the number of minutes of inactivity before the analytic instance times out. The default time out is the value specified for the Analytic Instance Idle Timeout domain parameter. See “Analytic Instance Idle Timeout” (System and Server Administration). **Note:** After an instance times out, you must reload it to continue working with it.

Field or Control	*Description*
Import from File	Import an analytic instance from a file. You use this option to import an analytic instance that you've captured with the Analytic Instance Capture Utility. See Understanding the Analytic Instance Capture Utility, Capturing Analytic Instance Data, Importing Analytic Instance Data.
File Directory	Specify the directory from which you want to retrieve the analytic instance that you are importing from file.
Load Analytic Instance	Click to load the selected analytic instance. Analytic Calculation Engine displays a confirmation message after it successfully loads the analytic instance:
Unload Analytic Instance	Click to unload the selected analytic instance. Analytic Calculation Engine displays a confirmation message after it successfully unloads the analytic instance. You must unload analytic instances once you're done working with them.

Related Links

“Load” (PeopleCode API Reference)