Oracle® Essbase

Database Administrator's Guide for Oracle Essbase

F17141-16
August 2024

Oracle Essbase Database Administrator's Guide for Oracle Essbase,

F17141-16

Primary Author: Essbase Information Development Team

Contents

2 Understanding Multidimensional Databases

3 Creating Applications and Databases

4 Creating and Changing Database Outlines

5 Creating and Working With Duplicate Member Outlines

6 Setting Dimension and Member Properties

7 Working with Attributes

10 Designing Partitioned Applications

11 Creating and Maintaining Partitions

12 Overview of Data Load and Dimension Build

13 Design Rules for Data Load or Dimension Build

15 Dimension Build Examples

21 Intelligent Calculation for Block Storage Cubes

22 Dynamically Calculating Data Values

23 Calculating Time Series Data

24 Develop Calculation Scripts for Block Storage Cubes

25 Examples of Calculation Scripts for Block Storage Databases

26 Using Parallel Calculation

27 Developing Custom-Defined Calculation Functions

35 Block Storage Calculation Optimization

40 Manage Aggregate Storage Applications and Cubes

A Limits

B Naming Conventions for Essbase

Accessibility and Support

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.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 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Case Study: Designing a Single-Server, Multidimensional Database

This case study provides an overview of the Essbase database planning process and discusses working rules that you can follow to design a single-server, multidimensional database solution for your organization.

Process for Designing a Database

To design an Essbase database (cube), model it around your organization's information needs. The location and structure of the source data matter less than how you model the outline, which determines how the pieces of information and associated metrics are related in your cube.

When you implement a multidimensional database, you design and create an application and database. You analyze data sources and define requirements carefully and decide whether a single-server approach or a partitioned, distributed approach better serves your needs. For criteria that you can review to decide whether to partition an application, see Guidelines for Partitioning a Database.

This case study provides an overview of the database planning process and discusses working rules that you can follow to design a single-cube, multidimensional database solution for your organization. See Creating Applications and Databases.

The process of designing a database includes the following basic steps:

1. Analyze business needs and design a plan.

   The application and database that you create must satisfy the information needs of your users and your organization. Therefore, you identify source data, define user information access needs, review security considerations, and design a database model. See Analyzing and Planning.

2. Draft a database outline.

   The outline determines the structure of the database—what information is stored and how different pieces of information interrelate. See Drafting Outlines.

3. Load test data into the database.

After an outline and a security plan are in place, you load the database with test data to enable the later steps of the process. See Loading Test Data.

4. Define calculations.

   You test outline consolidations, write and test formulas, and define calculation scripts for specialized calculations. See Defining Calculations.

5. Verify with users.

   To ensure that the database satisfies your user goals, solicit and carefully consider their feedback. See Verifying the Design.

6. Repeat the process.

   To fine-tune the design, repeat steps 1 through 5.

Case Study: The Beverage Company

This case study bases the database planning process on the needs of a fictitious company, *The Beverage Company* (TBC), as an example for how to build an Essbase database. TBC is a variation of the Sample.Basic application that is included with the Essbase installation.

TBC manufactures, markets, and distributes soft drink products internationally. Analysts at TBC prepare budget forecasts and compare performance to budget forecasts monthly. The financial measures that analysts track are profit, loss, and inventory.

TBC uses spreadsheet packages to prepare budget data and perform variance reporting. Because TBC plans and tracks a variety of products over several markets, the process of deriving and analyzing data is tedious. Last month, analysts spent most of their time entering and rekeying data and preparing reports.

TBC has determined that Essbase is the best tool for creating a centralized repository for financial data. The data repository will reside on a server that is accessible to analysts throughout the organization. Users can load data from various sources and retrieve data as needed. TBC has a variety of users, so TBC expects that different users will have different security levels for accessing data.

Analyzing and Planning

To ensure that your Essbase analyzes your business information efficiently, formulate a detailed plan that outlines data sources, user needs, and prospective database elements. Attention to this design phase can save you development and implementation time.

The planning and analysis phase involves these tasks:

- Analyzing Source Data
- Identifying User Requirements
- Planning for Security in a Multiple User Environment
- Creating Database Models

When designing a multidimensional application, consider these factors:

- How information flows within the company—who uses which data for what purposes
- The types of reporting the company does—what types of data must be included in the outline to serve user reporting needs

> **Note:**
>
> Defining only one database per application enhances memory usage and ease of database administration.

Analyzing Source Data

First, evaluate the source data to be included in the database. Think about where the data resides, its network connectivity, and the frequency and size of required updates. This up-front research saves time when you create the database outline and load data into the Essbase database.

Determine the scope of the database. If an organization has numerous product families containing a vast number of products, you may want to store data values only for product families. Interview members from each user department to find out what data they process, how they calculate and report data today, and how they want to do it in the future. You should store in Essbase only what is needed for multi-dimensional pivot reporting and drill-through. The remainder of the data can remain in a relational database and be partitioned in (federated) or drilled through.

Carefully define reporting and analysis needs.

- How do users want to view and analyze data?
- How much detail should the database contain?
- Does the data support the desired analysis and reporting goals?
- If not, what additional data do you need, and where can you find it?

Determine the location of the current data.

- Where does each department currently store data?
- Is data in a form that Essbase can use?
- Do departments store data in relational databases on Windows or UNIX servers, or in Excel spreadsheets?
- Who updates the database and how frequently?
- Do those who need to update data have access to it?

Ensure that the data is ready to load into Essbase.

- Does data come from a single source or multiple sources?
- Is data in a format that Essbase can use? For a list of valid data sources that you can load into Essbase, see Sources of Data.
- Is all data that you want to use readily available?

Identifying User Requirements

Discuss information needs with users and request sample reports from them. Review the information they use and the reports they must generate for review by others. Determine the following requirements:

- What types of analysis do users require?
- Do users require ad-hoc (pivot style) reporting and structured reports?

- What summary and detail levels of information do users need?
- Do some users require access to information that other users should not see?

Planning for Security in a Multiple User Environment

Consider user information needs when you plan how to set up security permissions. End your analysis with a list of users and permissions.

Use this checklist to plan for security:

- Who are the users and what permissions should they have for reading or writing data in the database?
- Who should have load data permissions?
- Who should have permission to execute calculations?
- Which users can be grouped and assigned similar permissions?

See Manage Users and Roles.

Creating Database Models

Create a model of the database. To build the model, identify the perspectives and views that are important to your business. These views translate into the dimensions of the database model.

Many businesses analyze the following views:

- Time periods
- Measures
- Scenarios
- Products
- Customers
- Geographical regions
- Business units

Use the following topics to help you gather information and make decisions.

Identifying Analysis Objectives

After you identify the major views of information in a business, the next step in designing an Essbase database is deciding how the database enables data analysis.

- If analyzing by time, which time periods are needed? Should the analysis include only the current year or multiple years? Should the analysis include quarterly and monthly data? Should it include data by season?
- If analyzing by geographical region, how do you define the regions? Do you define regions by sales territories? Do you define regions by geographical boundaries, such as states and cities?
- If analyzing by product line, should you review data for each product? Can you summarize data into product classes?

Regardless of the business views, you must determine the perspective and detail needed in the analysis. Each business area that you analyze provides a different view of the data.

Determining Dimensions and Members

You can represent each business view as a separate standard dimension in the database. You may hear business analysts refer to the "bys" of their business, such as by product, by geography, and by time period. If you need to analyze a business view by classification or attribute, such as by the size or color of products, you can use attribute dimensions or properties to represent the classification views.

The dimensions that you choose determine what types of analysis you can perform on the data. With Essbase, you can use as many dimensions as you need for analysis.

When you know approximately what dimensions and members you need, develop a tentative database design.

After you determine the dimensions of the database model, choose the elements or items within each dimension. These elements become the hierarchies and members of their respective dimensions. For example, a time hierarchy may include the time periods that you want to analyze, such as quarters, and within quarters, months. Each quarter and month becomes a member of the dimension that you create for time. Quarters and months represent a two-level hierarchy of members and their children. Months within a quarter can consolidate to a total for each quarter.

Relationships Among Dimensions

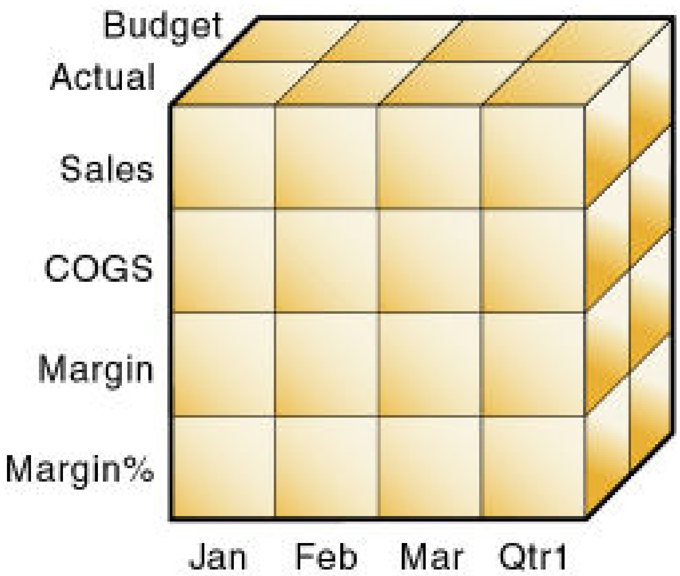
Consider the relationships among the dimensions. The structure of an Essbase database makes it easy for users to analyze information from many perspectives. A financial analyst, for example, may ask the following questions:

- What are sales for a particular month? How does this figure compare to sales in the same month over the last five years?
- By what percentage is profit margin increasing?
- How close are actual values to budgeted values?

In other words, the analyst may want to examine information from three dimensions—time, account, and scenario. The sample database illustrated below represents these three dimensions, with one dimension represented along each of the three axes:

- A time dimension, which comprises Jan, Feb, Mar, and the total for Qtr1, is displayed along the X-axis.
- An accounts dimension, which consists of accounting figures such as Sales, COGS, Margin, and Margin%, is displayed along the Y-axis.
- Another dimension, which provides a different point of view, such as Budget for budget values and Actual for actual values, is displayed along the Z-axis.

Figure 1-1 Cube Representing Three Database Dimensions



The cells within the cube, where the members intersect, contain the data relevant to all three intersecting members; for example, the actual sales in January.

Example Dimension-Member Structure

The table below shows a summary of the TBC dimensions. The application designer created three columns, with the dimensions in the left column and members in the two right columns. The members in column 3 are subcategories of the members in column 2. In some cases, members in column 3 are divided into another level of subcategories; for example, the Margin of the Measures dimension is divided into Sales and COGS.

Table 1-1 TBC Sample Dimensions

Dimensions	Members	Child Members
Year	Qtr1	Jan, Feb, Mar
Year	Qtr2	Apr, May, Jun
Year	Qtr3	Jul, Aug, Sep
Year	Qtr4	Oct, Nov, Dec
Measures	Profit	Margin: Sales, COGS Total Expenses: Marketing, Payroll, Miscellaneous
Measures	Inventory	Opening Inventory, Additions, Ending Inventory
Measures	Ratios	Margin %, Profit %, Profit per Ounce
Product	Colas (100)	Cola (100-10), Diet Cola (100-20), Caffeine Free Cola (100-30)
Product	Root Beer (200)	Old Fashioned (200-10), Diet Root Beer (200-20), Sarsaparilla (200-30), Birch Beer (200-40)

Table 1-1 (Cont.) TBC Sample Dimensions

Dimensions	Members	Child Members
Product	Cream Soda (300)	Dark Cream (300-10), Vanilla Cream (300-20), Diet Cream Soda (300-30)
Product	Fruit Soda (400)	Grape (400-10), Orange (400-20), Strawberry (400-30)
Market	East	Connecticut, Florida, Massachusetts, New Hampshire, New York
Market	West	California, Nevada, Oregon, Utah, Washington
Market	South	Louisiana, New Mexico, Oklahoma, Texas
Market	Central	Colorado, Illinois, Iowa, Missouri, Ohio, Wisconsin
Scenario	Actual	N/A
Scenario	Budget	N/A
Scenario	Variance	N/A
Scenario	Variance %	N/A

In addition, the application designer added the following attribute dimensions to enable product analysis based on size and packaging:

Table 1-2 TBC Sample Attribute Dimensions

Dimensions	Members	Child Members
Ounces	Large Small	64, 32, 20 16, 12
Pkg Type	Bottle Can	N/A

Checklist for Determining Dimensions and Members

Use the following checklist when determining the dimensions and members of your model database:

- What are the candidates for dimensions?
- Do any of the dimensions classify or describe other dimensions? These dimensions are candidates for attribute dimensions.
- Do users want to qualify their view of a dimension? The categories by which they qualify a dimension are candidates for attribute dimensions.
- What are the candidates for members?
- How many levels does the data require?
- How does the data consolidate?

Analyzing Database Design

While the initial dimension design is still on paper, you should review the design according to a set of guidelines. The guidelines help you fine-tune the database and leverage the multidimensional technology. The guidelines are processes or questions that help you achieve an efficient design and meet consolidation and calculation goals.

The number of members needed to describe a potential data point should determine the number of dimensions. If you are not sure whether you should delete a dimension, keep it and apply more analysis rules until you feel confident about deleting or keeping it.

Dense and Sparse Dimensions

Which dimensions are sparse, and which are dense, affects performance. See:

- Sparse and Dense Dimensions
- Designing an Outline to Optimize Performance

Standard and Attribute Dimensions

For simplicity, the examples in this topic show alternative arrangements for what was initially designed as two dimensions. You can apply the same logic to all combinations of dimensions.

Consider the design for a company that sells products to multiple customers over multiple markets; the markets are unique to each customer:

```
             Cust A  Cust B  Cust C
New York     100     N/A     N/A
Illinois     N/A     150     N/A
California   N/A     N/A     30
```

Cust A is only in New York, Cust B is only in Illinois, and Cust C is only in California. The company can define the data in one standard dimension:

```
Market
  New York
    Cust A
  Illinois
    Cust B
  California
    Cust C
```

However, if you look at a larger sampling of data, you may see that many customers can be in each market. Cust A and Cust E are in New York; Cust B, Cust M, and Cust P are in Illinois; Cust C and Cust F are in California. In this situation, the company typically defines the large dimension, Customer, as a standard dimension and the smaller dimension, Market, as an attribute dimension. The company associates the members of the Market dimension as attributes of the members of the Customer dimension. The members of the Market dimension describe locations of the customers — each customer has exactly one market.

```
Customer (Standard dimension)
  Cust A (Attribute:New York)
  Cust B (Attribute:Illinois)
  Cust C (Attribute:California)
  Cust E (Attribute:New York)
```

```
  Cust F (Attribute:California)
  Cust M (Attribute:Illinois)
  Cust P (Attribute:Illinois)
Market (Attribute dimension)
  New York
  Illinois
  California
```

Consider another situation. Again, the company sells products to multiple customers over multiple markets, but the company can sell to a customer that has locations in different markets:

```
             Cust A  Cust B  Cust C
New York     100      75     N/A
Illinois     N/A     150     N/A
California   150     N/A      30
```

Cust A is in New York and California. Cust B is in New York and Illinois. Cust C is only in California. Using an attribute dimension does not work in this situation; a customer member cannot have multiple attribute members. Therefore, the company designs the data in two standard dimensions:

```
Customer
  Cust A
  Cust B
  Cust C
Market
  New York
  Illinois
  California
```

Dimension Combinations

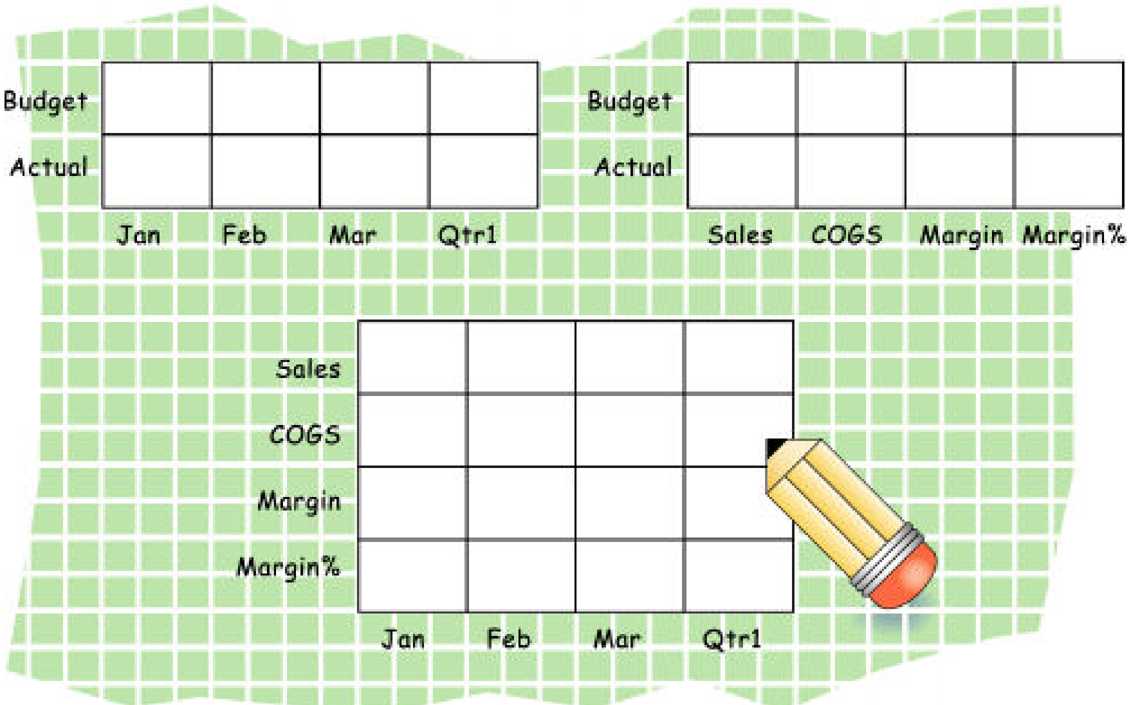
Break each combination of two dimensions into a two-dimensional matrix. For example, proposed dimensions at TBC include the following combinations:

- Year across Measures
- Year across Product
- Year across Market
- Year across Scenario
- Measures across Product
- Measures across Market
- Measures across Scenario
- Market across Product
- Market across Scenario
- Scenario across Product
- Ounces across Pkg Type

Ounces and Pkg Type, as attribute dimensions associated with the Product dimension, can be considered with the Product dimension.

To help visualize each dimension, draw a matrix and include a few of the first-generation members. The following image shows a simplified set of matrices for three dimensions.

Figure 1-2 Analyzing Dimensional Relationships



For each combination of dimensions, ask three questions:

- Does it add analytic value?
- Does it add utility for reporting?
- Does it avoid an excess of unused combinations?

For each combination, the answers to the questions help determine whether the combination is valid for the database. Ideally, the answer to each question is yes. If not, consider rearranging the data into more-meaningful dimensions. As you work through this process, discuss information needs with users.

Repetition in Outlines

The repetition of elements in an outline often indicates a need to split dimensions. The following examples show you how to avoid repetition.

In this example, the left column, labeled “Repetition,” shows Profit, Margin, Sales, COGS, and Expenses repeated under Budget and Actual in the Accounts dimension. The right column, labeled “No Repetition,” separates Budget and Actual into another dimension (Scenario), leaving just one set of Profit, Margin, Sales, COGS, and Expenses members in the Accounts dimension. This approach simplifies the outline and provides a simpler view of the budget and actual figures of the other dimensions in the database.

Figure 1-3 Example of Eliminating Repetition By Creating a Scenario Dimension

Repetition	No Repetition
Accounts	Accounts
Budget	Profit
Profit	Margin
Margin	Sales
Sales	COGS
COGS	Expenses
Expenses	Scenario
Actual	Budget
Profit	Actual
Margin	
Sales	
COGS	
Expenses	

In this example, the left column, labeled "Repetition," uses shared members in the Diet dimension to analyze diet beverages. Members 100–20, 200–20, and 300–20 are repeated: once under Diet, and once under their respective parents. The right column, labeled "No Repetition," simplifies the outline by creating a Diet attribute dimension of type Boolean (True or False). All members are shown only once, under their respective parents, and are tagged with the appropriate attribute ("Diet: True" or "Diet: False").

Figure 1-4 Example of Eliminating Repetition By Creating an Attribute Dimension

Repetition	No Repetition
Product	Product
100 (Alias: Colas)	100 (Alias: Colas)
100-10 (Alias: Cola)	100-10 (Alias: Cola) (Diet: False)
100-20 (Alias: Diet Cola)	100-20 (Alias: Diet Cola) (Diet: True)
200 (Alias: Root Beer)	200 (Alias: Root Beer)
200-20 (Alias: Diet Root Beer)	200-20 (Alias: Diet Root Beer) (Diet: True)
200-30 (Alias: Birch Beer)	200-30 (Alias: Birch Beer) (Diet: False)
300 (Alias: Cream Soda)	300 (Alias: Cream Soda)
300-10 (Alias: Dark Cream)	300-10 (Alias: Dark Cream) (Diet: False)
300-20 (Alias: Diet Cream)	300-20 (Alias: Diet Cream) (Diet: True)
Diet (Alias: Diet Drinks)	Diet Attribute (Type: Boolean)
100-20 (Alias: Diet Cola)	True
200-20 (Alias: Diet Root Beer)	False
300-20 (Alias: Diet Cream)	

Attribute dimensions also provide additional analytic capabilities. See Designing Attribute Dimensions.

Interdimensional Irrelevance

Interdimensional irrelevance occurs when many members of a dimension are irrelevant across other dimensions. Essbase defines irrelevant data as data that Essbase stores only at the summary (dimension) level. In such a situation, you may be able to remove a dimension from the database and add its members to another dimension or split the model into separate databases.

For example, TBC considered analyzing salaries as a member of the Measures dimension. But salary information often proves irrelevant in the context of a corporate database. Most salaries are confidential and apply to individuals. The individual and the salary typically represent one cell, with no reason to intersect with any other dimension.

TBC considered separating employees into a separate dimension. The following table shows an example of how TBC analyzed the proposed Employee dimension for interdimensional irrelevance. Members of the proposed Employee dimension (represented in the table header row) are compared with members of the Measures dimension (represented in the left-most column). The Measures dimension members (such as Revenue) apply to All Employees; only the Salary measure is relevant to individual employees.

Table 1-3 Example of Interdimensional Irrelevance

	Joe Smith	Mary Jones	Mike Garcia	All Employees
Revenue	Irrelevance	Irrelevance	Irrelevance	Relevance
Variable Costs	Irrelevance	Irrelevance	Irrelevance	Relevance
COGS	Irrelevance	Irrelevance	Irrelevance	Relevance
Advertising	Irrelevance	Irrelevance	Irrelevance	Relevance
Salaries	Relevance	Relevance	Relevance	Relevance
Fixed Costs	Irrelevance	Irrelevance	Irrelevance	Relevance
Expenses	Irrelevance	Irrelevance	Irrelevance	Relevance
Profit	Irrelevance	Irrelevance	Irrelevance	Relevance

Reasons to Split Databases

Because individual employee information is irrelevant to the other information in the database, and also because adding an Employee dimension would substantially increase database storage needs, TBC created a separate Human Resources (HR) database. The new HR database contains a group of related dimensions and includes salaries, benefits, insurance, and 401(k) plans.

There are many reasons for splitting a database; for example, suppose that a company maintains an organizational database that contains several international subsidiaries in several time zones. Each subsidiary relies on time-sensitive financial calculations. You can split the database for groups of subsidiaries in the same time zone to ensure that financial calculations are timely. You can also use a partitioned application to separate information by subsidiary.

Checklist to Analyze the Database Design

Use the following checklist to analyze the database design:

- Have you minimized the number of dimensions?
- For each dimensional combination, did you ask:
  - Does it add analytic value?
  - Does it add utility for reporting?
  - Does it avoid an excess of unused combinations?
- Did you avoid repetition in the outline?
- Did you avoid interdimensional irrelevance?
- Did you split the databases as necessary?

Drafting Outlines

Now you can create the application and database and build the first draft of the outline in Essbase. The draft defines all dimensions, members, and consolidations. Use the outline to design consolidation requirements and identify where you need formulas and calculation scripts.

> **Note:**
>
> Outlines are a part of an Essbase database (or cube), which exists inside an Essbase application.

The TBC application designer issued the following draft for a database outline. In this plan, Year, Measures, Product, Market, Scenario, Pkg Type, and Ounces are dimension names. Observe how TBC anticipated consolidations, calculations and formulas, and reporting requirements. The application designers also used product codes rather than product names to describe products.

- **Year.** TBC needs to collect data monthly and summarize the monthly data by quarter and year. Monthly data, stored in members such as Jan, Feb, and Mar, consolidates to quarters. Quarterly data, stored in members such as Qtr1 and Qtr2, consolidates to Year.
- **Measures.** Sales, Cost of Goods Sold, Marketing, Payroll, Miscellaneous, Opening Inventory, Additions, and Ending Inventory are standard measures. Essbase can calculate Margin, Total Expenses, Profit, Total Inventory, Profit %, Margin %, and Profit per Ounce from these measures. TBC needs to calculate Measures on a monthly, quarterly, and yearly basis.
- **Product.** The Product codes are 100-10, 100-20, 100-30, 200-10, 200-20, 200-30, 200-40, 300-10, 300-20, 300-30, 400-10, 400-20, and 400-30. Each product consolidates to its respective family (100, 200, 300, and 400). Each consolidation allows TBC to analyze by size and package, because each product is associated with members of the Ounces and Pkg Type attribute dimensions.
- **Market.** Several states make up a region; four regions make up a market. The states are Connecticut, Florida, Massachusetts, New Hampshire, New York, California, Nevada, Oregon, Utah, Washington, Louisiana, New Mexico, Oklahoma, Texas, Colorado, Illinois, Iowa, Missouri, Ohio, and Wisconsin. Each state consolidates into its region—East, West, South, or Central. Each region consolidates into Market.
- **Scenario.** TBC derives and tracks budget versus actual data. Managers must monitor and track budgets and actuals, as well as the variance and variance percentage between them.
- **Pkg Type.** TBC wants to see the effect that product packaging has on sales and profit. Establishing the Pkg Type attribute dimension enables users to analyze product information based on whether a product is packaged in bottles or cans.
- **Ounces.** TBC sells products in different sizes in ounces in different markets. Establishing the Ounces attribute dimension helps users monitor which sizes sell better in which markets.

The following topics present a review of the basics of dimension and member properties and a discussion of how outline design affects performance.

Dimension and Member Properties

The properties of dimensions and members define the roles of the dimensions and members in the design of the multidimensional structure. These properties include the following:

- Dimension types and attribute associations. See Dimension Types.
- Data storage properties. See Member Storage Properties.
- Consolidation operators. See Consolidation of Dimensions and Members.
- Formulas. See Formulas and Functions.

For a complete list of dimension and member properties, see Setting Dimension and Member Properties.

Dimension Types

A dimension type is a property that Essbase provides that adds special functionality to a dimension. The most commonly used dimension types: time, accounts, and attribute. This topic uses the following dimensions of the TBC database to illustrate dimension types.

```
Database:Design
  Year (Type: time)
  Measures (Type: accounts)
  Product
  Market
  Scenario
  Pkg Type (Type: attribute)
  Ounces (Type: attribute)
```

The following table defines each Essbase dimension type.

Table 1-4 Dimension Types

Dimension Types	Description
None	Specifies no particular dimension type.
Time	Defines the time periods for which you report and update data. You can tag only one dimension as time. The time dimension enables several accounts dimension functions, such as first and last time balances.
Accounts	Contains items that you want to measure, such as profit and inventory, and makes Essbase built-in accounting functionality available. Only one dimension can be defined as accounts.
Attribute	Contains members that can be used to describe members of another, so-called base dimension. For example, the Pkg Type attribute dimension contains a member for each type of packaging, such as bottle or can, that applies to members of the Product dimension.

Member Storage Properties

You can specify data storage properties for members; data storage properties define where and when consolidations are stored. For example, by default, members are tagged as store data. Essbase sums the values of store data members and stores the result at the parent level.

You can change the default logic for each member by changing the data storage property tag for the member. For example, you can change a store data member to a label only member. Members with the label only tag, for example, do not have data associated with them.

The following table describes the effect that Essbase data storage properties have on members.

Table 1-5 Essbase Data Storage Properties

Data Storage Properties	Effects on Members
Store data	Data for the member is stored in the database. Store data is the default storage property.
Dynamic Calc	The data associated with the member is calculated when requested by a user query. The calculated data is not stored; it is discarded after the query request is completed.
Shared member	The data associated with the member comes from another member with the same name.
Label only	Although a label only member has no data associated with it, a label only member can display a value. The label only tag groups members and eases navigation and reporting. Typically, label only members are not calculated. For example, in the Measures dimension, the member Ratios has three children, Margin%, Profit%, and Profit per Ounce. The member Ratios defines a category of members. When consolidated, Margin%, Profit%, and Profit per Ounce do not roll up to a meaningful figure for Ratios. Hence, Ratios is tagged as label only.

Checklist for Dimension and Member Properties

- Can you identify a time dimension?
- Can you identify an accounts dimension?
- Can you identify qualities or characteristics of dimensions that should be defined as separate attribute dimensions?
- Which members require special data storage properties?

Designing an Outline to Optimize Performance

Position attribute dimensions at the end of the outline. Position dense dimensions before sparse dimensions.

The position of dimensions in an outline and the storage properties of dimensions can affect two areas of performance—how quickly calculations are run and how long it takes users to retrieve information.

See these topics to understand performance optimization basics:

- Optimizing Query Performance
- Optimizing Calculation Performance
- Meeting the Needs of Both Calculation and Retrieval

Optimizing Query Performance

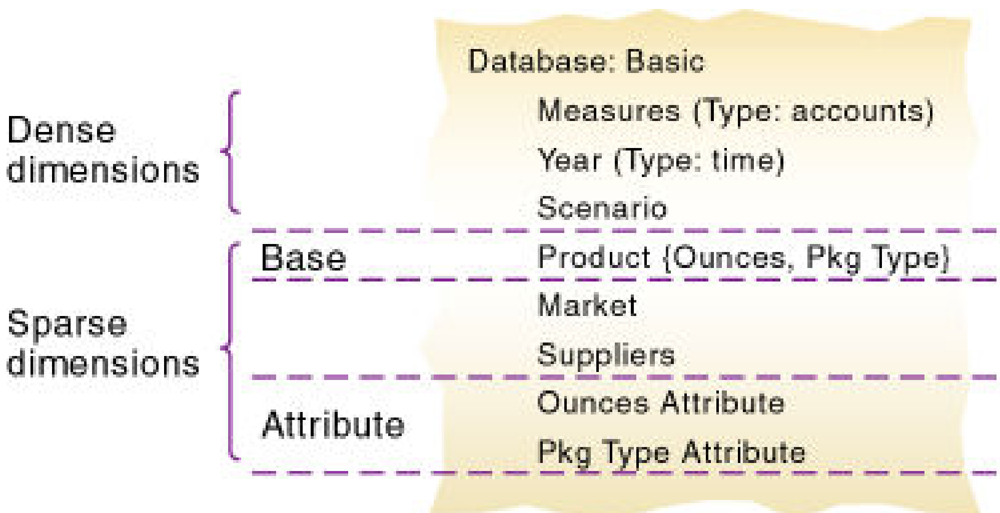
To optimize query performance, use the following guidelines when you design an outline:

- If the outline contains attribute dimensions, ensure that the attribute dimensions are the only sparse Dynamic Calc dimensions in the outline.
- In the outline, place the more-queried sparse dimensions before the less-queried sparse dimensions.

The outline illustrated below is designed for optimum query performance:

- Because the outline contains attribute dimensions, the storage property for standard dimensions and all standard dimensions members is set as store data.
- As the most-queried sparse dimension, the Product dimension is the first of the sparse dimensions. Base dimensions are typically queried more than other dimensions.

Figure 1-5 Designing an Outline for Optimized Query Times



Optimizing Calculation Performance

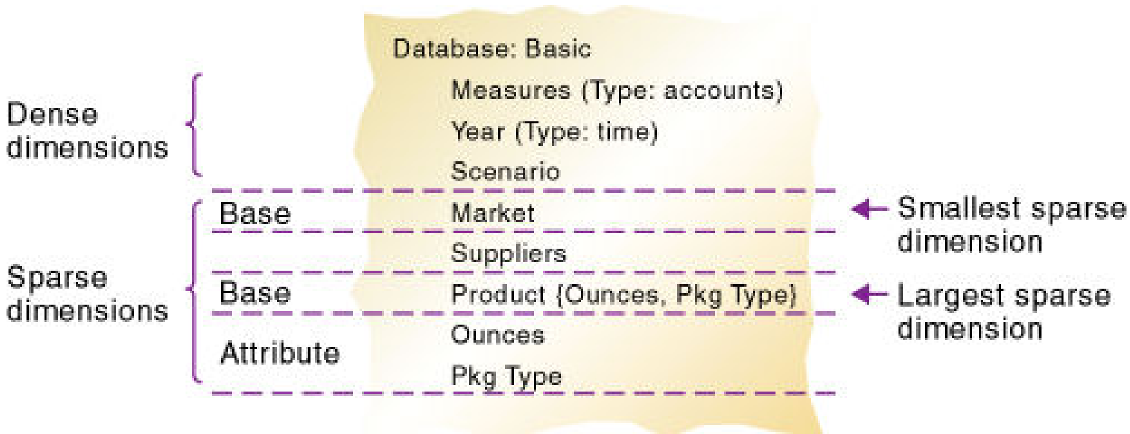
To optimize calculation performance, order the sparse dimensions in the outline by their number of members, starting with the dimension that contains the fewest.

See Design for Calculation Performance.

The outline illustrated below is designed for optimum calculation performance:

- The smallest standard dimension that is sparse, Market, is the first of the sparse dimensions in the outline.
- The largest standard dimension that is sparse, Product, is immediately above the first attribute dimension. If the outline did not contain attribute dimensions, the Product dimension would be at the end of the outline.

Figure 1-6 Designing an Outline for Optimized Calculation Times



Meeting the Needs of Both Calculation and Retrieval

Although they contain the same dimensions, the example outlines shown previously are different. To determine the best outline sequence for a situation, prioritize the data retrieval requirements of the users against the time needed to run calculations on the database. How often do you expect to update and recalculate the database? What is the nature of user queries? What is the expected volume of user queries?

A possible workaround is to initially position the dimensions in the outline to optimize calculation. After you run the calculations, you can manually resequence the dimensions to optimize retrieval. When you save the outline after you reposition its dimensions, choose to restructure the database by index only. Before you run calculations again, resequence the dimensions in the outline to optimize calculation.

Loading Test Data

Before you can test calculations, consolidations, and reports, you need data in the database. During the design process, loading mocked-up data or a subset of real data provides flexibility and shortens the time required to test and analyze results.

See the following topics for detailed instructions for loading data:

- Overview of Data Load and Dimension Build
- Design Rules for Data Load or Dimension Build

If you're satisfied with your database design after the preliminary test, load the complete set of real data with which you'll populate the final database. Use the test rules files, if possible. This final test may reveal source data problems that were not anticipated during earlier design phases.

Defining Calculations

Calculations are essential to derive certain types of data. Data that is derived from a calculation is called calculated data; source data that is not calculated is called input data.

The following topics use the Product and Measures dimensions of the TBC application to illustrate several types of common calculations that are found in many Essbase databases.

- Consolidation of Dimensions and Members
- Tags and Operators on Example Measures Dimension
- Accounts Dimension Calculations
- Formulas and Functions
- Dynamic Calculations
- Two-Pass Calculations
- Checklist for Calculations

Consolidation of Dimensions and Members

When you define members of standard dimensions, Essbase automatically tags the members with the + consolidation operator (plus sign representing addition), meaning that during consolidation members are added to derive their parent's value. As appropriate, you can change a member consolidation property.

Consolidation is the most frequently used calculation in Essbase. This topic uses the Product dimension to illustrate consolidations.

The TBC application has several consolidation paths:

- Individual products roll up to product families, and product families consolidate into Product. The TBC outline also requires multiple consolidation paths; some products must consolidate in multiple categories.
- States roll up to regions, and regions consolidate into Market.
- Months roll up into quarters, and quarters consolidate into Year.

The following topics discuss consolidation in greater detail:

- Effect of Position and Operator on Consolidation
- Consolidation of Shared Members
- Checklist for Consolidation

Consolidation operators define how Essbase rolls up data for each member in a branch to the parent. For example, using the default addition (+) operator, Essbase adds 100-10, 100-20, and 100-30 and stores the result in their parent, 100, as shown below.

Figure 1-7 TBC Product Dimension

```
Product
    100 (+) (Alias: Colas)
        100-10 (+) (Alias: Cola)
        100-20 (+) (Alias: Diet Cola)
        100-30 (+) (Alias: Caffeine Free Cola)
    200 (+) (Alias: Root Beer)
        200-10 (+) (Alias: Old Fashioned)
        200-20 (+) (Alias: Diet Root Beer)
        200-30 (+) (Alias: Sasparilla)
        200-40 (+) (Alias: Birch Beer)
    300 (+) (Alias: Cream Soda)
        300-10 (+) (Alias: Dark Cream)
        300-20 (+) (Alias: Vanilla Cream)
        300-30 (+) (Alias: Diet Cream)
    400 (+) (Alias: Fruit Soda)
        400-10 (+) (Alias: Grape)
        400-20 (+) (Alias: Orange)
        400-30 (+) (Alias: Strawberry)
    Diet (~) (Alias: Diet Drinks)
        100-20 (+) (Shared Member)
        200-20 (+) (Shared Member)
        300-30 (+) (Shared Member)
```

The Product dimension contains mostly addition (+) operators, which indicate that each group of members is added and rolled up to the parent. Diet has a tilde (~) operator, which indicates that Essbase does not include the Diet member in the consolidation to the parent, Product. The Diet member consists entirely of members that are shared. The TBC product management group wants to be able to isolate Diet drinks in reports, so TBC created a separate Diet member that does not impact overall consolidation.

Effect of Position and Operator on Consolidation

Essbase calculates the data of a branch in top-down order. For example, if you have, in order, two members tagged with an addition (+) operator and a third member tagged with a multiplication (*) operator, Essbase adds the first two and multiplies that sum by the third.

Because Essbase always begins with the top member when it consolidates, the order and the labels of the members is important. See Examples of Member Consolidation.

Consolidation of Shared Members

Shared members also affect consolidation paths. The shared member concept enables two members with the same name to share the same data. The shared member stores a pointer to data contained in the other member, so Essbase stores the data only once. Shared members must be in the same dimension. Data can be shared by multiple members.

Checklist for Consolidation

Use the following checklist to help define consolidation:

- Did you identify the consolidations in the outline?
- Did you tag each member with the proper consolidation operator?

- Did you specify a shared member tag for designated members?
- Would shared members be more efficient if designed within an attribute dimension (other than shared)?

Tags and Operators on Example Measures Dimension

The Measures dimension is the most complex dimension in the TBC outline because it uses both time and accounts data. It also contains formulas and special tags to help Essbase calculate the outline. This topic discusses the formulas and tags that TBC included in the Measures dimension (the dimension tagged as accounts).

Examine the Measures dimension tags defined by TBC. Many of the properties of the Measures dimension are discussed in previous topics: addition (+), subtraction (–), and no consolidation (~) operators, and accounts and label only tags:

- The Inventory and Ratios member names assist the user in data navigation. They do not contain data, and therefore receive a label only tag.
- The Measures dimension itself has a label only tag. Some members of Measures have a Dynamic Calc tag.
- Some members of Measures have a time balance tag (TB First or TB Last).

Figure 1-8 TBC Measures Dimension

```
Measures Accounts (Label Only)
    Profit (+) (Dynamic Calc)
        Margin (+) (Dynamic Calc)
            Sales (+)
            COGS (-) (Expense Reporting)
        Total Expenses (-) (Dynamic Calc) (Expense Reporting)
            Marketing (+) (Expense Reporting)
            Payroll (+) (Expense Reporting)
            Misc (+) (Expense Reporting)
    Inventory (~) (Label Only)
        Opening Inventory (+) (TB First) (Expense Reporting)
        Additions (~) (Expense Reporting)
        Ending Inventory (~) (TB Last) (Expense Reporting)
    Ratios (~) (Label Only)
        Margin % (+) (Dynamic Calc) (Two Pass Calc) Margin % Sales;
        Profit % (~) (Dynamic Calc) (Two Pass Calc) Profit % Sales;
        Profit per Ounce (~) Profit/@ATTRIBUTEVAL(Ounces);
```

Accounts Dimension Calculations

This topic discusses two forms of calculations for a dimension tagged as accounts.

- Time Balance Properties
- Variance Reporting

Time Balance Properties

Time balance tags or properties, provide instructions to Essbase about how to calculate the data in a dimension tagged as accounts. Using the tags requires a dimension tagged as accounts and a dimension tagged as time. The first, last, average, and expense tags are available exclusively for use with accounts dimension members.

In the TBC Measures dimension, Opening Inventory data represents the inventory that TBC carries at the beginning of each month. The quarterly value for Opening Inventory equals the Opening value for the first month in the quarter. Opening Inventory requires the time balance tag, TB first.

Ending Inventory data represents the inventory that TBC carries at the end of each month. The quarterly value for Ending Inventory equals the ending value for the last month in the quarter. Ending Inventory requires the time balance tag, TB last. The following table defines the time balance tags for the accounts dimension.

Table 1-6 Accounts Member Tags

Tags	Description
Time Balance Last	The value for the last child member is carried to the parent. For example, March is carried to Qtr1.
Time Balance First	The value for the first child is carried to the parent. For example, Jan is carried to Qtr1.

In the following table, Qtr1 (second column from the right) and Year (right-most column) show how consolidation in the time dimension is affected by time balance properties in the accounts dimension. Data is shown for the first quarter only.

Table 1-7 TBC Consolidations Affected by Time Balance Properties

Dimensions	Jan	Feb	Mar	Qtr1	Year
Accounts Member1	11	12	13	*36*	*Qtr1 + Qtr2 + Qtr3 + Qtr4*
Accounts Member2 (TB First)	20	25	21	*20*	*20*
Accounts Member3 (TB Last)	25	21	30	*30*	*Value of Qtr4*

Normally, the calculation of a parent in the time dimension is based on the consolidation and formulas of children of the parent. However, if a member in an accounts branch is marked as TB First, any parent in the time dimension matches the member marked as TB First.

For examples, see Setting Time Balance Properties.

Variance Reporting

One TBC Essbase requirement is the ability to perform variance reporting on actual versus budget data. The variance reporting calculation requires that any item that represents an expense to the company must have an expense reporting tag. Inventory members, Total Expense members, and the COGS member each receive an expense reporting tag for variance reporting.

Essbase provides two variance reporting properties—expense and nonexpense (default). Variance reporting properties define how Essbase calculates the difference between actual and budget data in members with the @VAR or @VARPER function in their member formulas.

When you tag a member as expense, the @VAR function calculates Budget – Actual. For example, if the budgeted amount is $100 and the actual amount is $110, the variance is –10.

Without the expense reporting tag, the @VAR function calculates Actual – Budget. For example, if the budgeted amount is $100 and the actual amount is $110, the variance is 10.

Formulas and Functions

Formulas calculate relationships between members in the database outline. You can apply formulas to members in the outline, or you can place formulas in a calculation script. This topic explains how TBC optimized the performance of its database by using formulas.

Functions are predefined routines that perform specialized calculations and return sets of members or sets of data values. Formulas comprise operators and functions, as well as dimension names, member names, and numeric constants.

Essbase supports the following operators:

- Mathematical operators, which perform arithmetic operations
- Conditional operators, which build logical conditions into calculations
- Cross-dimensional operators, which point to data values of specific database member combinations

The Essbase functions include more than 175 predefined routines to extend the calculation capabilities of Essbase. Essbase includes the following functions:

- Boolean functions, which provide a conditional test by returning a TRUE or FALSE value
- Mathematical functions, which perform specialized mathematical calculations
- Relationship functions, which look up data values within a database during a calculation based on the position of the current member
- Range functions, which declare a range of members as an argument to another function or to a command
- Financial functions, which perform specialized financial calculations
- Member set functions, which are based on a specified member and which generate lists of members
- Allocation functions, which allocate values that are input at a parent level across child members
- Forecasting functions, which manipulate data for the purpose of smoothing data, interpolating data, or calculating future values
- Statistical functions, which calculate advanced statistics
- Date and time functions, which use date and time characteristics in calculation formulas
- Calculation mode functions, which specify the calculation mode that Essbase uses to calculate a formula

The Measures dimension uses the following formulas:

- Margin = Sales – COGS
- Total Expenses = Marketing + Payroll + Miscellaneous
- Profit = Margin – Total Expenses
- Profit % = Profit % Sales
- Margin % = Margin % Sales
- Profit per Ounce = Profit / @ATTRIBUTEVAL(@NAME(Ounces))

Essbase uses consolidation operators to calculate the Margin, Total Expenses, and Profit members. The Margin% formula uses a % operator, which means “express Margin as a percentage of Sales.” The Profit% formula uses the same % operator. The Profit per Ounce formula uses a division operator (/) and a function (@ATTRIBUTEVAL) to calculate profitability by ounce for products sized in ounces.

> **Note:**
>
> In the Profit per Ounce formula, the @NAME function is also used to process the string “Ounces” for the @ATTRIBUTEVAL function.

For a complete list of operators, functions, and syntax, see Calculation Function List. Also see Develop Formulas for Block Storage Cubes.

Dynamic Calculations

When you design the overall database calculation, you may want to define a member as a Dynamic Calc member. When you tag a member as Dynamic Calc, Essbase calculates the combinations of that member when you retrieve the data, instead of precalculating the member combinations during the regular database calculation. Dynamic calculations shorten regular database calculation time, but may increase retrieval time for dynamically calculated data values.

In the following outline, the TBC Measures dimension contains several members tagged as Dynamic Calc—Profit, Margin, Total Expenses, Margin %, and Profit %.

Figure 1-9   TBC Measures Dimension, Dynamic Calc Tags

```
Measures Accounts (Label Only)
    Profit (+) (Dynamic Calc)
        Margin (+) (Dynamic Calc)
            Sales (+)
            COGS (-) (Expense Reporting)
        Total Expenses (-) (Dynamic Calc) (Expense Reporting)
            Marketing (+) (Expense Reporting)
            Payroll (+) (Expense Reporting)
            Misc (+) (Expense Reporting)
    Inventory (~) (Label Only)
        Opening Inventory (+) (TB First) (Expense Reporting)
        Additions (~) (Expense Reporting)
        Ending Inventory (~) (TB Last) (Expense Reporting)
    Ratios (~) (Label Only)
        Margin % (+) (Dynamic Calc) (Two Pass Calc) Margin % Sales;
        Profit % (~) (Dynamic Calc) (Two Pass Calc) Profit % Sales;
        Profit per Ounce (~) Profit/@ATTRIBUTEVAL(Ounces);
```

When an overall database calculation is performed, the Dynamic Calc members and their corresponding formulas are not calculated. These members are calculated when a user queries them, for example, from Smart View. Essbase does not store the queried values; it recalculates the values for every subsequent query.

To decide when to calculate data values dynamically, consider your priorities in the following areas:

- Optimum regular calculation time (batch calculation)
- Low disk space usage
- Reduced database restructure time
- Speedy data retrieval for users
- Reduced backup time

See Dynamically Calculating Data Values.

Two-Pass Calculations

In the TBC database, Margin % and Profit % contain the label two-pass. This label indicates that some member formulas must be calculated twice to produce the desired value. The two-pass property works only on members of the dimension tagged as accounts and on members tagged as Dynamic Calc.

The following example illustrates why Profit % (based on the formula Profit % Sales) has a two-pass tag. The tables have five columns (column headers are labeled left to right as Dimension, Jan, Feb, Mar, and Qtr1) and three rows (labeled as Profit, Sales, and Profit %). Jan, Feb, Mar, and Qtr1 are members of the Year dimension. Profit, Sales, and Profit % are members of the Measures (accounts) dimension.

The following example defines the initial data to load into Essbase. The data values for Profit -> Jan, Profit -> Feb, and Profit -> Mar are 100. The data value for Sales -> Jan, Sales -> Feb, and Sales -> Mar are 1000.

Table 1-8 Data Loaded into Essbase

Dimension	Jan	Feb	Mar	Qtr1
Profit	100	100	100	N/A
Sales	1000	1000	1000	N/A
Profit %	N/A	N/A	N/A	N/A

First, Essbase calculates the Measures dimension. In the following table, the data values for Profit % -> Jan, Profit % -> Feb, and Profit % -> Mar are 10%.

Table 1-9 Data After Essbase Calculates the Measures Dimension

Dimension	Jan	Feb	Mar	Qtr1
Profit	100	100	100	
Sales	1000	1000	1000	
Profit %	10%	10%	10%	N/A

Next, Essbase calculates the Year dimension. The data rolls up across the dimension. In the following table, the data values for Profit -> Qtr1 (300) and Sales -> Qtr1 (3000) are correct. The data value for Profit % -> Qtr1 (30%) is incorrect because Profit % is tagged as a two-pass calculation.

Table 1-10 Data After Essbase Calculates the Year Dimension

Dimension	Jan	Feb	Mar	Qtr1
Profit	100	100	100	300
Sales	1000	1000	1000	3000
Profit %	10%	10%	10%	30%

Essbase then recalculates profit percentage at each occurrence of the member Profit %. In the following table, the data value for Profit % -> Qtr1 (10%) is correct after the second pass.

Table 1-11 Data After Essbase Recalculates Profit Percentage

Dimension	Jan	Feb	Mar	Qtr1
Profit	100	100	100	300
Sales	1000	1000	1000	3000
Profit %	10%	10%	10%	10%

Checklist for Calculations

Use the following checklist when you define a calculation:

- Does the default calculation logic achieve accurate results?
- Which members require formulas?
- Which members require time balance tags?
- Which members require variance reporting?
- Which members require two-pass calculation?
- Which members can be tagged as Dynamic Calc?

Defining Reports

To ensure that the design meets user information requirements, you must view data as users view it. Users typically view data through spreadsheets, printed reports, or reports published on the Web. Oracle and its partners offer many tools for producing reporting systems for users.

Several tools can help you display and format data quickly, and test whether the database design meets user needs. For example, those who are familiar with spreadsheets can use Smart View.

During the design phase, check the following:

- Grouping and sequencing of data. Do the intersections enabled by the design provide the data that users need?
- Levels of totals. What consolidation levels are required by, for example, a Smart View user who drills down and up through the hierarchy of the outline design?
- Attribute reporting. Does the database design facilitate an analysis that is based on the characteristics or attributes of specific dimensions or members? For example, do you need to compare sales by specific combinations of size and packaging, such as comparing the sales of 16-ounce bottled colas with the sales of 32-ounce bottled colas?

Be sure to use the appropriate tool to create and test predesigned use reports against test data. The reports that you design should provide information that meets your original objectives. The reports should be easy to use, providing the right combinations of data, and the right amount of data. Because reports with too many columns and rows are difficult to use, you may need to create several reports instead of one all-inclusive report, or pivot a dimension to the point of view of the report so that each user can choose a different member from that dimension without viewing all members.

Verifying the Design

After you analyze the data and create a preliminary design, check all aspects of the design with users. You should already have verified that the database satisfies the users' analysis and reporting needs. Ensure that the database satisfies all of their goals.

Do the calculations provide the information they need? Are they able to generate reports quickly? Are they satisfied with consolidation times? In short, ask users if the database works for them.

Near the end of the design cycle, test with real data. Does the outline build correctly? Does all data load? If the database fails in any area, repeat the steps of the design cycle to identify the cause of the problem.

Essbase provides several sources of information to help you isolate problems. Sources include application and Essbase Server logs, and exception logs. Look at documentation topics relevant to your problem; for example, topics about security, calculations, reports, or general error messages.

Most likely, you will need to repeat one or more steps of the design process to arrive at the ideal database solution.

2 Understanding Multidimensional Databases

Online analytical processing (OLAP) is a multidimensional, multiuser, client-server computing environment for users who need to analyze enterprise data. Key features of OLAP applications include multidimensional views of data and calculation-intensive capabilities. Review the topics in this chapter to understand basic concepts of an Essbase database.

- OLAP and Multidimensional Databases
- Dimensions and Members
- Data Storage
- The Essbase Solution for Creating Optimized Databases

Some information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases. Also see Comparison of Aggregate and Block Storage.

OLAP and Multidimensional Databases

Multidimensional views of data sets enable analysis of relationships between data categories, with limitless applications in finance, sales and retail, marketing, manufacturing, CRM, human resources, business operations, IT, and more. The calculation and time intelligence features of Essbase make it an excellent choice for flexible data modeling and analytics.

Finance departments use OLAP for applications such as budgeting, activity-based costing (allocations), financial performance analysis, and financial modeling. Sales departments use OLAP for sales analysis and forecasting. Marketing departments use OLAP for market research analysis, sales forecasting, promotions analysis, customer analysis, and market/customer segmentation. Typical manufacturing OLAP applications include production planning and defect analysis.

Important to all of these applications is the ability to provide managers the information that they need to make effective decisions about an organization's strategic directions. A successful OLAP application provides information as needed; that is, it provides “just-in-time” information for effective decision-making.

Providing such information requires more than a base level of detailed data. Just-in-time information is computed data that usually reflects complex relationships and is often calculated on the fly. Analyzing and modeling complex relationships are practical only if response times are consistently short. In addition, because the nature of data relationships may not be known in advance, the data model must be flexible. A truly flexible data model ensures that OLAP systems can respond to changing business requirements as needed for effective decision making.

Although OLAP applications are found in widely divergent functional areas, all require the following key features:

- Multidimensional views of data
- Calculation-intensive capabilities
- Time intelligence

Key to OLAP systems are multidimensional databases, which not only consolidate and calculate data; but also provide retrieval and calculation of a variety of data subsets. A multidimensional database supports multiple views of data sets for users who need to analyze the relationships between data categories. For example, a marketing analyst might ask following questions:

- How did Product A sell last month? How does this figure compare to sales in the same month over the last five years? How did the product sell by branch, region, and territory?
- Did this product sell better in particular regions? Are there regional trends?
- Did customers return Product A last year? Were the returns due to product defects? Did the company manufacture the products in a specific plant?
- Did commissions and pricing affect how salespeople sold the product? Did certain salespeople sell more?

In multidimensional databases, the number of data views is limited only by the database outline, the structure that defines all elements of the database. Users can pivot the data to see information from a different viewpoint, drill down to find more detailed information, or drill up to see an overview.

Dimensions and Members

This section introduces the concepts of outlines, dimensions, and members within a multidimensional database. If you understand dimensions and members, you are well on your way to understanding the power of a multidimensional database.

A dimension represents the highest consolidation level in the database outline. The database outline presents dimensions and members in a tree structure to indicate a consolidation relationship. For example, Year is a dimension (of type Time) and Qtr1 is a member:

```
Year Time
   Qtr1 (+)
      Jan (+)
......Feb (+)
......Mar (+)
```

There are two types of dimensions: standard and attribute.

- Standard dimensions represent the core components of a business plan and often relate to departmental functions. Typical standard dimensions: Time, Accounts, Product Line, Market, and Division. Dimensions change less frequently than members.
- Attribute dimensions are associated with standard dimensions. Through attribute dimensions, you group and analyze members of standard dimensions based on the member attributes (characteristics). For example, you can compare the profitability of noncaffeinated products that are packaged in glass to the profitability of noncaffeinated products packaged in cans.

Members are the individual components of a dimension. For example, Product A, Product B, and Product C might be members of the Product dimension. Each member has a unique name. Data associated with a member can be stored (referred to as a stored member in this chapter), or the data can be dynamically calculated when a user retrieves it.

Outline Hierarchies

Start designing your Essbase cube by defining an outline. The outline organization reflects the relationship of your data categories as structural and mathematical hierarchies.

Essbase database (cube) development begins with creating an outline, which accomplishes the following objectives:

- Defines structural relationships
- Organizes data
- Defines the consolidations and mathematical relationships between items

The concept of members is used to represent data hierarchies. Each dimension consists of one or more members. The members, in turn, may consist of other members. When you create a dimension, you define how to consolidate the values of its individual members. Within the tree structure of the cube outline, a consolidation is a group of members in a branch of the tree.

For example, many businesses summarize their data monthly, rolling up monthly data to obtain quarterly figures and rolling up quarterly data to obtain annual figures. Businesses may also summarize data by zip code, city, state, and country. Any dimension can be used to consolidate data for reporting purposes.

In the Sample.Basic cube, for example, the Year dimension includes quarter members: Qtr1, Qtr2, Qtr3, and Qtr4, each storing data for an individual quarter, and Year, storing summary data for the year. Qtr1 includes Jan, Feb, and Mar, each storing data for a month, and Qtr1, storing summary data for the quarter. Similarly, Qtr2, Qtr3, and Qtr4 include the members that represent the individual months, and the member that stores the quarterly totals.

The following hierarchical structure represents the data consolidations and relationships in Qtr1.

```
Year Time
   Qtr1 (+)
      Jan (+)
......Feb (+)
......Mar (+)
```

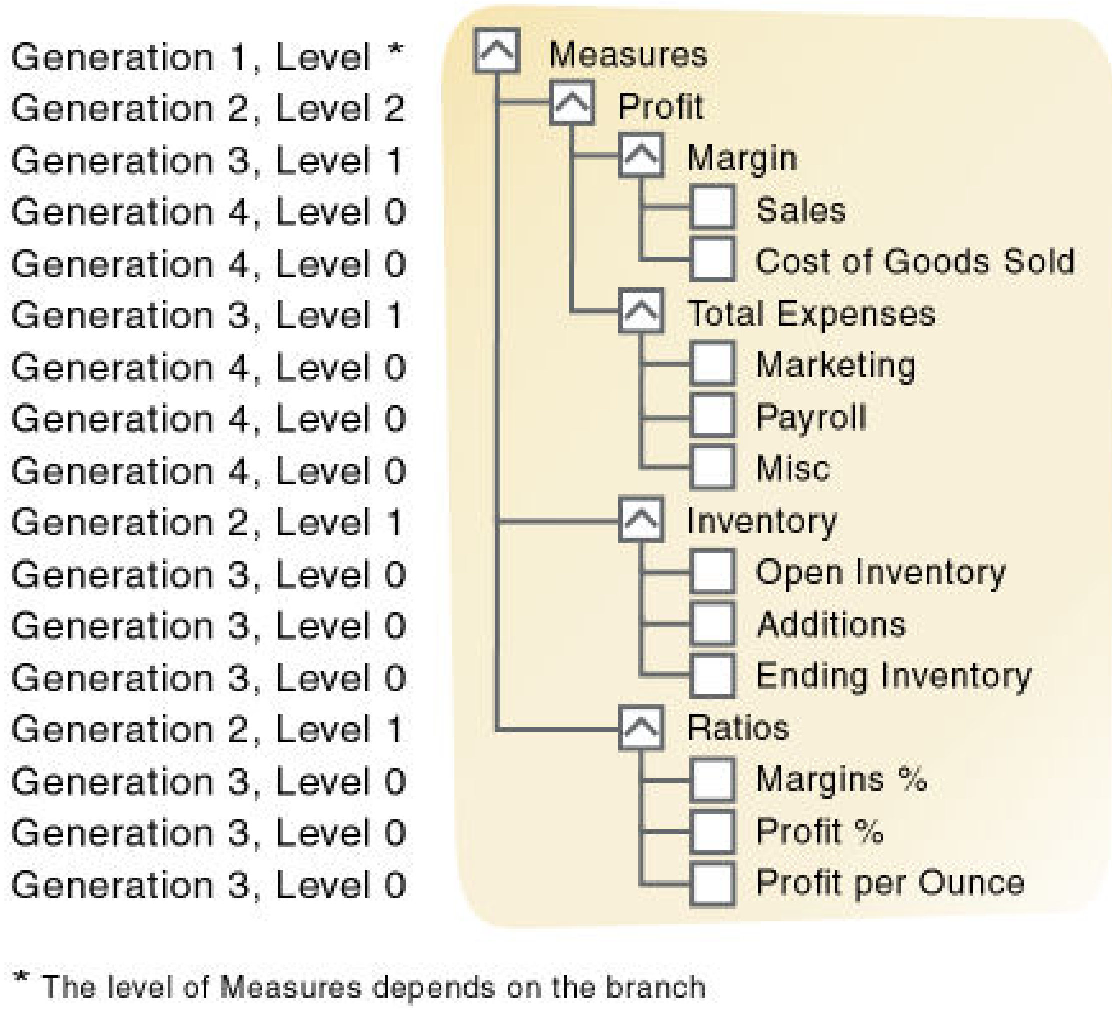
Some dimensions consist of relatively few members, while others may have hundreds or thousands of members.

Dimension and Member Relationships

The organization of an Essbase outline is described using hierarchical and familial terminology, as it's an easy way to conceptualize roles and relationships of members within a dimension. Hierarchical terms include generations and levels, which are analogous to roots and leaves. Familial terms include parents, children, siblings, descendants, and ancestors.

The subtopics in this section reference the outline shown below in describing the position of the members in the outline.

Figure 2-1 Member Generation and Level Numbers



Parents, Children, and Siblings

The outline illustrates the following parent, child, and sibling relationships:

- A parent is a member that has a branch below it. For example, Margin is a parent member for Sales and Cost of Goods Sold.
- A child is a member that has a parent above it. For example, Sales and Cost of Goods Sold are children of the parent Margin.
- Siblings are child members of the same immediate parent, at the same generation. For example, Sales and Cost of Goods Sold are siblings (they both have the parent Margin), but Marketing (at the same branch level) is not a sibling, because its parent is Total Expenses.

Descendants and Ancestors

The outline illustrates the following descendant and ancestral relationships:

- Descendants are members in branches below a parent. For example, Profit, Inventory, and Ratios are descendants of Measures. The children of Profit, Inventory, and Ratios are also descendants of Measures.
- Ancestors are members in branches above a member. For example, Margin, Profit, and Measures are ancestors of Sales.

Roots and Leaves

The outline illustrates the following root and leaf member relationships:

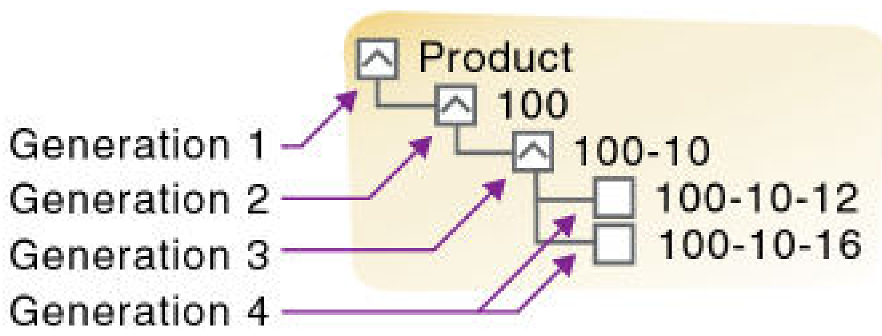
- The root is the top member in a branch. Measures is the root for Profit, Inventory, Ratios.
- Leaf members have no children and are also referred to as level 0 members. For example, Opening Inventory, Additions, and Ending Inventory are level 0 members.

Generations and Levels

Generation refers to a consolidation level within a dimension. A root branch of the tree is generation 1. Generation numbers increase as you count from the root toward the leaf member. In the outline, Measures is generation 1, Profit is generation 2, and Margin is generation 3. All siblings of each level belong to the same generation; for example, both Inventory and Ratios are generation 2.

In the following illustration, part of the Product dimension is shown, with its generations numbered. Product is generation 1, 100 is generation 2, 100-10 is generation 3, and 100-10-12 and 100-10-16 are generation 4.

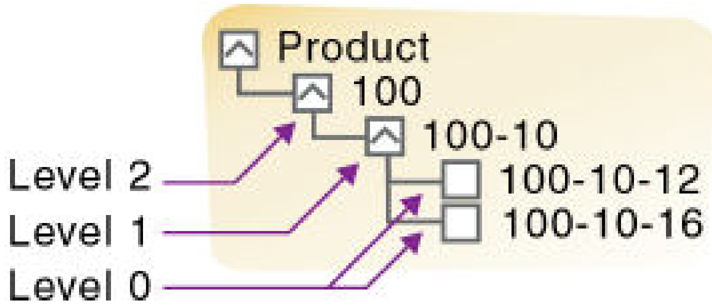
Figure 2-2 Generations



Level also refers to a branch within a dimension; levels reverse the numerical ordering used for generations. Levels count up from the leaf member toward the root. The root level number varies depending on the depth of the branch. In the outline illustration at the beginning of this section, Sales and Cost of Goods Sold are level 0. All other leaf members are also level 0. Margin is level 1, and Profit is level 2. Notice that the level number of Measures varies depending on the branch. For the Ratios branch, Measures is level 2. For the Total Expenses branch, Measures is level 3.

In the following illustration, part of the Product dimension is shown, with its levels numbered. 100 is level 2, 100-10 is level 1, and 100-10-12 and 100-10-16 are level 0.

Figure 2-3 Levels



Generation and Level Names

To ease report maintenance, you can assign a name to a generation or level and then use the name as a shorthand for all members in that generation or level. Because changes to an outline are automatically reflected in a report, when you use generation and level names, you do not need to change the report if a member name is changed or deleted from the database outline.

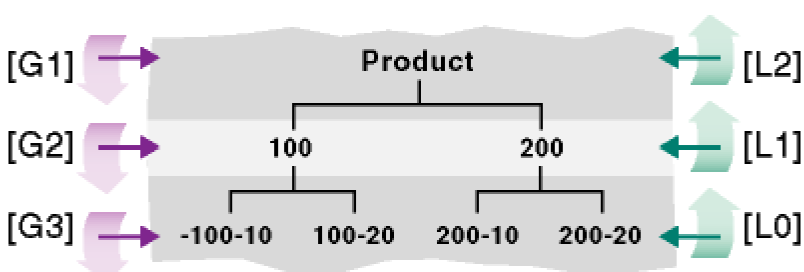
Hierarchy Shapes

Hierarchies in Essbase can be symmetric or asymmetric (ragged).

Essbase handles some operations differently depending on the hierarchy shape. Tabular data export, certain calculation functions (including @ANCESTVAL), and drill through report mapping can have different outcomes when a dimension contains asymmetric hierarchies.

Symmetric Hierarchies

In symmetric hierarchies, members with the same level number are at the same depth in the outline. For example, in the following diagram, members 100-10 and 200-10 are both level 0 members, and they are both generation 3 members:

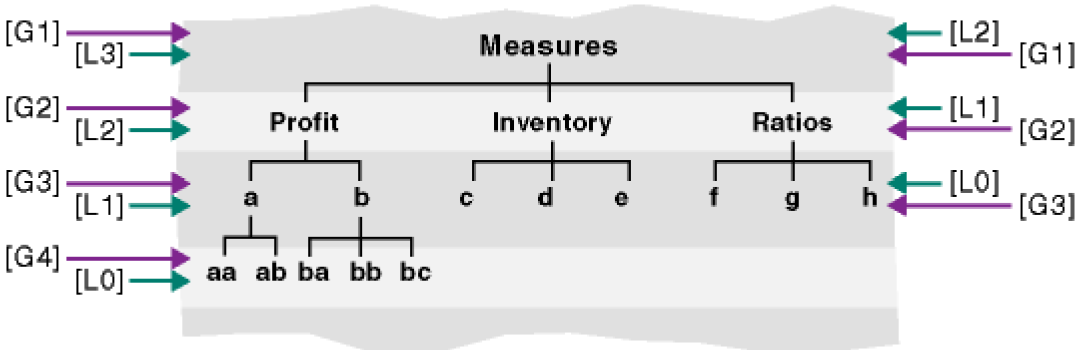


Generation numbers begin counting with 1 at the dimension name; higher generation numbers are those that are closest to leaf members in a hierarchy.

Level numbers begin with 0 at the deepest part of the hierarchy; the highest level number is a dimension name.

Asymmetric Hierarchies

In asymmetric (or ragged) hierarchies, having the same level number does not mean that members are at the same depth in the outline. For example, in the following diagram, member aa and member f are both level 0 members, and yet they are not at the same depth:



Standard Dimensions and Attribute Dimensions

Essbase has standard dimensions and attribute dimensions. This chapter focuses on standard dimensions, because Essbase does not allocate storage for attribute dimension members. Instead, it dynamically calculates the members when the user queries data associated with them.

An attribute dimension is a special type of dimension that is associated with a standard dimension. See Working with Attributes.

Sparse and Dense Dimensions

Most data sets of multidimensional databases have two characteristics:

- Data is *not* smoothly and uniformly distributed.
- Data does *not* exist for the majority of member combinations. For example, all products may not be sold in all areas of the country.

Essbase maximizes performance by dividing the standard dimensions of an application into two types: dense dimensions and sparse dimensions. This division allows Essbase to cope with data that is not smoothly distributed, without losing the advantages of matrix-style access to the data. Essbase speeds data retrieval while minimizing memory and disk requirements.

Most multidimensional databases are inherently sparse; they lack data values for the majority of member combinations. A sparse dimension is one with a low percentage of available data positions filled.

For example, the outline of the Sample.Basic database in Figure 2-4 includes the Year, Product, Market, Measures, and Scenario dimensions. Product represents the product units, Market represents the geographical regions in which the products are sold, and Measures represents the accounts data. Because not every product is sold in every market, Market and Product are chosen as sparse dimensions.

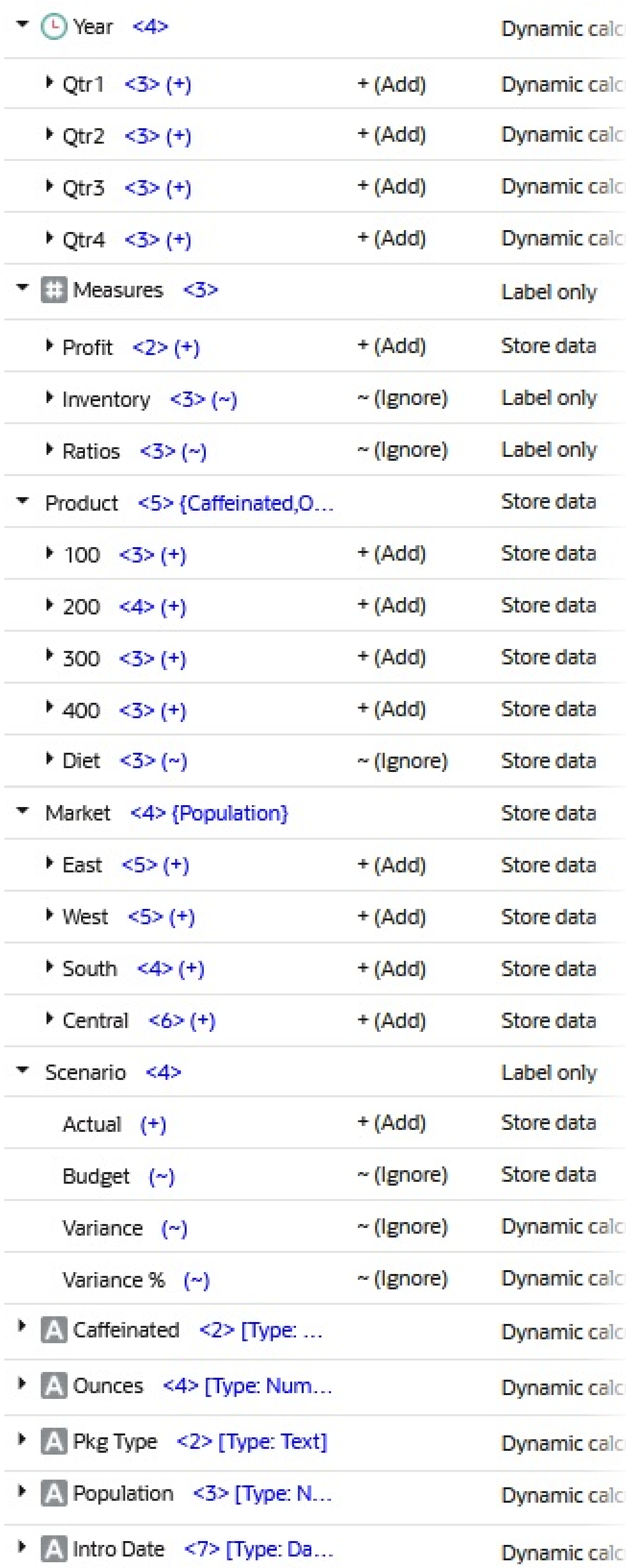
Multidimensional databases also contain dense dimensions. A dense dimension has a high probability that one or more cells is occupied in every combination of dimensions. For example, in the Sample.Basic database, accounts data exists for almost all products in all markets, so

Measures is chosen as a dense dimension. Year and Scenario are also chosen as dense dimensions. Year represents time in months, and Scenario represents whether the accounts values are budget or actual values.

Caffeinated, Intro Date, Ounces, Pkg Type and Population are attribute dimensions. See Working with Attributes.

When an Essbase database is stored on disk, the cartesian product of dense member combinations form units of storage called blocks, and a block is written to disk for every sparse member combination in the database.

Figure 2-4   Sample.Basic Database Outline



Selection of Dense and Sparse Dimensions

In most data sets, existing data tends to follow predictable patterns of density and sparsity. If you match patterns correctly, you can store the existing data in a reasonable number of fairly dense data blocks, rather than in many highly sparse data blocks.

By default, a new dimension is set to sparse. Attribute dimensions are always sparse dimensions. Keep in mind that you can associate attribute dimensions only with sparse standard dimensions.

Dense-Sparse Configuration for Sample.Basic

Consider the Sample.Basic database, which represents data for The Beverage Company (TBC).

Because TBC does not sell every product in every market, the data set is reasonably sparse. Data values do not exist for many combinations of members in the Product and Market dimensions. For example, if Caffeine Free Cola is not sold in Florida, data values do not exist for the combination Caffeine Free Cola (100-30) -> Florida, so Product and Market are sparse dimensions. Therefore, if no data values exist for a specific combination of members in these dimensions, a data block is not created for the combination.

However, consider combinations of members in the Year, Measures, and Scenario dimensions. Data values almost always exist for some member combinations on these dimensions. For example, data values exist for the member combination Sales -> January -> Actual, because at least some products are sold in January. Thus, Year and, similarly, Measures and Scenario, are dense dimensions.

The sparse-dense configuration of the standard dimensions in the Sample.Basic database may be summarized:

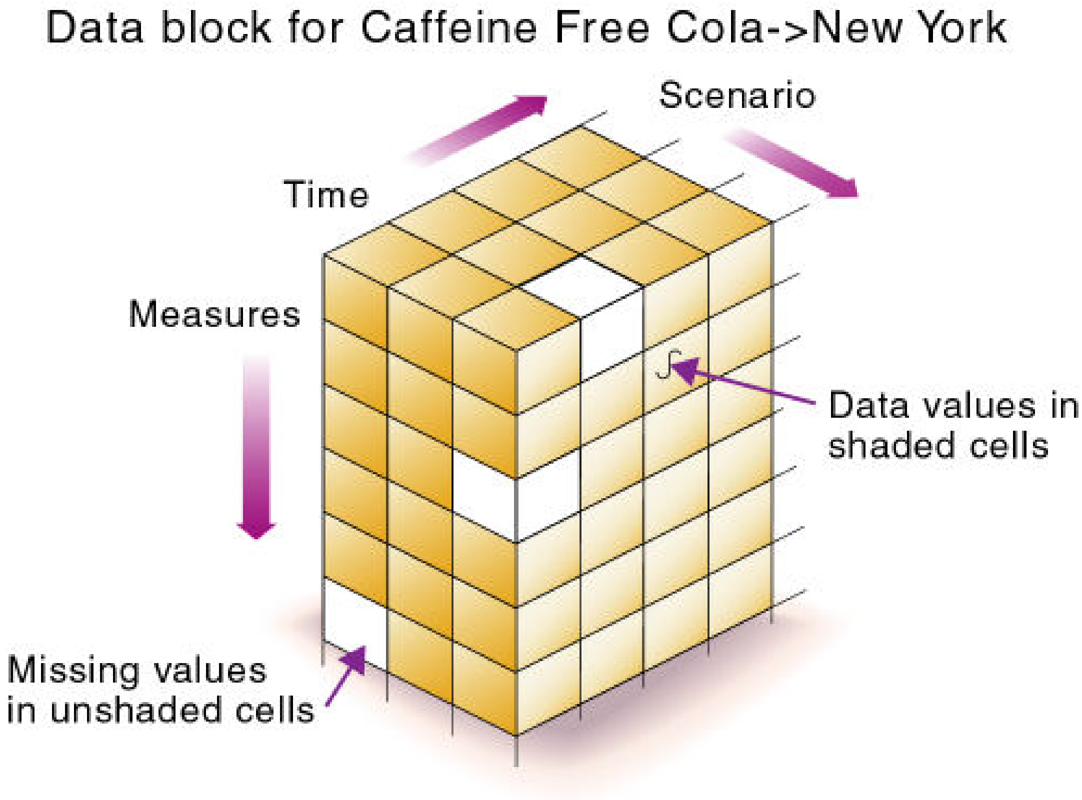
- The sparse standard dimensions are Product and Market.
- The dense standard dimensions are Year, Measures, and Scenario.

A data block is created for each unique combination of members in the Product and Market dimensions (see Data Storage). Each data block represents data from the dense dimensions. The data blocks likely will have few empty cells.

For example, consider the sparse member combination Caffeine Free Cola (100-30), New York, in Figure 2-5:

- If accounts data (represented by the Measures dimension) exists for this combination for January, it probably exists for February and for all members in the Year dimension.
- If a data value exists for one member on the Measures dimension, it is likely that other accounts data values exist for other members in the Measures dimension.
- If Actual accounts data values exist, it is likely that Budget accounts data values exist.

Figure 2-5 Dense Data Block for Sample.Basic Database



Dense and Sparse Selection Scenarios

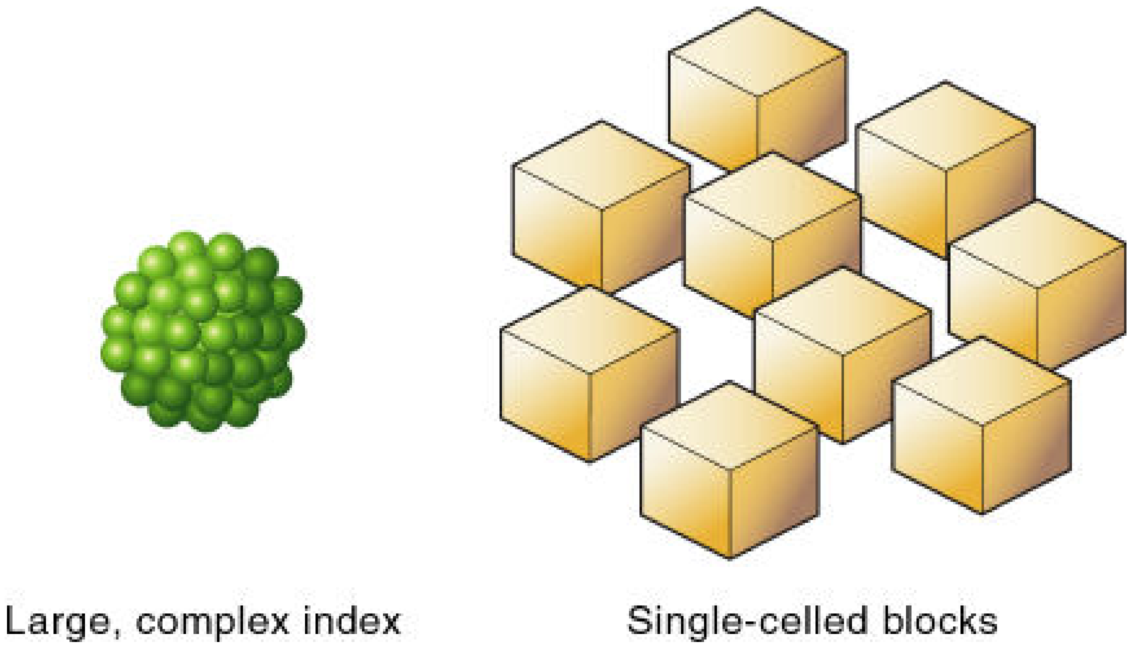
In the following scenarios, you'll see how a database is affected when you select different standard dimensions. Assume that these scenarios are based on typical databases with at least seven dimensions and several hundred members.

Scenario 1: All Sparse Standard Dimensions

If you make all dimensions sparse, Essbase creates data blocks that consist of single data cells that contain single data values. An index entry is created for each data block and, therefore, in this scenario, for each existing data value.

This configuration produces an index that requires a large memory. The more index entries, the longer Essbase searches for a specific block.

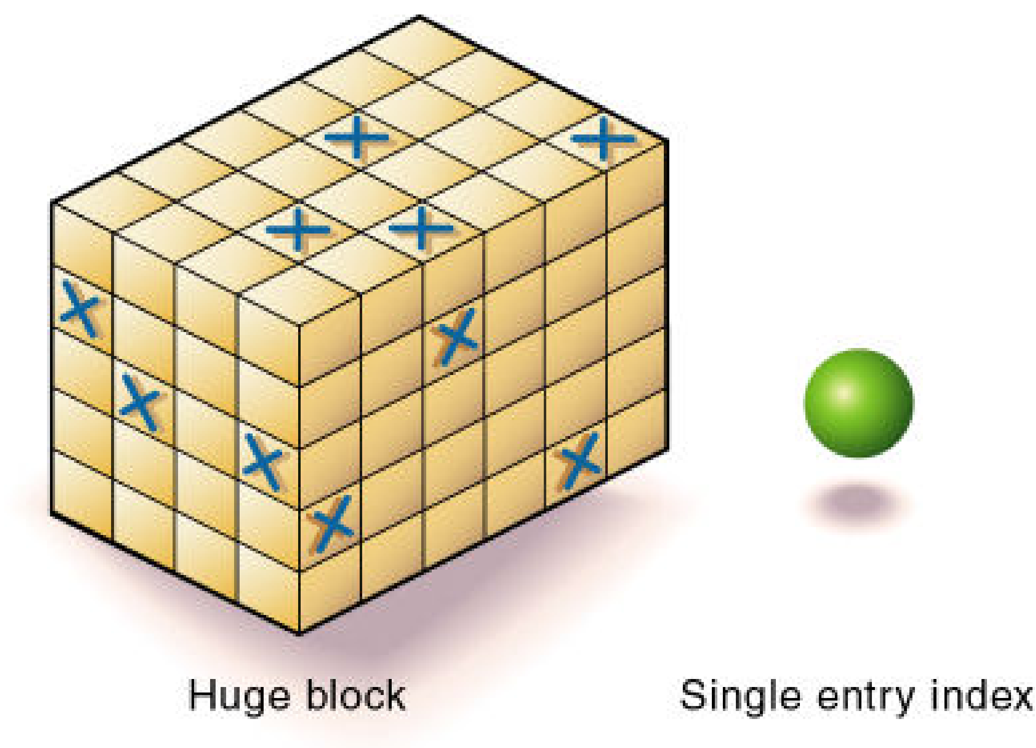
Figure 2-6   Database with All Sparse Standard Dimensions



Scenario 2: All Dense Standard Dimensions

If you make all dimensions dense, Essbase creates one index entry and one large, sparse block. In most applications, this configuration requires thousands of times more storage than other configurations. Essbase must load the entire database into memory when it searches for any data value, which requires enormous memory.

Figure 2-7   Database with All Dense Standard Dimensions

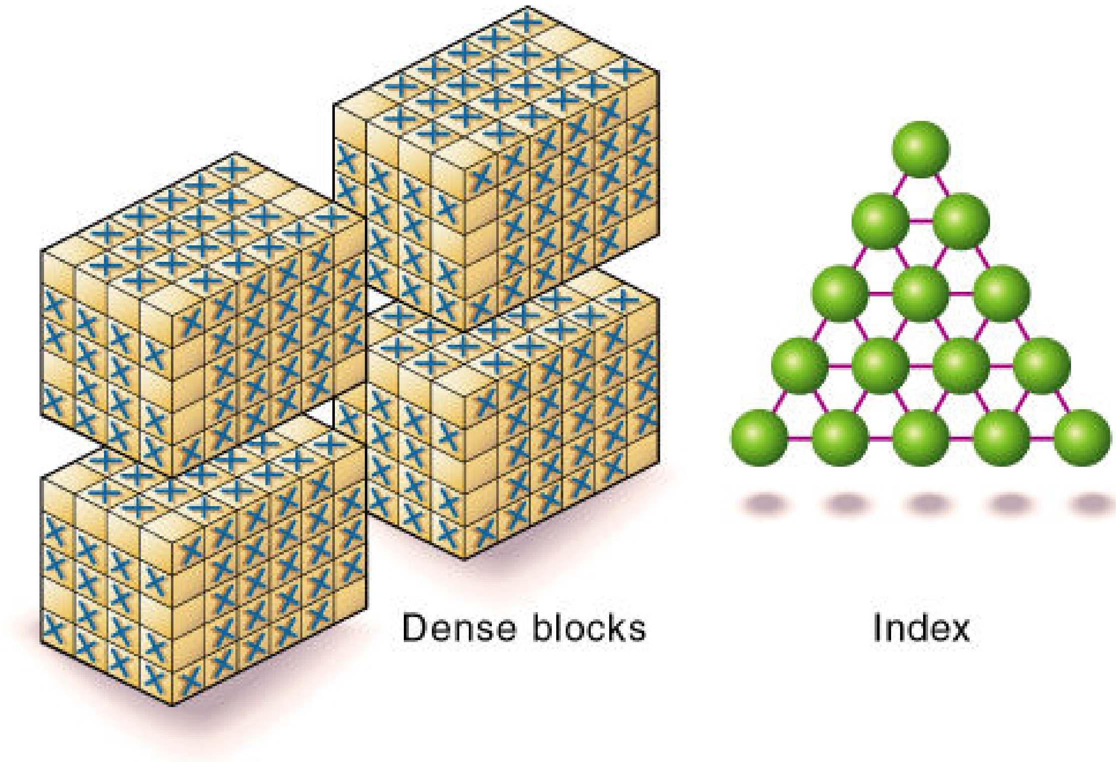


Scenario 3: Dense and Sparse Standard Dimensions

Based on your knowledge of your company's data, you have identified all your sparse and dense standard dimensions.

Essbase creates dense blocks that can fit into memory easily and creates a relatively small index. Your database runs efficiently using minimal resources.

Figure 2-8 An Ideal Configuration with Combination of Dense and Sparse Dimensions

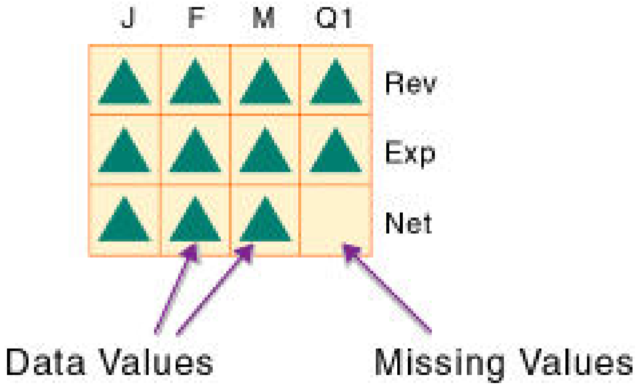


Scenario 4: A Typical Multidimensional Problem

Consider a database with four standard dimensions: Time, Accounts, Region, and Product. In the following example, Time and Accounts are dense dimensions, and Region and Product are sparse dimensions.

The two-dimensional data blocks shown in the image below represent data values from the dense dimensions: Time and Accounts. The members in the Time dimension are J, F, M, and Q1. The members in the Accounts dimension are Rev, Exp, and Net.

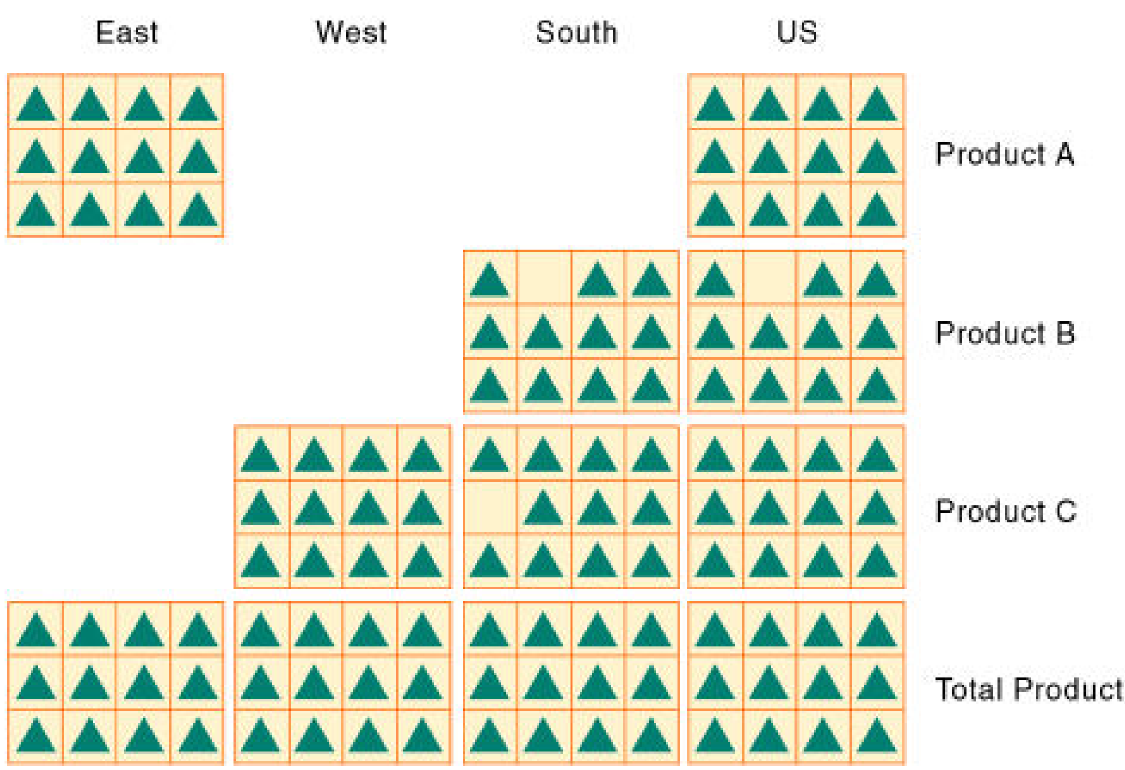
Figure 2-9 Two-dimensional Data Block for Time and Accounts



Essbase creates data blocks for combinations of members in the sparse standard dimensions (providing that at least one data value exists for the member combination). The sparse dimensions are Region and Product. The members of the Region dimension are East, West, South, and Total US. The members in the Product dimension are Product A, Product B, Product C, and Total Product.

The image below shows 11 data blocks. No data values exist for Product A in the West and South, for Product B in the East and West, or for Product C in the East. Therefore, Essbase has not created data blocks for these member combinations. The data blocks that Essbase has created have few empty cells. This example effectively concentrates all sparseness into the index and concentrates all data into fully utilized blocks. This configuration provides efficient data storage and retrieval.

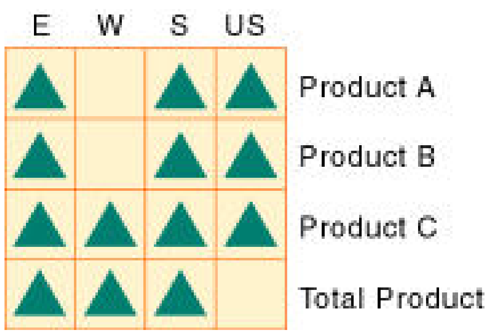
Figure 2-10 Data Blocks Created for Sparse Members on Region and Product



Next, consider a reversal of the dense and sparse dimension selections. In the following example, Region and Product are dense dimensions, and Time and Accounts are sparse dimensions.

In the image below, the two-dimensional data blocks represent data values from the dense dimensions: Region and Product. In the West region, data is not available for Product A and Product B. Data is also not available for Total Product in US.

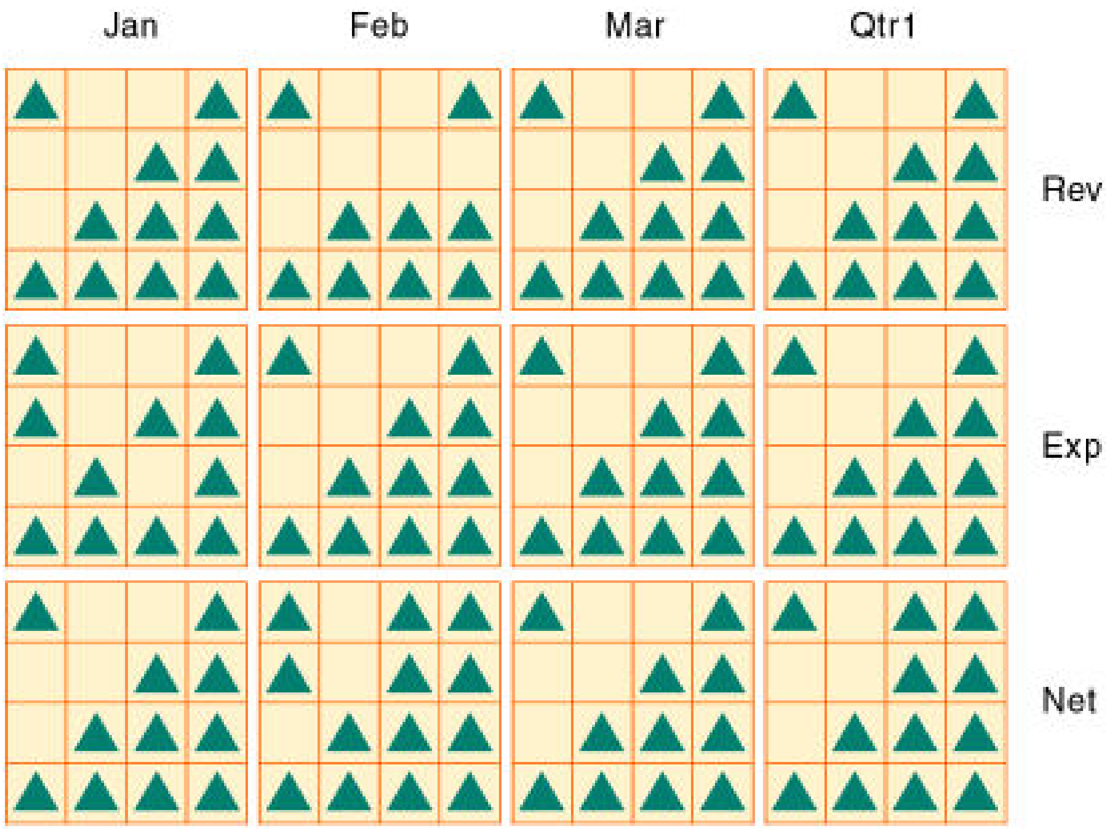
Figure 2-11 Two-Dimensional Data Block for Region and Product



Essbase creates data blocks for combinations of members in the sparse standard dimensions (providing that at least one data value exists for the member combination). The sparse standard dimensions are Time and Accounts.

The image below shows 12 data blocks. Data values exist for all combinations of members in the Time and Accounts dimensions; therefore, Essbase creates data blocks for all member combinations. Because data values do not exist for all products in all regions, the data blocks have many empty cells. Data blocks with many empty cells store data inefficiently.

Figure 2-12 Data Blocks Created for Sparse Members on Time and Accounts



Data Storage

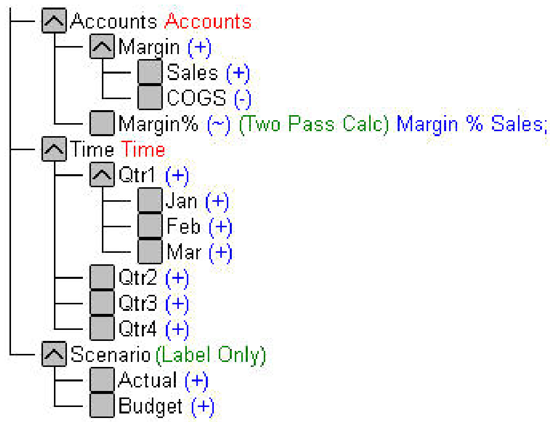
Each data value in a multidimensional database is stored in one cell. A particular data value is referenced by specifying its coordinates along *each* standard dimension.

> **Note:**
>
> Essbase does not store data for attribute dimensions. Essbase dynamically calculates attribute dimension data when a user retrieves the data.

Consider the simplified database shown in Figure 2-13. This database has three dimensions: Accounts, Time, and Scenario:

- The Accounts dimension has four members: Sales, COGS, Margin, and Margin%.
- The Time dimension has four quarter members, and Qtr1 has three month members
- The Scenario dimension has two child members: Budget for budget values and Actual for actual values.

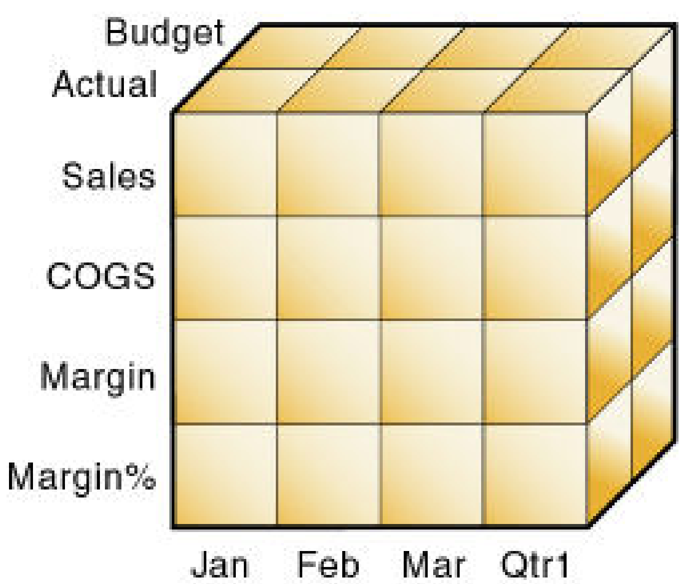
Figure 2-13 A Multidimensional Database Outline



Data Values

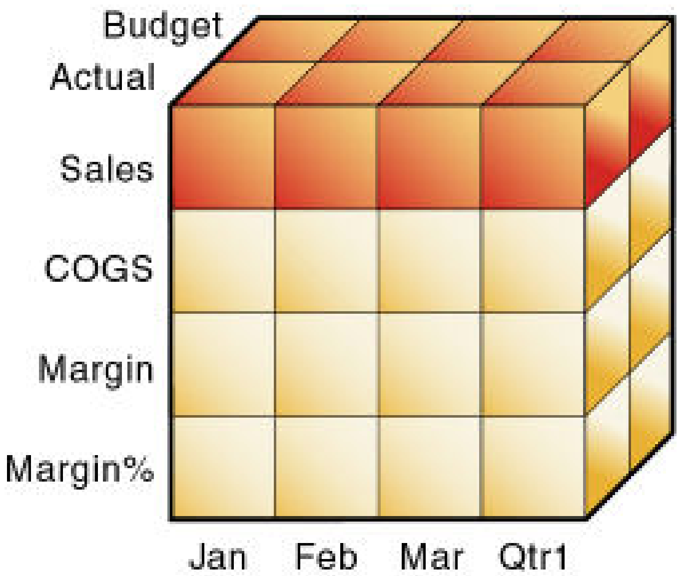
The intersection of one member from one dimension with one member from each of the other dimensions represents a data value. The example in Figure 2-14 has three dimensions (Accounts, Time, and Scenario); therefore, the dimensions and data values in the database can be represented as a cube.

Figure 2-14 Three-Dimensional Database



As illustrated in Figure 2-15, when you specify Sales, you are specifying the slice of the database that contains eight Sales values, where Sales intersect with Actual and Budget.

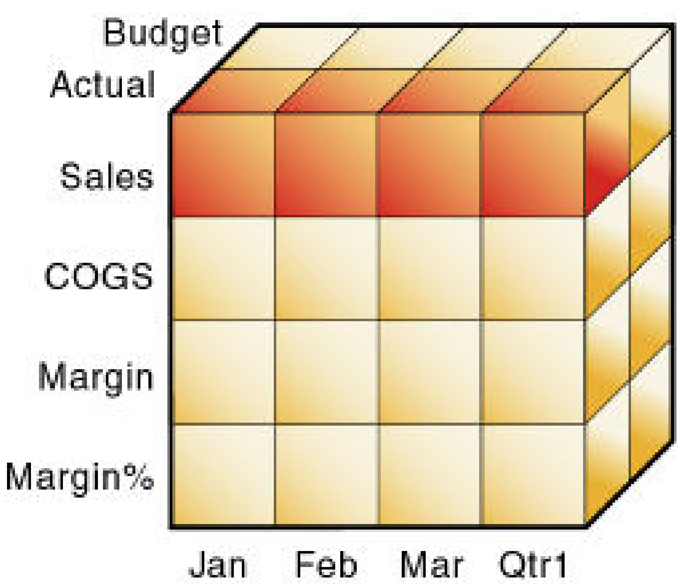
Figure 2-15 Sales Slice of the Database



Slicing a database amounts to fixing one or more dimensions at a constant value while allowing the other dimensions to vary.

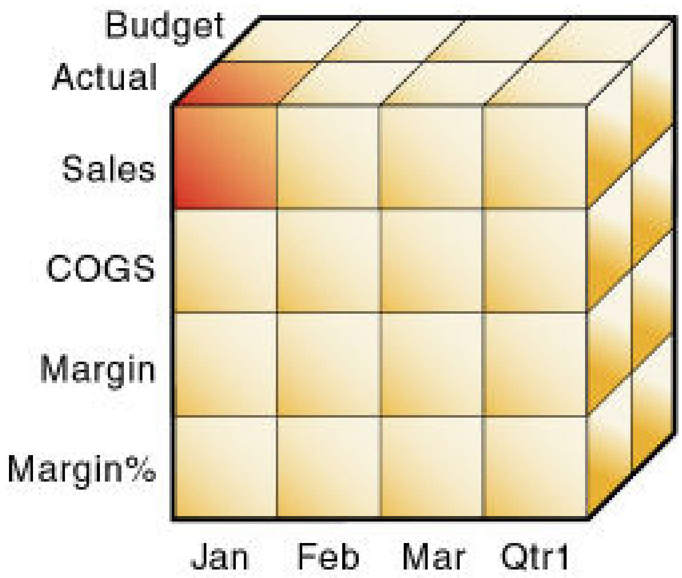
As illustrated in Figure 2-16, when you specify Actual Sales, you are specifying the slice of the database that contains four Sales values, where Actual and Sales intersect.

Figure 2-16 Actual, Sales Slice of the Database



A data value is stored in one cell in the database. To refer to a specific data value in a multidimensional database, you specify its member on each dimension. In Figure 2-17, the cell containing the data value for Sales, Jan, Actual is shaded. The data value can also be expressed using the cross-dimensional operator (->) as Sales -> Actual -> Jan.

Figure 2-17 Sales->Jan->Actual Slice of the Database



Data Blocks and the Index System

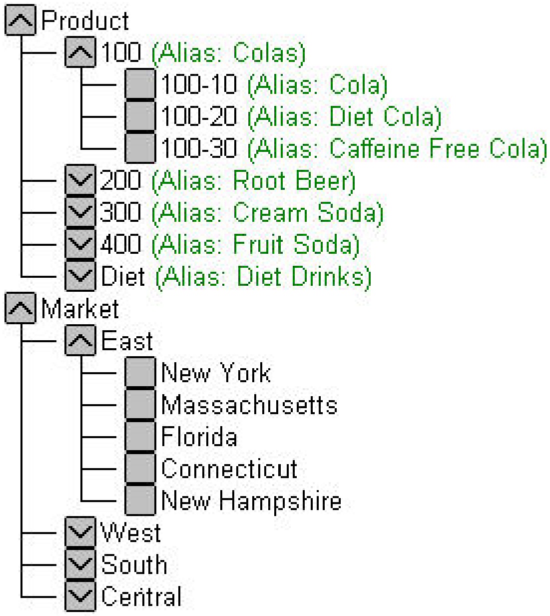
Essbase uses two types of internal structures to store and access data: data blocks and the index system.

Essbase creates a data block for each unique combination of sparse standard dimension members (providing that at least one data value exists for the sparse dimension member combination). The data block represents all the dense dimension members for its combination of sparse dimension members.

Essbase creates an index entry for each data block. The index represents the combinations of sparse standard dimension members. It contains an entry for each unique combination of sparse standard dimension members for which at least one data value exists.

For example, in the Sample.Basic database outline shown in Figure 2-18, Product and Market are sparse dimensions.

Figure 2-18 Product and Market Dimensions from the Sample.Basic Database



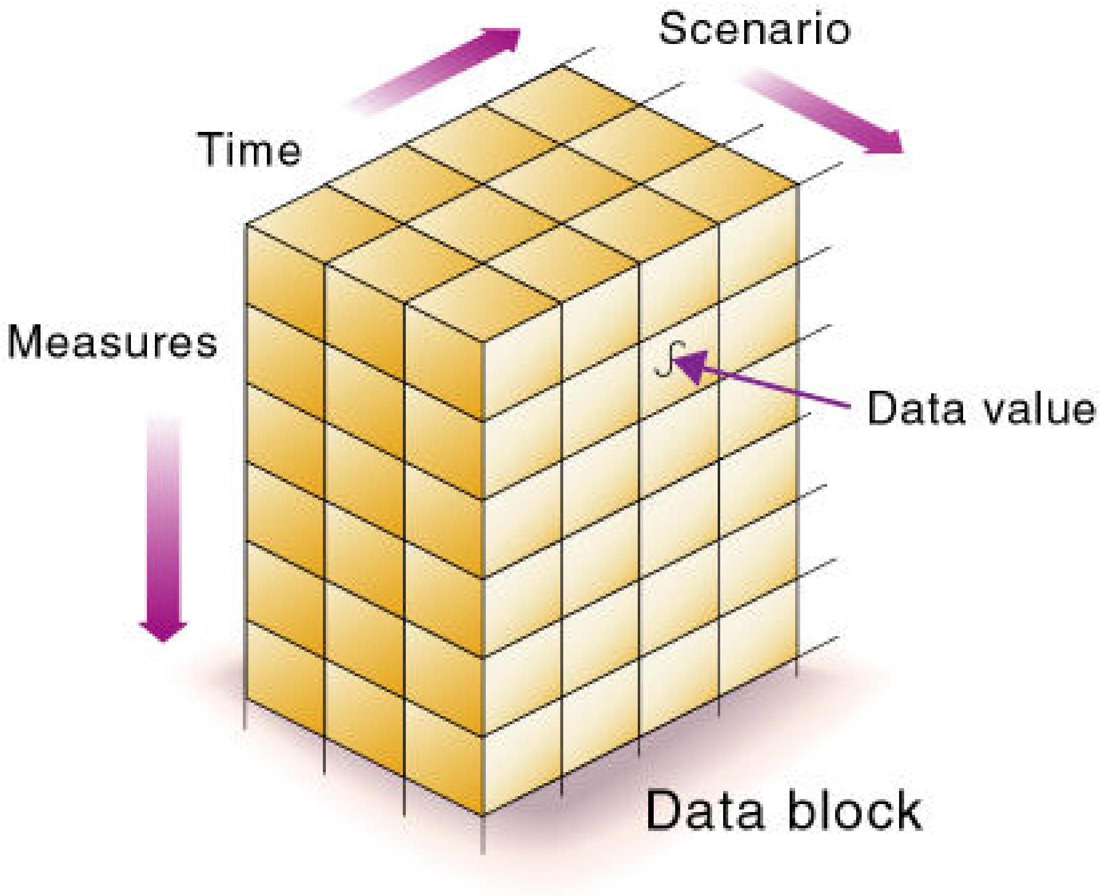
If data exists for Caffeine Free Cola in New York, Essbase creates a data block and an index entry for the sparse member combination of Caffeine Free Cola (100-30) -> New York. If Caffeine Free Cola is *not* sold in Florida, Essbase does *not* create a data block or an index entry for the sparse member combination: Caffeine Free Cola (100-30) -> Florida.

The data block Caffeine Free Cola (100-30) -> New York represents all the Year, Measures, and Scenario dimensions for Caffeine Free Cola (100-30) -> New York.

Each unique data value can be considered to exist in a cell in a data block. When Essbase searches for a data value, it uses the index to locate the appropriate data block. Then, within the data block, it locates the cell containing the data value. The index entry provides a pointer to the data block. The index handles sparse data efficiently because it includes only pointers to existing data blocks.

Figure 2-19 shows part of a data block for the Sample.Basic database. Each dimension of the block represents a dense dimension in the Sample.Basic database: Time, Measures, and Scenario. A data block exists for each unique combination of members of the Product and Market sparse dimensions (providing that at least one data value exists for the combination).

Figure 2-19 Part of a Data Block for the Sample.Basic Database



Each data block is a multidimensional array that contains a fixed, ordered location for each possible combination of dense dimension members. Accessing a cell in the block does not involve sequential or index searches. The search is almost instantaneous, resulting in optimal retrieval and calculation speed.

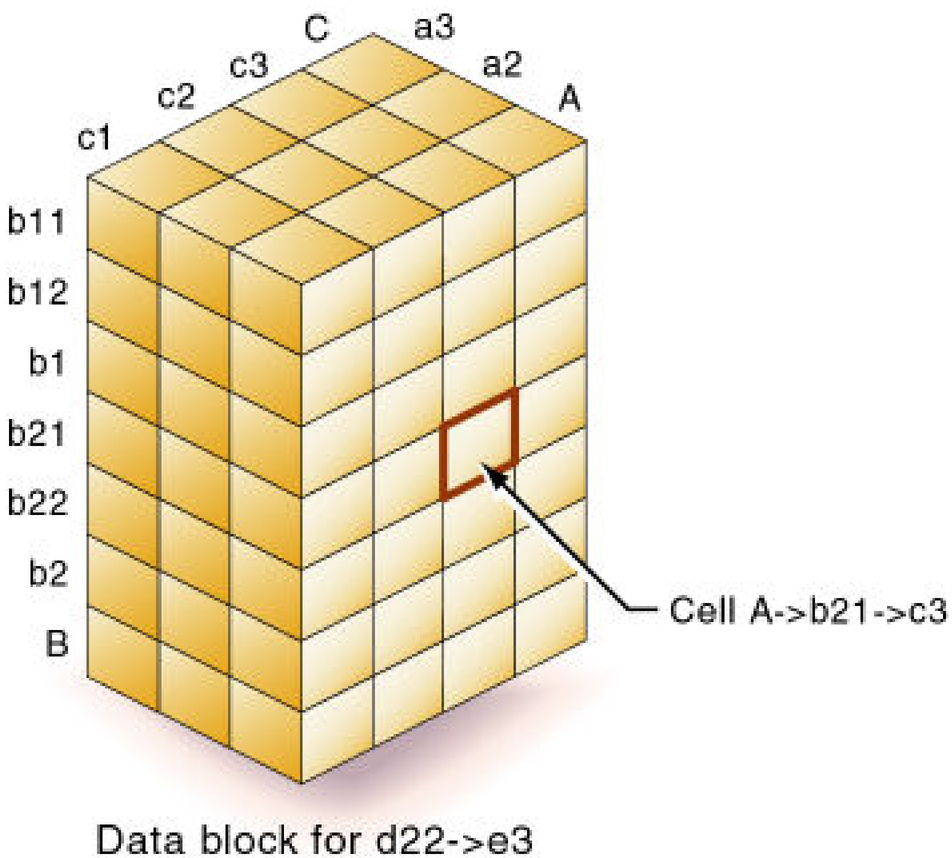
Essbase orders the cells in a data block according to the order of the members in the dense dimensions of the database outline.

```
A (Dense)
   a1
   a2
B (Dense)
   b1
       b11
       b12
   b2
       b21
       b22
C (Dense)
   c1
   c2
   c3
D (Sparse)
```

```
   d1
   d2
       d21
       d22
E (Sparse)
   e1
   e2
   e3
```

The block in Figure 2-20 represents the three dense dimensions from within the combination of the sparse members d22 and e3, from the preceding database outline. In Essbase, member combinations are denoted by the cross-dimensional operator. The symbol for the cross-dimensional operator is ->, so d22 -> e3 denotes the block for d22 and e3. The intersection of A, b21, and c3 is written as A -> b21 -> c3.

Figure 2-20 Data Block Representing Dense Dimensions for d22 -> e3



Essbase creates a data block for every unique combination of the members of the sparse dimensions D and E (providing that at least one data value exists for the combination).

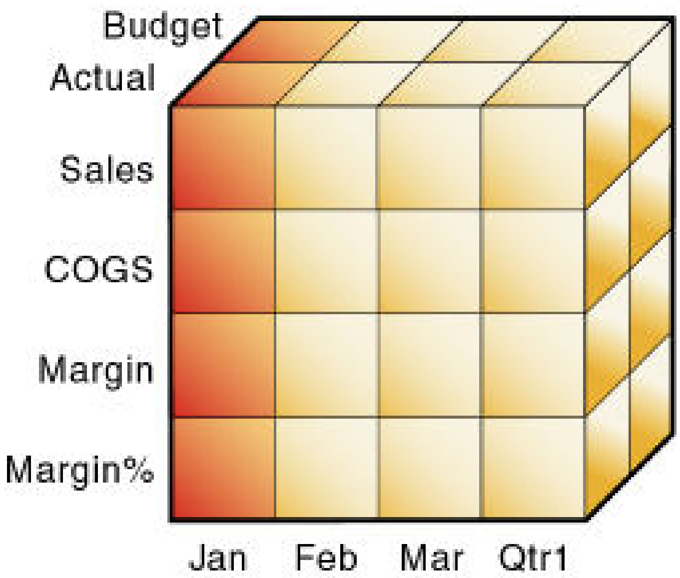
Data blocks, such as the one in Figure 2-20, may include cells that do not contain data values. A data block is created if at least one data value exists in the block. Essbase compresses data blocks with missing values on disk, expanding each block fully as it brings the block into memory. Data compression is optional but is enabled by default.

By carefully selecting dense and sparse standard dimensions, you can ensure that data blocks do not contain many empty cells, minimizing disk storage requirements and improving performance. In Essbase, empty cells are known as #MISSING data.

Multiple Data Views

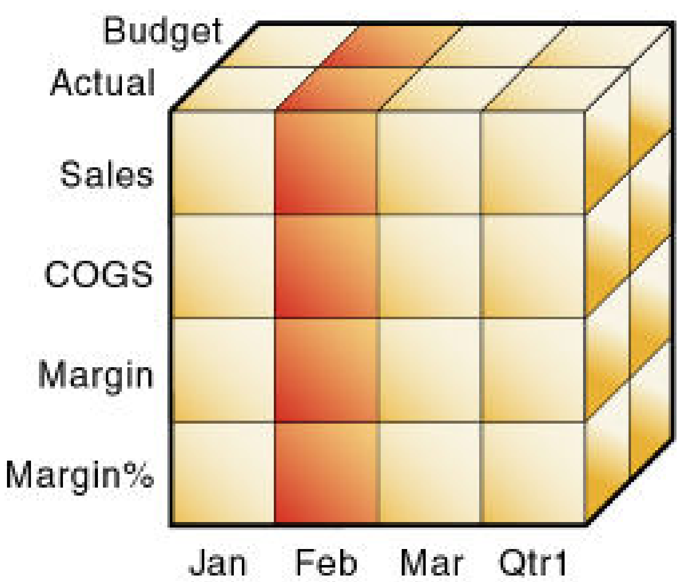
A multidimensional database supports multiple views of data sets for users who need to analyze the relationships between data categories. Slicing the database in different ways gives you different perspectives of the data. For example, in Figure 2-21, the slice for Jan examines all data values for which the Year dimension is fixed at Jan.

Figure 2-21 Data for January



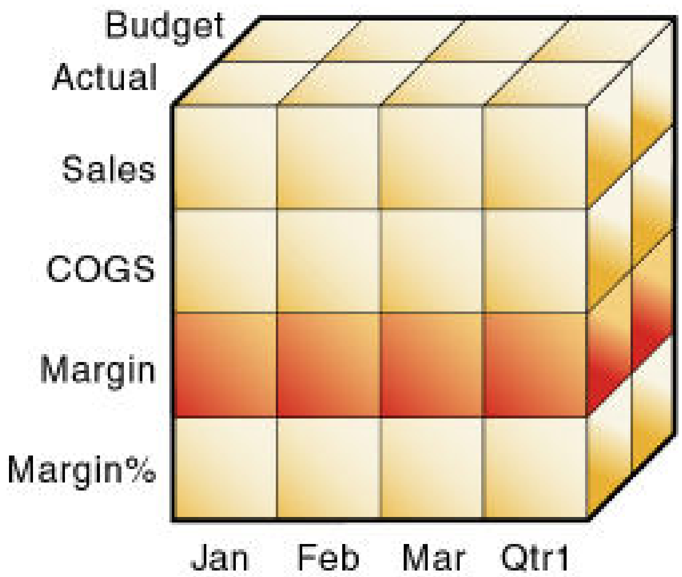
The slice in Figure 2-22 shows data for the month of Feb:

Figure 2-22 Data for February



The slice in Figure 2-23 shows data for profit margin:

Figure 2-23 Data for Profit Margin



The Essbase Solution for Creating Optimized Databases

To create an optimized database, ask:

- How does your company use the data?
- How will you build and order the dimensions?
- How will you create and order calculations?

See these topics:

- Planning the development of your multidimensional database, see Case Study: Designing a Single-Server, Multidimensional Database.
- Selecting dense and sparse dimensions, see Selection of Dense and Sparse Dimensions.
- Loading data, see Overview of Data Load and Dimension Build.
- Calculating your database, see Calculating Essbase Databases.

3

Creating Applications and Databases

After you familiarize yourself with applications, and the databases contained within them, you can proceed to create them.

- Understanding Applications and Databases
- Understanding Database Artifacts
- Creating an Application and Database
- Using Substitution Variables
- Using Location Aliases

Some information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases. Also see Comparison of Aggregate and Block Storage.

Understanding Applications and Databases

An application is a management structure that contains one or more databases and related files. Block storage applications can contain multiple databases; aggregate storage applications can contain one database only. Even for block storage databases, Oracle recommends creating one database per application. An Essbase service can store multiple applications and databases.

The database is a data repository that contains a multidimensional data storage array. A multidimensional database supports multiple views of data so that users can analyze the data and make meaningful business decisions. See Understanding Multidimensional Databases.

> **Note:**
>
> Some information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases. Also see:
>
> - Comparison of Aggregate and Block Storage
> - Designing Aggregate Storage Applications

Understanding Database Artifacts

Files that are related to databases are called artifacts (or objects). Database artifacts perform actions against one or more databases, such as defining calculations. By default, artifacts are stored in their associated database folder, and can also be saved to a client computer or to other available network directories.

Common artifact types:

- A database outline (a storage structure definition)
- Source data files

- Rules for loading data and building dimensions dynamically (rule files)
- Scripts that define how to calculate data (calculation scripts)
- Partition definitions

Understanding Database Outlines

Database outlines define the structure of a multidimensional database, including all the dimensions, members, aliases, properties, types, consolidations, and mathematical relationships. The structure defined in the outline determines how data is stored in the database.

When a database is created, an outline for that database is created automatically.

See Creating an Application and Database and Creating and Changing Database Outlines.

Understanding Source Data

Source data is external data that is loaded into an Essbase database. Common types of source data include the following:

- Text files
- External databases, such as an SQL database

See Supported Source Data Types.

Understanding Rule Files for Data Load and Dimension Build

An Essbase database contains no data when it is created. Data load rule files are sets of operations that Essbase performs on data from an external source as the data is loaded, or copied, into the Essbase database. Dimension build rule files create or modify the dimensions and members in an outline based on data in an external data source. Rules files are typically associated with a particular database, but you can define rules for use with multiple databases. One rule file can be used for both data loads and dimension builds. Rule files have a `.rul` extension.

See Load Rules and Design Rules for Data Load or Dimension Build.

Understanding Calculation Scripts

Calculation scripts are text files that contain sets of instructions telling Essbase how to calculate data in the database. Calculation scripts perform calculations different from the consolidations and mathematical operations that are defined in the database outline. Because calculation scripts perform specific mathematical operations on members, they are typically associated with a particular database. You can, however, define a calculation script for use with multiple databases. Calculation scripts files have a `.csc` extension.

See Develop Calculation Scripts for Block Storage Cubes.

Creating an Application and Database

Create an application and its database.

You can easily create an application and database using an Excel-based application workbook. You can use a sample application workbook and modify it for your use. See About Application Workbooks.

If you already created an application and database, and you want to base a new application and database on it with some modifications, export the application and database to an application workbook. After you make the changes to the application workbook, you can then import the new application and database. See Create a Cube from an Application Workbook.

You can use MaxL statements to create applications and databases:, you can use the **create application** MaxL statement.

- **create application**
- **create database**

Before naming applications and databases, see Naming Conventions for Applications and Databases.

Using Substitution Variables

Substitution variables are global placeholders for regularly changing information. Because changes to a variable value are reflected everywhere the variable is used, manual changes are reduced.

For example, many reports depend on reporting periods; if you generate a report based on the current month, you must update the report script manually every month. With a substitution variable, such as `CurMnth`, set on the server, you can change the assigned value each month to the appropriate time period. When you use the variable name in a report script, the information is dynamically updated when you run the final report.

You can use substitution variables with both aggregate storage and block storage applications (unless otherwise noted) in the following areas:

- Aggregate storage outline formulas
- Block storage outline formulas
- Calculation scripts (block storage databases only)

  Substitution variables and runtime substitution variables are supported in calculation scripts.
- Data load rules file header definitions and field definitions. You can enter variable names for dimension and member names.
- Data source name (DSN) specifications in rules files for SQL data sources
- SELECT, FROM, or WHERE clauses in rules files for SQL data sources
- Security filters
- MDX queries
- Smart View

You can set substitution variables at these levels:

- Globally: Provides access to the variable from all applications and databases on the Essbase instance.
- Application: Provides access to the variable from all databases within the application.
- Database: Provides access to the variable within the specified database.

Rules for Setting Substitution Variable Names and Values

When you use substitution variables in Essbase, take note of guidelines pertaining to the characters and data types you can use in substitution variable names and values.

The following rules apply to substitution variable names and values:

Restriction type	Guideline
Characters allowed in variable *name*	Alphanumeric characters and underscores are permitted, but not special characters or spaces. Limit of characters: see Limits.
Characters allowed in variable *value*	Any character is allowed except a leading ampersand (`&`). Limit of characters: see Limits.
Essbase member names used as variable values	Enclose member names that start with a number or contain any special characters in quotation marks (`" "`) for block storage databases, and in brackets (`[ ]`) for aggregate storage databases. See Naming Conventions in Calculation Scripts, Report Scripts, Formulas, Filters, and Substitution and Environment Variable Values for more details. To set a substitution variable value to be a duplicate member name, use the qualified member name enclosed in double quotation marks; for example, a value for *&Period* can be `"[2022].[Qtr1]"`. Do not use a substitution variable name as a part of a qualified member name. For example, it is invalid to specify `"[2022].[&CurrentQtr]"`.
Numbers used as variable values	Enclose numeric values in quotation marks (`" "`) for block storage databases, and in brackets (`[ ]`) for aggregate storage databases. For example, if the variable name is Month, and its value is 01, then for block storage, set the value as `"01"`. For aggregate storage, set the value as `[01]`.
Multiple variables having the same name	Allowed if the variables are defined at different scopes. Precedence rules: Variable resolves at database level before at application level. Variable resolves at application level before at global level.

Setting Substitution Variables

You can set substitution variables at the server, application, or database level. Before setting a substitution variable, see Rules for Setting Substitution Variable Names and Values.

When you add or update substitution variables, they are sent to the application and dynamically resolved.

To set a substitution variable, see Using Variables. You can also use application workbook to set at the application level, or use these MaxL statements:

- alter system
- alter application
- alter database

Deleting Substitution Variables

You may need to delete a substitution variable that is no longer used.

To delete a substitution variable, you can remove it from your application workbook, delete it using the Essbase web interface, or use these MaxL statements:

- alter system
- alter application
- alter database

Updating Substitution Variables

You can modify or update existing substitution variables. Before updating a substitution variable, see Rules for Setting Substitution Variable Names and Values.

To update a substitution variable, you can use these MaxL statements:

- alter system
- alter application
- alter database

Copying Substitution Variables

You can copy substitution variables to any Essbase Server, application, or database to which you have appropriate access.

Using Location Aliases

A location alias is a descriptor for a source of data in another cube. A location alias maps an alias name for a cube to the location of that cube. A location alias is set at the cube level. Set the location alias on the cube from which the calculation script is run.

After you create a location alias, you can use the alias to refer to the other cube. If the location of the cube changes, edit the location definition accordingly.

You can use location aliases only with the @XREF and @XWRITE functions. With @XREF, you can retrieve a data value from another cube to include in a calculation on the current cube. In this case, the location alias points to the cube from which the value is to be retrieved. With @XWRITE, you can write values to another Essbase cube, or to the same cube.

You can create a location alias for a particular cube. To create a location alias, see Create a Location Alias Based on a Defined Connection. You can also use the create location alias MaxL statement.

4

Creating and Changing Database Outlines

The database outline defines the structure of the database. The outline includes dimensions and members, and their properties.

- Creating and Editing Outlines
- Locking and Unlocking Outlines
- Setting the Dimension Storage Type
- Positioning Dimensions and Members
- Verifying Outlines
- Saving Outlines

Some information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases. Also see Comparison of Aggregate and Block Storage.

All examples in this chapter are based on the Sample.Basic database, which you can create using the Block Storage Sample (Stored) application workbook (Sample_Basic.xlsx). See About Application Workbooks.

Process for Creating Outlines

This section provides an overview of creating outlines. For basic information about outlines, see Understanding Multidimensional Databases.

1. Create a database. The new database automatically contains a blank outline.

   See Creating Applications and Databases.

2. Open the outline.

   See Creating and Editing Outlines.

3. Add dimensions and members to the outline.
4. Set each dimension as dense or sparse.

   See Setting the Dimension Storage Type.

5. Position dimensions and members in the outline.

   See Positioning Dimensions and Members.

6. Set dimension and member properties.

   See Setting Dimension and Member Properties.

7. If necessary, create attribute dimensions and associate them with the appropriate base dimensions.

   See Working with Attributes.

8. Verify and save the outline.

   See Figure 4-1 and Saving Outlines.

Creating and Editing Outlines

You can create content in the new outline in the following ways:

- Create a new application and cube, and add dimensions and members, by importing an Excel-based application workbook.

  See About Application Workbooks.
- Open the empty outline created by default when you create a database and add content manually.

  See Add Dimensions to Outlines Manually and Add Members to Outlines Manually.
- Build dimensions using a flat file and a rules file.

  See Build Dimensions Using a Rules File.
- Import tabular data and allow Essbase to create a cube
- Import an On Premises application using LCM utility export and CLI import.

You can create an outline using the **create database** MaxL statement.

You can copy an outline using the **create database as** MaxL statement.

Locking and Unlocking Outlines

When an outline is locked, other users are not allowed to save over, rename, delete, or edit the outline.

You can review a list of the modifications that you made. When you are finished editing the outline, you can either save or revert your modifications.

> **Note:**
>
> Essbase uses a different process for locking and unlocking outlines than for other database artifacts.

The outline is automatically unlocked upon saving it.

Setting the Dimension Storage Type

If you are using an application workbook to design the database, you assign the storage type for each dimension in the Essbase.Cube worksheet. See Understanding the Essbase.Cube Worksheet.

In the Essbase web interface, you can also assign the storage type for each dimension in the outline.

Positioning Dimensions and Members

The following sections describe how to position dimensions and members in the outline.

- Moving Dimensions and Members

- Sorting Dimensions and Members

Moving Dimensions and Members

After you create dimensions and members, you can rearrange them within the outline. Before moving members and dimensions in an outline, consider the following information:

- The positions of dimensions and members in an outline can affect performance.

  See Optimizing Outline Performance.
- Moving dimensions and members can affect the performance of calculations and retrievals.

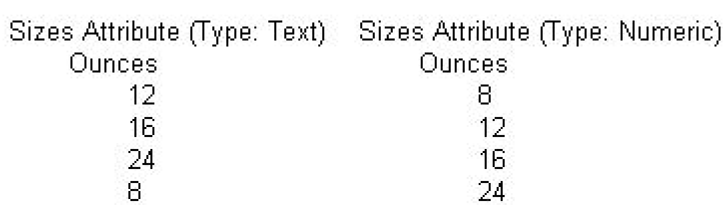
  See Designing an Outline to Optimize Performance.
- Moving members could move a shared member before the actual member in the outline (which is not recommend).
- If you add, delete, or move nonattribute dimensions or members, Essbase restructures the database, and you might need to recalculate the data.
- Position attribute dimensions at the end of the outline.

Sorting Dimensions and Members

You can have Essbase arrange dimensions within an outline or members within a dimension in alphabetical order (A–Z) or reverse alphabetical order (Z–A). For a list of consequences of sorting dimensions and members, see Moving Dimensions and Members.

When you sort level 0 members of numeric attribute dimensions in outlines, the members are sorted by their values. For example, Figure 4-1 shows text and numeric versions of the Sizes attribute dimension after sorting the members in ascending order. The members of the numeric attribute dimension (on the right) are sequenced by the numeric values of the members; the member 8 is before the other members. In the text attribute dimension (on the left), because the characters are sorted left to right, the member 8 is after the member 24.

Figure 4-1 Sorting Numeric Versus Text Attribute Dimension in Ascending Order



You cannot sort Boolean attribute dimensions. See Understanding Attribute Types.

Verifying Outlines

When you save an outline, the outline is automatically verified. When verifying an outline, the following items are checked:

- All member and alias names are valid. Members and aliases cannot have the same name as other members, aliases, generations, or levels, in non-duplicate member outlines.

  See Naming Conventions for Applications and Databases.

- Only one dimension is tagged as accounts or time.
- Shared members are valid.

  See Guidelines for Shared Members.
- Level 0 members are not tagged as label only or Dynamic Calc (unless the member has a formula).
- Label-only members have not been assigned formulas.
- A descendant of a label only member is not tagged as Dynamic Calc.

  See Label Only Members.
- Parent members with only one Dynamic Calc child, are Dynamic Calc.
- Parent members with only oneDynamic Calc child, which is two-pass, the parent member must also be Dynamic Calc, two-pass.
- The two names of members of Boolean attribute dimensions are the same as the two Boolean attribute dimension member names defined for the outline.
- The level 0 member name of a date attribute dimension matches the date format name setting (mm-dd-yyyy or dd-mm-yyyy).
- The level 0 member name of a numeric attribute dimension is a numeric value.
- Attribute dimensions are located at the end of the outline, following all standard dimensions.
- Level 0 Dynamic Calc members of standard dimensions have a formula.
- Formulas for members are valid.

> **Note:**
>
> An outline with Boolean, numeric or date type attribute dimensions that have no members (the attribute dimension is empty) is allowed but should only be used in an interim outline. The results of queries, calculations, and grid operations on empty attribute dimensions can be unpredictable.

During outline verification, the following conversions to appropriate numeric attribute dimension member names are made in the outline:

- It moves minus signs in member names from the front to the end of the name; for example, –1 becomes 1–.
- It strips out leading or trailing zeroes in member names; for example, 1.0 becomes 1, and 00.1 becomes 0.1.

See Understanding Attribute Types.

Saving Outlines

When you save changes to an outline, the outline is restructured. For example, if you change a member name from Market to Region, the data stored in Market is moved to Region. Each time that you save an outline, the outline is verified to ensure that it is correct.

Saving an Outline with Added Standard Dimensions

If you add one or more new standard (nonattribute) dimensions, any data that existed previously in the database must be mapped to a member of each new dimension before the outline can be saved. For example, adding a dimension called Channel to the Sample.Basic outline implies that all previous data in Sample.Basic is associated with a particular channel or the sum of all channels.

Saving an Outline with One or More Deleted Standard Dimensions

If you delete one or more standard (nonattribute) dimensions, the data associated with only one member of each deleted dimension must be retained and associated with a member of one of the other dimensions. For example, removing a dimension called Market from the outline implies that all of the data that remains in the database after the restructure operation is associated with a single, specified member of the Market dimension.

If you delete an attribute dimension, Essbase deletes the associations to its base dimension. See Working with Attributes.

Creating Sub-Databases Using Deleted Members

1. Delete a dimension from an existing outline.
2. Save the database using a different name, and specify the member to keep.

   Only one member can be kept when a dimension is deleted. See Saving an Outline with One or More Deleted Standard Dimensions.

5

Creating and Working With Duplicate Member Outlines

When you enable duplicate member names in an Essbase outline, then member names do not have to be unique. You can have multiple members using the same name in the outline. Learn how to manage duplicate member outlines and how to reference member names in a unique way when required.

- Creating Duplicate Member Names in Outlines
- Restrictions for Duplicate Member Names and Aliases in Outlines
- Syntax for Specifying Duplicate Member Names and Aliases

The information in this chapter applies to block storage and aggregate storage databases.

Creating Duplicate Member Names in Outlines

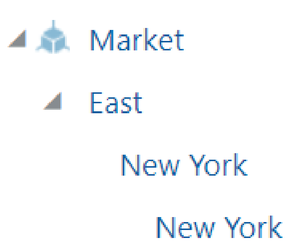
You can use duplicate names in an outline only if the outline has the allow duplicate members option enabled.

When you enable duplicate member names in an outline, Essbase displays multiple members in the outline using the same name. Create the names in the usual way. See Naming Conventions for Dimensions, Members, and Aliases.

When you save the outline, Essbase validates and saves the outline with the duplicate member names. A qualified name format differentiates the duplicate member names.

Figure 5-1 shows an example of a duplicate member outline in which the New York state member and the New York city member appear in the outline as New York.

Figure 5-1 Duplicate Member Name "New York"



The qualified member names for the example in Figure 5-1 are [State].[New York] and [City].[New York]. See Syntax for Specifying Duplicate Member Names and Aliases.

To create an outline that enables duplicate member names, use the create database MaxL statement with **using non_unique_members**. Or, if you build the cube from an application workbook, on the Cube.Settings tab, set the **Outline Type** property to **Duplicate**.

When you create a duplicate member outline, by default, all dimensions in the outline are tagged as duplicate.

When duplicate members are enabled in a dimension, you can tag particular generations or levels within the dimension as unique. If a member is assigned conflicting properties, the unique property takes precedence.

> **Note:**
>
> Duplicate member outline attribute dimensions do not have prefixes or suffixes attached that apply to attribute dimensions in unique outlines. For example, in a duplicate member Boolean attribute dimension, members do not include dimension, parent, grandparent, or ancestors affixed to the TRUE and FALSE members. See Setting Prefix and Suffix Formats for Member Names of Attribute Dimensions.

Restrictions for Duplicate Member Names and Aliases in Outlines

When creating duplicate member names and aliases in database outlines, the following must always be unique:

- Dimension names
- Generation names and level names
- Siblings under a parent member

If you are using aliases, this additional restriction applies: an alias table that contains duplicate alias names is valid only with a duplicate member outline.

> **Note:**
>
> Do not use quotation marks (" "), brackets ([ ]), or tabs in member, dimension, or alias names. For example, you cannot create a member name “[New York].[Area 1]”. Outline verification does not display an error for member names that contain the invalid sequence of characters, and you can save the outline; however, Essbase cannot accurately query the data.

Syntax for Specifying Duplicate Member Names and Aliases

Although duplicate member names appear in the outline, each nonshared member name uniquely identifies a member in the database. A qualified name format differentiates the duplicate member names. A qualified name must be used to specify a duplicate member name.

A qualified member or alias name can be specified in any of the following formats:

- Fully qualified member name
- Member name qualified by differentiating ancestor
- Shortcut qualified member name

> **Note:**
>
> A qualified name must comprise all alias names or all member names. You cannot mix member names and alias names in a qualified name.

Using Fully Qualified Member Names

A fully qualified member name comprises the duplicate member or alias name and *all* ancestors up to and including the dimension name. Each name must be enclosed in brackets ([ ]) and separated by a period (.). The syntax is as follows:

```
[DimensionMember].[Ancestors...].[DuplicateMember]
```

For example:

```
[Market].[East].[State].[New York]
[Market].[East].[City].[New York]
```

Qualifying Members by Differentiating Ancestor

A member name qualified by differentiating ancestor uses the member or alias name and *all* ancestors up to and including the ancestor that *uniquely* identifies the duplicate member or alias. The top ancestor in the path will always be a unique member name. Each name must be enclosed in brackets ([ ]) and separated by a period (.). The syntax is as follows:

```
[DifferentiatingAncestor].[Ancestors...].[DuplicateMember]
```

For example:

```
[State].[New York]
[City].[New York]
```

Using Shortcut Qualified Member Names

Essbase internally constructs shortcut qualified names for members in duplicate member outlines. You can manually insert shortcut qualified names into scripts, spreadsheets, or MDX queries.

Essbase uses the syntax shown below to construct shortcut qualified names. Using the same syntax that Essbase uses when you reference members in scripts, spreadsheets, and MDX queries is optimal but not required.

Table 5-1 Shortcut Qualified Name Syntax

Scenario	Qualified Name Syntax	Example
Duplicate member names exist at generation 2	[*DimensionMember*].[*DuplicateMember*]	`[Year].[Jan]` `[Product].[Jan]`

Table 5-1 (Cont.) Shortcut Qualified Name Syntax

Scenario	Qualified Name Syntax	Example
Duplicate member names exist in an outline but are unique within a dimension	`[`*DimensionMember*`]@[`*DuplicateMember*`]`	`[Year]@[Jan]`
Duplicate member names have a unique parent	`[`*ParentMember*`].[`*DuplicateMember*`]`	`[East].[New York]`
Duplicate member names exist at generation 3	`[`*DimensionMember*`].[`*ParentMember*`].[`*DuplicateMember*`]`	`[Products].[Personal Electronics].[Televisions]`
Duplicate member names exist at a named generation or level, and the member is unique at its generation or level	`[`*DimensionMember*`]@[`*GenLevelName*`]|[`*DuplicateMember*`]`	`[2006]@[Gen1]|[Jan]`
In some scenarios, the differentiating ancestor method is used as a shortcut.	`[`*DifferentiatingAncestor*`].[`*Ancestors...*`].[`*DuplicateMember*`]`	`[2006].[Qtr1].[Jan]`

Using Qualified Member Names in Unique Member Name Outlines

Qualified member names are also applicable for referencing member names in a unique member name outline (an outline with duplicate member names *not* enabled). Qualified member names can be used to differentiate shared members from their original members.

For example, in the Sample Basic database, the member [100-20] is an original member under parent [100], and has a shared member associated with it under parent [Diet]. The shared member [100-20] can be referred to explicitly, using the unique name [Diet].[100-20], as shown in the following query:

```
SELECT
 {Sales}
ON COLUMNS,
 {[[Diet]].[100-20]]]} PROPERTIES MEMBER_UNIQUE_NAME
ON ROWS
FROM Sample.Basic;
```

The double closing brackets in the example are needed as escape characters. For more details about the MDX syntax, see MDX Syntax for Specifying Duplicate Member Names and Aliases.

6

Setting Dimension and Member Properties

When you tag an Essbase dimension as a specific type, such as accounts, time, or attribute, the dimension can access built-in functionality designed for that type. You can set member properties to specify member consolidation, how data values are stored, member aliases, and UDAs. You can also create member formulas.

Some information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases. Also see Comparison of Aggregate and Block Storage.

Setting Dimension Types

When you tag a dimension as a specific type, the dimension can access built-in functionality designed for that type. For example, if you define a dimension as accounts, you can specify accounting capabilities for members in that dimension. Essbase calculates the two primary dimension types, time and accounts, before other dimensions in the database. By default, all dimensions are tagged as none.

To set a dimension type, see Setting Information Properties.

Creating a Time Dimension

Tag a dimension as time if it contains members that describe how often you collect and update data. In the Sample.Basic database, for example, the Year dimension is tagged as time. The time dimension also enables several functions on account dimension members, such as first and last time balances.

Rules when tagging a dimension as time:

- You can tag only one dimension in an outline as time.
- All members in the time dimension inherit the time property.
- You can add time members to dimensions that are not tagged as time.
- You can create an outline that does not have a time dimension.

To tag a dimension as time, see Understanding the Essbase.Cube Worksheet.

Creating an Accounts Dimension

Tag a dimension as accounts if it contains facts that you want to analyze, such as profit or inventory.

Rules when tagging a dimension as accounts:

- You can tag only one dimension in an outline as accounts.
- All members in the accounts dimension inherit the accounts property.
- You can specify that members of the accounts dimension are calculated on the second pass through an outline. See Two-Pass Calculation Property.
- You can create an outline that does not have an accounts dimension.

To tag a dimension as accounts, see Understanding the Essbase.Cube Worksheet.

The following sections describe built-in functionality for accounts dimensions.

- Setting Time Balance Properties
- Setting Skip Properties
- Setting Variance Reporting Properties

Setting Time Balance Properties

If an accounts dimension member uses the time balance property, it affects how Essbase calculates the parent of that member in the time dimension. By default, a parent in the time dimension is calculated based on the consolidation and formulas of its children. For example, in the Sample.Basic database, the Qtr1 member is the sum of its children (Jan, Feb, and Mar). However, setting a time balance property causes parents, for example Qtr1, to roll up differently.

To set time balance properties, see Understanding Dimension Worksheets.

Example 6-1 Example of Time Balance as None

None is the default value. When you set the time balance property as none, Essbase rolls up parents in the time dimension in the usual way—the value of the parent is based on the formulas and consolidation properties of its children.

Example 6-2 Example of Time Balance as First

Set the time balance as “first” when you want the parent value to represent the value of the first member in the branch (often at the beginning of a time period).

For example, assume that a member named OpeningInventory represents the inventory at the beginning of the time period. If the time period was Qtr1, OpeningInventory represents the inventory at the beginning of Jan; that is, the OpeningInventory for Qtr1 is the same as the OpeningInventory for Jan. For example, if you had 50 cases of Cola at the beginning of Jan, you also had 50 cases of Cola at the beginning of Qtr1.

Tag OpeningInventory as first, as shown in the following example consolidation:

```
OpeningInventory (TB First), Cola, East, Actual, Jan(+),  50
OpeningInventory (TB First), Cola, East, Actual, Feb(+),  60
OpeningInventory (TB First), Cola, East, Actual, Mar(+),  70
OpeningInventory (TB First), Cola, East, Actual, Qtr1(+), 50
```

Example 6-3 Example of Time Balance as Last

Set the time balance as “last” when you want the parent value to represent the value of the last member in the branch (often at the end of a time period).

For example, assume that a member named EndingInventory represents the inventory at the end of the time period. If the time period is Qtr1, EndingInventory represents the inventory at the end of Mar; that is, the EndingInventory for Qtr1 is the same as the EndingInventory for Mar. For example, if you had 70 cases of Cola at the end of Mar, you also had 70 cases of Cola at the end of Qtr1.

Tag EndingInventory as last, as shown in the following example consolidation:

```
EndingInventory (TB Last), Cola, East, Actual, Jan(+),  50
EndingInventory (TB Last), Cola, East, Actual, Feb(+),  60
EndingInventory (TB Last), Cola, East, Actual, Mar(+),  70
EndingInventory (TB Last), Cola, East, Actual, Qtr1(+), 70
```

Example 6-4 Example of Time Balance as Average

Set the time balance as “average” when you want the parent value to represent the average value of its children.

For example, assume that a member named AverageInventory represents the average of the inventory for the time period. If the time period was Qtr1, then AverageInventory represents the average of the inventory during Jan, Feb, and Mar.

Tag AverageInventory as average, as shown in the following example consolidation:

```
AverageInventory (TB Average), Cola, East, Actual, Jan(+),  60
AverageInventory (TB Average), Cola, East, Actual, Feb(+),  62
AverageInventory (TB Average), Cola, East, Actual, Mar(+),  67
AverageInventory (TB Average), Cola, East, Actual, Qtr1(+), 63
```

Setting Skip Properties

If you set the time balance as first, last, or average, set the skip property to tell Essbase what to do when it encounters missing values or values of 0.

Table 6-1 Skip Properties

Setting	Essbase Action
None	Does not skip data when calculating the parent value.
Missing	Skips #MISSING data when calculating the parent value.
Zeros	Skips data that equals zero when calculating the parent value.
Missing and Zeros	Skips #MISSING data and data that equals zero when calculating the parent value.

If you mark a member as last with a skip property of missing or missing and zeros, the parent of that time period matches the last nonmissing child. In the following example, EndingInventory is based on the value for Feb, because Mar does not have a value.

```
Cola, East, Actual, Jan, EndingInventory (Last), 60
Cola, East, Actual, Feb, EndingInventory (Last), 70
Cola, East, Actual, Mar, EndingInventory (Last), #MI
Cola, East, Actual, Qtr1, EndingInventory (Last),70
```

To set skip properties, see Understanding Dimension Worksheets.

Setting Variance Reporting Properties

Variance reporting properties determine how Essbase calculates the difference between actual and budget data in a member with the @VAR or @VARPER function in its member formula. Any member that represents an expense to the company requires an expense property.

When you are budgeting expenses for a time period, the actual expenses should be less than the budget. When actual expenses are greater than budget expenses, the variance is negative. The @VAR function calculates Budget – Actual. For example, if budgeted expenses are $100, and you spend $110, the variance is -10.

When you are budgeting nonexpense items, such as sales, the actual sales should be more than the budget. When actual sales are less than budget, the variance is negative. The @VAR function calculates Actual – Budget. For example, if budgeted sales were $100, and you made $110 in sales, the variance is 10.

By default, members are nonexpense.

To set variance reporting properties, see Setting Information Properties.

Creating Attribute Dimensions

Use attribute dimensions to report and aggregate data based on characteristics of standard dimensions. In the Sample.Basic database, for example, the Product dimension is associated with the Ounces attribute dimension. Members of the Ounces attribute dimension categorize products based on their size in ounces.

Review the rules for using attribute dimensions in Working with Attributes.

To tag a dimension as an attribute, see Understanding the Essbase.Cube Worksheet.

Examples of Member Consolidation

Essbase member consolidation properties control how data values roll up, from child members in the hierarchy to their parents. Examples follow for using most of the consolidation operators in outline member properties, including `+` (add), `-` (subtract), `*` (multiply), `/` (divide), `%` (percentage calculation), and `~` (no consolidation).

In these examples, assume that initially, the Essbase cube has not yet been calculated, and only level-0 members have values. These values have been loaded to the cube, but prior to calculation, the data has not yet been rolled up to parent members.

Consolidation using member operators happens in top-down order. As the default consolidation is addition, the (+) operator can be demonstrated first.

Example with (+) Operator

In this example below, P1 has no value before calculation. Its value will display as #MISSING if queried in the spreadsheet before the cube has been calculated.

```
P1
  M1 (+)  10
  M2 (+)  15
  M3 (+)  20
```

Once the cube has been calculated, P1 will have a value of 45. The consolidation proceeds top down, as follows:

```
P1 = P1+M1 = #MISSING+10 = 10
P1 = P1+M2 = 10+15 = 25
P1 = P1+M3 = 25+20 = 45
```

Example with (-) Operator

Now, consider a similar hierarchy with members that consolidate using subtraction.

```
P2
  M1 (-)  10
  M2 (-)  15
  M3 (-)  20
```

Once the cube has been calculated, P2 will have a value of -45. The consolidation proceeds top down, as follows:

```
P2 = P2-M1 = #MISSING-10 = -10
P2 = P2-M2 = -10-15 = -25
P2 = P2-M3 = -25-20 = -45
```

Example with (*) Operator

Consider a hierarchy with members that consolidate using multiplication.

```
P3
  M1 (*)  10
  M2 (*)  15
  M3 (*)  20
```

Once the cube has been calculated, P3 will have a value of #MISSING. The consolidation proceeds top down, as follows:

```
P3 = P3*M1 = #MISSING*10 = #MISSING
P3 = P3*M2 = #MISSING*15 = #MISSING
P3 = P3*M3 = #MISSING*20 = #MISSING
```

The final consolidated value of P3 may not have been the intended result. If the requirement is for P3 to be the multiplied product of all the child members, try using the (+) operator on the first child member:

```
P3
  M1 (+)  10
```

```
  M2 (*)  15
  M3 (*)  20
```

With the first operator changed to addition, P3 will have a non-#MISSING value after the cube is calculated. The consolidation proceeds top down, as follows:

```
P3 = P3+M1 = #MISSING+10 = 10
P3 = P3*M2 = 10*15 = 150
P3 = P3*M3 = 150*20 = 3000
```

Example with (/) Operator

Consider a hierarchy with members that consolidate using division (except for the first child member).

```
P4
  M1 (+)  10
  M2 (/)  15
  M3 (/)  20
```

Once the cube has been calculated, P4 will have a value of .033. The consolidation proceeds top down, as follows:

```
P4 = P4+M1 = #MISSING+10 = 10
P4 = P4/M2 = 10/15 = 0.666
P4 = P4/M3 = 0.666/20 = 0.033
```

Example with (%) Operator

Consider a hierarchy with members that consolidate using percentage calculation ([a/b]*100), except for the first child member.

```
P5
  M1 (+)  10
  M2 (%)  15
  M3 (%)  20
```

Once the cube has been calculated, P5 will have a value of 333. The consolidation proceeds top down, as follows:

```
P5 = P5+M1 = #MISSING+10 = 10
P5 = (P5/M2)*100 = (10/15)*100 = 66.6
P5 = (P5/M3)*100 = (66.6/20)*100 = 333
```

Example with Many Operators

Consider the following hierarchy that consolidates with a variety of operators.

```
Parent1
  Member1 (+)  10
  Member2 (+)  20
  Member3 (-)  25
  Member4 (*)  40
```

```
Member5 (%)   50
Member6 (/)   60
Member7 (~)   70
```

Essbase calculates Member1 through Member6 as follows:

```
Parent1 = Parent1+Member1 = #MISSING+10 = 10;
Parent1 = Parent1+Member2 = 10+20 = 30;
Parent1 = Parent1+Member3 = 30-25 = 5;
Parent1 = Parent1+Member4 =  5*40 = 200;
Parent1 = Parent1+Member5 = (200/50)*100 = 400
Parent1 = Parent1+Member6 = 400/60 = 6.666;
```

Because Member7 is set to No Consolidation(~), Essbase ignores Member7 in the consolidation. The final value of Parent1 is therefore 6.666.

See also some important considerations in Using the Calculation Operators *, /, and %.

Member Consolidation Properties

When you define the Essbase consolidation properties for outline members, you determine how child members roll up into their parents. By default, new members are given the addition (+) operator, meaning that members are added.

For example, Jan, Feb, and Mar figures are added, and the result stored in their parent, Qtr1.

> **Note:**
>
> Essbase does not use consolidation properties with members of attribute dimensions. See Calculating Attribute Data.

The member consolidation properties are listed below.

Table 6-2 Consolidation Operators

Operator	Description
+	Adds the member to the resulting value from calculations performed on prior members within the branch. + is the default operator.
-	Multiplies the member by –1 and adds it to the sum of previous calculations performed on other members.
*	Multiplies the member by the result of previous calculations performed on other members.
/	Divides the member into the result of previous calculations performed on other members.
%	Divides the member into the sum of previous calculations performed on other members. The result is multiplied by 100 to yield a percentage value.

Table 6-2 (Cont.) Consolidation Operators

Operator	Description
~	Does not use the member in the consolidation to its parent.
^	Does not use the member in any consolidation in any dimension except attribute dimensions.

To set member consolidation properties, see Understanding Dimension Worksheets.

Operation Results on #MISSING Values and Zero (0) Values

If a data value does not exist for a unique combination of members, Essbase gives the combination a value of `#MISSING`. A `#MISSING` value is different from a zero (0) value. Therefore, Essbase treats `#MISSING` values differently from 0 values.

The following tables shows how Essbase calculates `#MISSING` values. In this table, X represents any number.

Table 6-3 How Essbase Calculates Missing Values

Calculation/Operation	Result
X + `#MISSING`	X
X – `#MISSING`	X
`#MISSING` – X	-X
X * `#MISSING`	`#MISSING`
X / `#MISSING`	`#MISSING`
`#MISSING` / X	`#MISSING`
X / 0	`#MISSING`
X % `#MISSING`	`#MISSING`
`#MISSING` % X	`#MISSING`
X % 0	`#MISSING`
X == `#MISSING`	False, unless X is `#MISSING`
X != `#MISSING`	True, unless X is `#MISSING`
X <> `#MISSING`	True, unless X is `#MISSING`
(X <= `#MISSING`)	(X <=0)
(X >= `#MISSING`)	(X >=0) or (X == `#MISSING`)
(X > `#MISSING`)	(X > 0)
(X < `#MISSING`)	(X < 0)
X AND `#MISSING`**:**	`#MISSING`
1 AND `#MISSING` (1 represents any nonzero value)	0
0 AND `#MISSING`	`#MISSING`
`#MISSING` AND `#MISSING`	

Table 6-3 (Cont.) How Essbase Calculates Missing Values

Calculation/Operation	Result
X OR `#MISSING`**:**	1
1 OR `#MISSING` (1 represents any nonzero value)	`#MISSING`
0 OR `#MISSING`	`#MISSING`
`#MISSING` OR `#MISSING`	
IF (`#MISSING`)	IF (0)
f (`#MISSING`)	`#MISSING` for any Essbase function of one variable
f (X)	`#MISSING` for any X not in the domain of f, and any Essbase function of more than one variable (except where specifically noted)

Determining How Members Store Data Values

You can determine how and when Essbase stores the data values for a member. For example, you can tell Essbase to calculate the value for a member only when a user requests it, and then discard the data value. Each storage property is described in the following table.

Table 6-4 Choosing Storage Properties

Storage Property	Behavior
Store	Stores the data value with the member.
Dynamic Calc	Does not calculate the data value until a user requests it, and then discards the data value.
Label only	Label Only members are for grouping or labeling other members. No data is stored in the database for Label Only members.
Shared member	Shares values between members. For example, in the Sample.Basic database, the 100-20 member is stored under the 100 parent and shared under Diet parent.
Never share	Does not allow members to be shared implicitly. Members tagged as Never share can only be explicitly shared. To explicitly share a member, create the shared member with the same name and tag it as shared.

To set member storage properties, use application workbooks. See Understanding Dimension Worksheets.

Stored Members

Stored members contain calculated values that are stored with the member in the database after calculation. By default, members are set as stored.

Dynamic Calculation Members

When a member is Dynamic Calc, Essbase does not calculate the value for that member until a user requests it. After the user views it, Essbase does not store the value for that member.

Essbase automatically tags members of attribute dimensions as Dynamic Calc. You cannot change this setting.

See Dynamically Calculating Data Values.

Label Only Members

Label only members have no associated data. Use them to group members or to ease navigation and reporting from Smart View. Typically, you should give label only members the “no consolidation” property. See Member Consolidation Properties.

You cannot associate attributes with label only members. If you tag a base dimension member that has attribute associations as label only, Essbase removes the attribute associations and displays a warning message.

A descendent of a label only member cannot be tagged as Dynamic Calc. In the following example, when verifying the outline, Essbase issues an error message indicating that ChildB cannot be tagged as label only:

```
ParentA = Label Only
   ChildB = Label Only
      DescendantC = Dynamic Calc
```

Tagging DescendantC as Store Data resolves the issue.

Shared Members

Essbase shared members can help you calculate the same member across multiple parents in the outline, which can be useful when an item belongs to more than one category. The data values associated with a shared member come from another outline member with the same name, called the prototype member.

The shared member stores a pointer to data contained in the prototype member, and the data is stored only once. To define a member as shared, a prototype member of the same name must exist. For example, in the Sample.Basic database, the 100-20 member under 100 stores the data for that member. The 100-20 member under Diet points to that value.

Shared members typically are used to calculate the same member across multiple parents; for example, to calculate a Diet Cola member in both the 100 and Diet parents.

Using shared members lets you use members repeatedly throughout a dimension. Essbase stores the data value only once, but it displays in multiple locations. Storing the data value only once saves space and improves processing efficiency.

Read the following sections to learn more about shared members.

- Guidelines for Shared Members
- Shared Member Retrieval During Drill-Down
- Implied Shared Members

> **Note:**
>
> Members with the same name may be duplicate members instead of shared members. See Creating and Working With Duplicate Member Outlines.

Guidelines for Shared Members

Essbase shared members must have no children, UDAs, attributes, or formulas. They must be in the same dimension as their prototypes, and follow after them in the outline order. Shared members can have unique aliases, and they can have the same name as other members.

Follow these guidelines to avoid errors when creating shared members:

- Shared members must be in the same dimension as their prototype member. For example, both 100-20 members in the Sample.Basic database are in the Product dimension.
- The prototype member must precede the shared member in the outline order.
- Shared members cannot have children.
- An unlimited number of shared members can have the same name.
- UDAs or formulas cannot be assigned to shared members.
- You can create a shared member for a member with a duplicate member name.
- Attributes cannot be associated with shared members.
- If accounts properties are assigned to shared members, the values for those accounts properties are taken from the prototype member, even if the accounts properties on the shared member are changed.
- Aliases can be assigned to shared members.
- If an alias for a shared member is empty, Essbase uses the stored member's alias.
- If you assign an alias for a shared member and you change the alias for a prototype member, the shared member's alias doesn't change.
- A prototype member must be located before its shared member.
- Avoid complex relationships between prototype and shared members that will be part of an attribute calculation, or a calculation may return unexpected results. See Understanding Attribute Calculation and Shared Members.
- You cannot place a shared member and its prototype member under the same parent. In a duplicate member outline, siblings must be unique.
- In grid clients (for example, Smart View), shared members can easily be differentiated from their prototype members, because you can specify for them to be displayed with a qualified name (for example, [Parent].[Child]). Shared members can be displayed with qualified names even if you have not set the outline to enable duplicate member names. Additionally, you can use qualified member names to search for shared members in the grid or using member selection.

Shared Member Retrieval During Drill-Down

During zoom in (drill-down) operations, Essbase retrieves prototype members instead of shared members, when tavailable. Zooming out (drill-up) on a shared member retrieves the shared member parent rather than the prototype.

Essbase retrieves shared members during drill-down, depending on their location in the spreadsheet or grid client. Essbase follows these rules during grid retrieval:

- retrieves prototype members (not their shared members) by default.
- retrieves from the bottom of a spreadsheet first.
- If the parent of a shared member is a sibling of the prototype member of one of the shared members, retrieves the prototype member.

Example of Shared Members from a Single Dimension

If you create a test dimension with all shared members based on the prototype members of the dimension East from the Sample.Basic outline, the outline would be similar to the one shown in Figure 6-1:

Figure 6-1 Shared Members from a Single Dimension

```
Database:  (Current Alias Table: Default)
    Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Measures Accounts (Label Only)
    Product {Caffeinated, Ounces, Pkg Type, Intro Date }
    Market {Population }
        East (+) (UDAs: Major Market)
            New York (+) (UDAs: Major Market) {Population:21000000}
            Massachusetts (+) (UDAs: Major Market) {Population:9000000}
            Florida (+) (UDAs: Major Market) {Population:15000000}
            Connecticut (+) (UDAs: Small Market) {Population:6000000}
            New Hampshire (+) (UDAs: Small Market) {Population:3000000}
        West (+)
        South (+) (UDAs: Small Market)
        Central (+) (UDAs: Major Market)
        test (+)
            New York (+) (Shared Member)
            Massachusetts (+) (Shared Member)
            Florida (+) (Shared Member)
            Connecticut (+) (Shared Member)
            New Hampshire (+) (Shared Member)
    Scenario (Label Only)
```

Zooming out on a shared member returns the shared parent, not the prototype member. For example, if you zoom out on Florida, Essbase returns the shared parent, `test`.

If you retrieve only the children of East, all results are from prototype members because Essbase retrieves prototype members by default.

If, however, you retrieve data with the children of test above it in the spreadsheet, Essbase retrieves the shared members:

```
New York
Massachusetts
Florida
Connecticut
New Hampshire
test
```

If you move `test` above its last two children, Essbase retrieves the first three children as shared members, but the last two as prototype members. Similarly, if you insert a member in the middle of the list above which was not a sibling of the shared members (for example, California inserted between Florida and Connecticut), Essbase retrieves shared members only between the nonsibling and the parent (in this case, between California and test).

Example of Retrieval with Crossed Generation Shared Members

You can modify the Sample.Basic outline to create a shared member whose prototype member counterpart is a sibling to its own parent, as shown in Figure 6-2:

Figure 6-2 Retrieval with Crossed Generation Shared Members

```
Database:  (Current Alias Table: Default)
    Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Measures Accounts (Label Only)
    Product {Caffeinated, Ounces, Pkg Type, Intro Date }
    Market {Population }
        East (+) (UDAs: Major Market)
            New York (+) (UDAs: Major Market) {Population:21000000}
            Massachusetts (+) (UDAs: Major Market) {Population:9000000}
            Florida (+) (UDAs: Major Market) {Population:15000000}
            Connecticut (+) (UDAs: Small Market) {Population:6000000}
            New Hampshire (+) (UDAs: Small Market) {Population:3000000}
        West (+)
        South (+) (UDAs: Small Market)
        Central (+) (UDAs: Major Market)
        test (+)
            west (+) (Shared Member)
            New York (+) (Shared Member)
            Massachusetts (+) (Shared Member)
            Florida (+) (Shared Member)
            Connecticut (+) (Shared Member)
            New Hampshire (+) (Shared Member)
```

If you create a spreadsheet with shared members in this order, Essbase retrieves all the shared members, except it retrieves the prototype member West, not the shared member west:

```
West
New York
Massachusetts
Connecticut
New Hampshire
test
```

Essbase retrieves the members in this order because `test` is a parent of `west` and a sibling of its prototype member counterpart, `West`.

Implied Shared Members

Essbase outline members may be implicitly shared even when not specified as shared.

If implied sharing is enabled, parent members implicitly share the value of their single child members (when there is only one), or of their only consolidating child (when there is only one that consolidates, meaning it has an operator other than `~` or `^`).

Such parents are *implied shared members*, unless:

- they have a formula
- they are set to Never Share
- implied sharing is not enabled

By default, implied sharing is not enabled in Essbase 21c. See IMPLIED_SHARE_ON_CREATE configuration setting for more information, or to enable it.

If implied sharing is enabled, Essbase assumes (or implies) a shared member relationship in the following situations:

- **A parent has only one child.** In this situation, the parent and the child contain the same data. Essbase ignores the consolidation property on the child and stores the data only once—thus the parent has an implied shared relationship with the child. In the following example, the parent 500 has only one child, 500-10, so the parent shares the value of that child:

```
500 (+)
  500-10 (+)
```

- **A parent has only one child that consolidates to the parent.** If the parent has four children, but three are marked as no consolidation, the parent and child that consolidates contain the same data. Essbase ignores the consolidation property on the child and stores the data only once—thus the parent has an implied shared relationship with the child. In the following example, the parent 500 has only one child, 500-10, that rolls up to it. The other children are marked as No Consolidate(~), so the parent implicitly shares the value of 500-10.

```
500 (+)
  500-10 (+)
  500-20 (~)
  500-30 (~)
```

The cases above do not apply when the parent member has a formula.

If you do not want a member to be shared implicitly, mark the parent as Never Share so that the data is duplicated instead.

Setting Aliases

An alias is an alternate name for a member or shared member. Aliases, stored in alias tables, can improve the readability of outlines or reports.

For example, in the Sample.Basic cube outline, members in the Product dimension are identified both by product SKU codes, such as 100-10, and by multiple descriptive aliases, such as Cola. The following outline view shows some Product member names with their corresponding, default aliases:

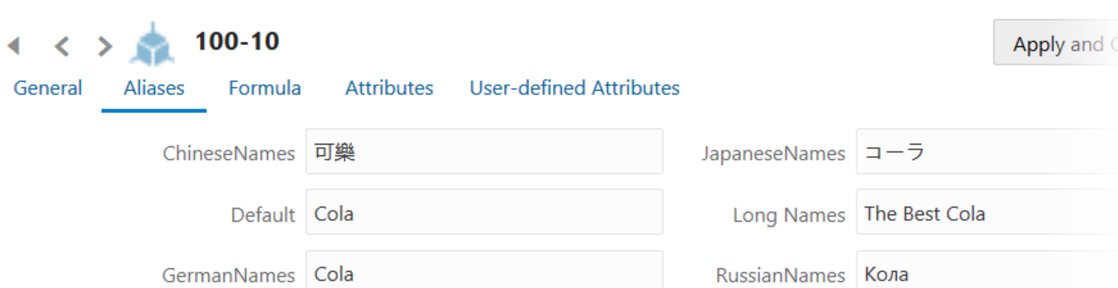
- Product <5>
  - 100 (+) <3> (Alias: Colas)
    - 100-10 (+) (Alias: Cola)
    - 100-20 (+) (Alias: Diet Cola)
    - 100-30 (+) (Alias: Caffeine Free Cola)
  - 200 (+) <4> (Alias: Root Beer)
  - 300 (+) <3> (Alias: Cream Soda)
  - 400 (+) <3> (Alias: Fruit Soda)
  - Diet (~) <3> (Alias: Diet Drinks)

In the following Smart View grid, the aliases are displayed alongside the member names.

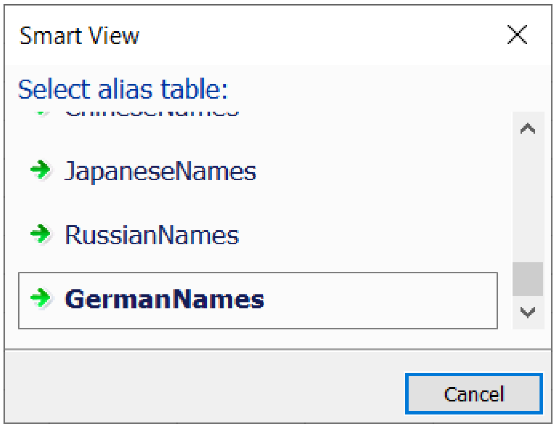
	A	B	C
1			**Scenario**
2			**East**
3			**Year**
4			**Measures**
5	**100**	**Colas**	12656
6	**200**	**Root Beer**	2534
7	**300**	**Cream Soda**	2627
8	**400**	**Fruit Soda**	6344
9	**Diet**	**Diet Drinks**	2408
10	**Product**	**Product**	24161

In the Smart View and outline views above, the aliases are from the default alias table defined in Sample Basic's outline. However, a member can have multiple aliases, if you add and populate more alias tables in addition to the built-in alias table named **Default**.

Consider the Product member 100-10 in Sample Basic. As Sample Basic has six alias tables, this member can be displayed using any of the aliases.



When multiple alias tables are implemented in the cube, Smart View users can click **Change Alias** on the Essbase ribbon to select a different set of aliases. A common use of multiple alias tables is for language conversion. Users in different countries can set the appropriate alias table to read member names in their language.



	A	B	C
1			**Szenario**
2			**Ost**
3			**Jahr**
4			**Umsätze**
5	**100**	**Cola Getränke**	12656
6	**200**	**Kohlensäurehaltige Getränke**	2534
7	**300**	**Milchgetränke**	2627
8	**400**	**Fruchtsaftgetränke**	6344
9	**Diet**	**Diätgetränke**	2408
10	**Product**	**Produkt**	24161

- Creating Aliases
- Working With Aliases and Alias Tables

Information about aliases and alias tables applies to block storage and aggregate storage cubes.

Creating Aliases

You can provide an alias for any member. Alias names must follow the same rules as for member names.

You can use any of the following methods to create aliases in an existing alias table:

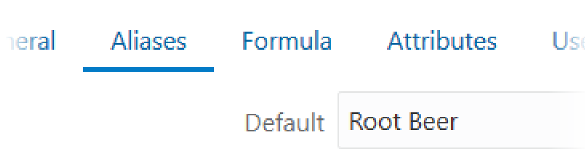
- Manually assign an alias to a member while editing an outline.
- Use Cube Designer and an application workbook to create alias tables and add aliases to them.
- Use dimension build and a data source to add aliases to an alias table.

- Import alias values from an alias table source file created in a predefined format.

Add Aliases in Outline Editor

To add an alias while editing an outline,

1. In the Essbase web interface, open the outline for editing.
2. Right click on a member name and click **Inspect**.
3. Click the **Aliases** tab.
4. Enter the alias name for the current member in one of the alias tables. If you have not defined any, you can enter the alias in the alias table named **Default**.



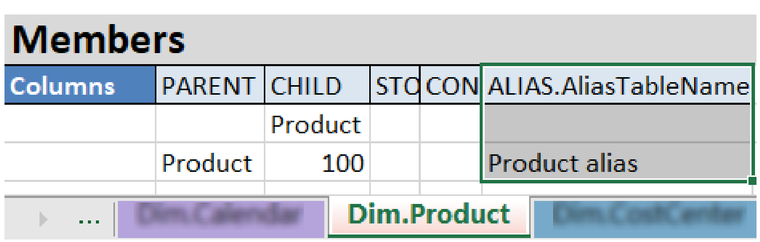
5. Click **Apply and Close**.
6. Save the outline.

Add Aliases in an Application Workbook

Using an application workbook, you build and populate alias tables when you build the cube.

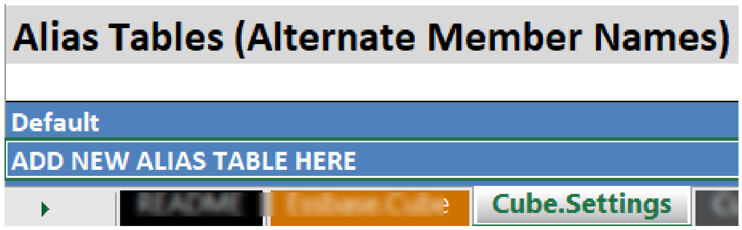
To add an alias using an application workbook,

1. Open the application workbook.
2. If the alias table already exists for the alias you want to add,
   a. Go to the appropriate dimension worksheet to add an alias. For example, to add an alias for a product, go to the worksheet for the product dimension.
   b. If this is the first alias you have added to this dimension and/or alias table, add a new column named ALIAS.<AliasTableName>, and add aliases to that column. If aliases already exist in this dimension and alias table, then simply add a new alias to the appropriate row for the member. For example, in the image below, a new column is added: ALIAS.AliasTableName, containing a new alias -- "Product alias" -- for product member 100.



3. If the alias table does not exist yet, specify a new alias table name in the Cube.Settings worksheet, adding it below the alias table named **Default**. Then return to the previous step to add aliases to dimensions.

   For example, in the image below, the space labeled ADD NEW ALIAS TABLE HERE is where you should enter a new alias table name.



4. Build the cube using the application workbook, either using Cube Designer or by importing the application workbook from the Essbase web interface.

See also Understand the Cube.Setting Worksheet: Alias Tables.

Add Aliases in a Dimension Build

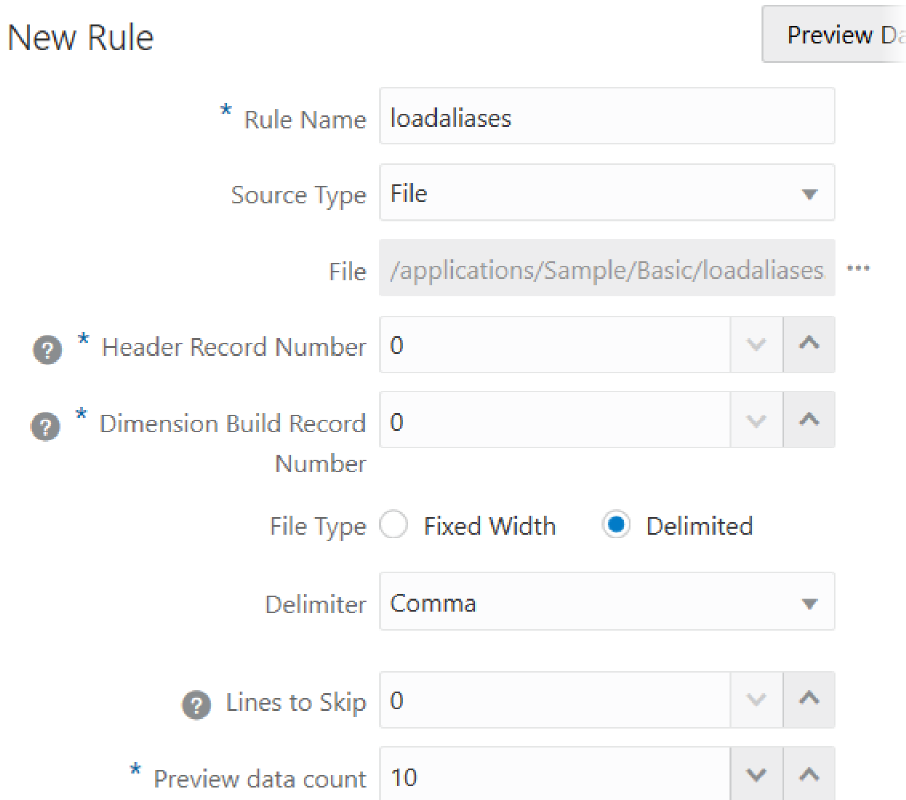
To add aliases in a dimension build, use the following workflow. This example works on Sample Basic, which already has an empty alias table called Long Names.

The following instructions assume you have already built or imported Sample Basic.

1. Create a dimension build data file. For example, the following data file is named `loadaliases.txt`:

```
"Product","100","Amazing Colas"
"100","100-10","The Best Cola"
"100","100-20","Minimalist Cola"
"100","100-30","Sleepy Cola"
"Product","200","Rockin Root Beers"
"200","200-10","Old Times Root Beer"
"200","200-20","Skinny Root Beer"
"200","200-30","Rooty Root Beer"
"200","200-40","Moms Birch Beer"
"Product","300","Crazy Cream Sodas"
"300","300-10","Milkies Dark Cream"
"300","300-20","Vanilla Dream Cream"
"300","300-30","Lean Cream"
"Product","400","Fruituristic Drinks"
"400","400-10","Grape Expectations"
"400","400-20","Orange You Glad"
"400","400-30","Strawberry Hill"
"Product","Diet","No Frills Drinks"
```

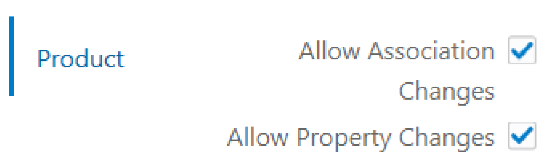
2. Upload the dimension build data file to the Sample Basic cube directory in the catalog.
3. Create a new dimension build rule (Regular), associated with the data file (`loadaliases.txt` in our example).



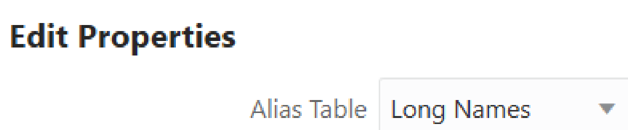
4. In the rule editor,
   a. In Field - 1, set **Dimension** to Product and set **Type** to Parent.
   b. In Field - 2, set **Type** to Child.
   c. In Field - 3, set **Type** to Alias.

Field - 1	Field - 2	Field - 3
Product	Product	Product
Parent	Child	Alias
Product	100	Amazing Colas
100	100-10	The Best Cola
100	100-20	Minimalist Cola
100	100-30	Sleepy Cola
Product	200	Rockin Root Beers
200	200-10	Old Times Root Beer
200	200-20	Skinny Root Beer
200	200-30	Rooty Root Beer
200	200-40	Moms Birch Beer
Product	300	Crazy Cream Sodas

5. Click **Dimensions**.
6. In the Edit Dimensions dialog, click the dimension name Product, and click **Advanced**.
7. Click **Allow Property Changes**, and click OK.

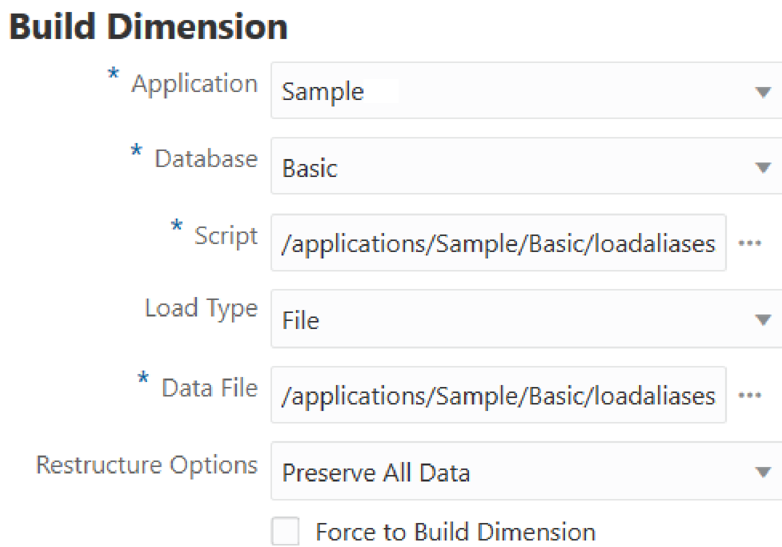


8. Click the **Global Properties** button.
9. Select the alias table to load to; for example, select Long Names, and click OK.

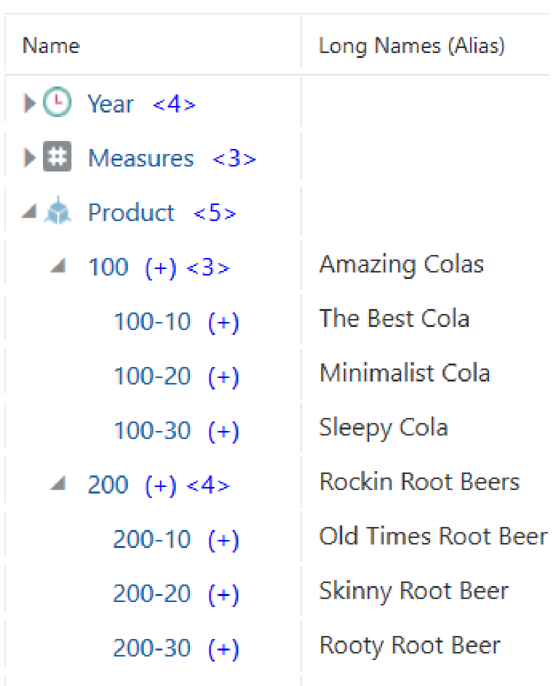


10. Verify, save, and close the rule.
11. Close the cube inspector and go to Jobs.
12. Click **New Job** and select **Build Dimension**.

13. Enter the application and database name.
14. Enter the rule file name in **Script**, and enter the data file name in **Data File**.



15. Click OK to begin the dimension build.
16. When the job completes, check the outline to see that the aliases were loaded. In the following view of the outline, the administrator has set Long Names aliases to display. To learn how to change the display, see Viewing Aliases in the Outline, in Working With Aliases and Alias Tables.



Working With Aliases and Alias Tables

Alias tables are containers for aliases.

Aliases are stored in one or more tables as part of a cube outline. An alias table maps a set of alias names to member names. Essbase cubes include one empty alias table named Default. If you do not add any other alias tables, the aliases that you create are stored in the Default alias table.

When you create a new alias table, it is empty. To make a new set of aliases available to users, you need to populate the alias table with aliases for some of the members.

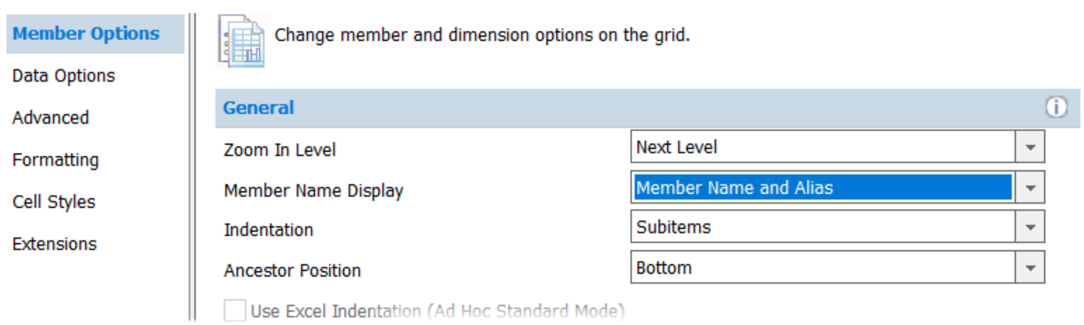
An alias table applies to all members in the outline, although you don't have to provide an alias name for every member unless you need it. You may use up to 56 alias tables if you require more than one name for any members in the outline.

Viewing Aliases in Smart View

If a cube has aliases defined for any members, Smart View users can decide whether and how to view the aliases.

To enable viewing aliases in Smart View,

1. On the Smart View ribbon, click **Options**.
2. In **Member Options** under the **General** group, select the **Member Name Display** option you want to use.

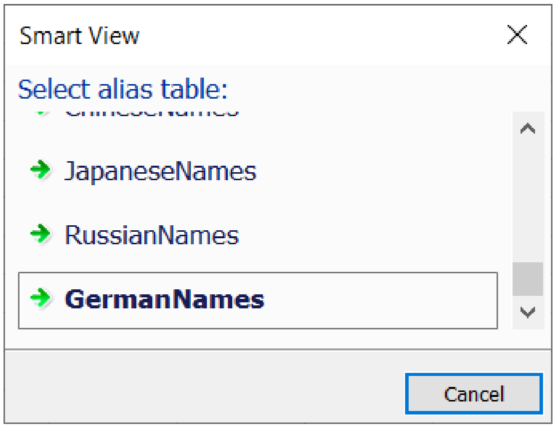


3. Click OK, and refresh the grid.

   In the following grid, the aliases are displayed alongside the member names.

	A	B	C
1			Scenario
2			East
3			Year
4			Measures
5	100	Colas	12656
6	200	Root Beer	2534
7	300	Cream Soda	2627
8	400	Fruit Soda	6344
9	Diet	Diet Drinks	2408
10	Product	Product	24161

To change the aliases you see in Smart View, click **Change Alias** on the Essbase ribbon to select a different set of aliases.

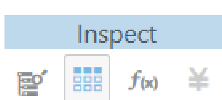


	A	B	C
1			**Szenario**
2			**Ost**
3			**Jahr**
4			**Umsätze**
5	**100**	**Cola Getränke**	12656
6	**200**	**Kohlensäurehaltige Getränke**	2534
7	**300**	**Milchgetränke**	2627
8	**400**	**Fruchtsaftgetränke**	6344
9	**Diet**	**Diätgetränke**	2408
10	**Product**	**Produkt**	24161

Viewing Aliases in the Outline

To customize your view of aliases in the outline viewer/editor in the Essbase web interface,

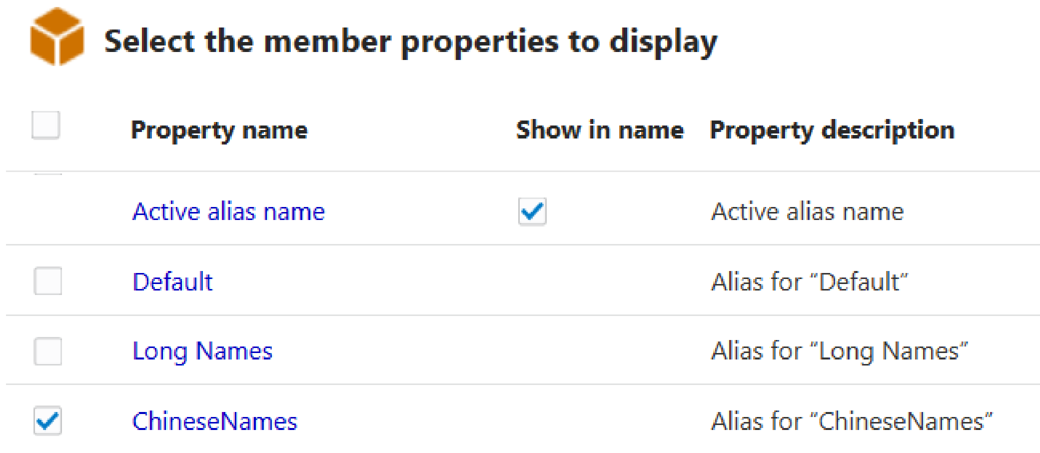
1. In the **Inspect** group, click **Display selected columns in the table**.

2. Select **Active alias name** if you want the outline hierarchy to display aliases next to the member name, using aliases in the currently active alias table.

   Select an alternate alias table in **Property name** if you want to also display another set of aliases in the outline viewer.

   For example, use the following display selections to make the outline viewer show current alias names and Chinese names.



3. Click **Apply and Close** to save your settings.

Viewing, Creating and Exporting Alias Tables

To view a list of alias tables in the outline, and to add an alias table,

- Using the Essbase web interface, work in the outline editor's **Aliases** tab to add and export alias tables. See Understand and Create Alias Tables.

  OR

- Using an application workbook, build alias tables when you build the cube. See Understand the Cube.Setting Worksheet: Alias Tables.

To export an alias table,

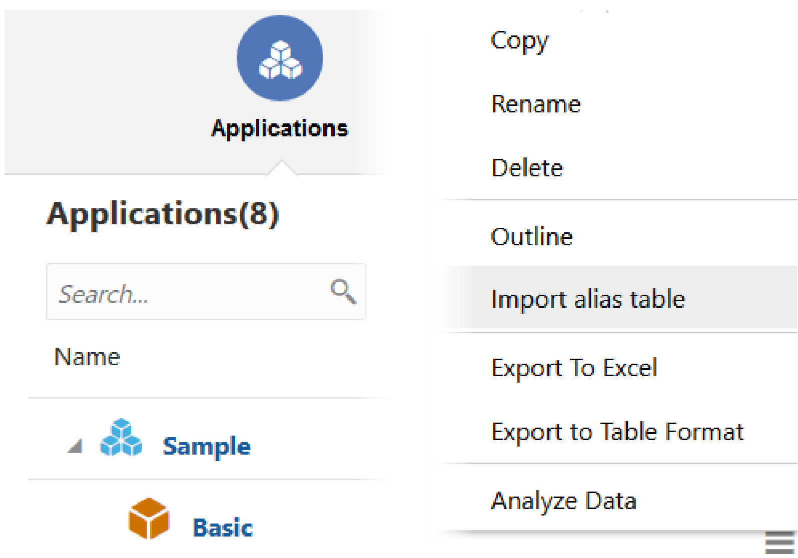
1. Using the Essbase web interface, open the outline, click **Outline Properties**, and go to the **Aliases** tab.
2. Next to the alias table you want to export, click .
3. By default, exported alias tables are saved as an `.alt` file in the user directory.

Import Alias Table from a File

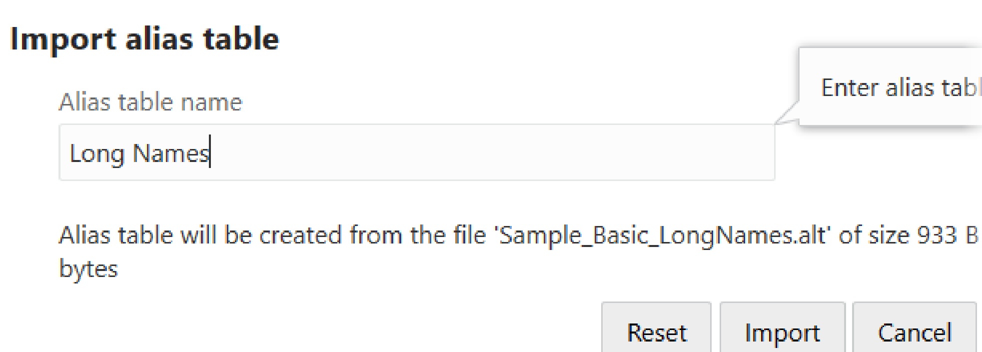
The following instructions are based on Sample Basic, and assume you already exported an alias table from another cube to your user directory. The exported alias table is saved in an `.alt` file.

To import aliases from an alias table `.alt` export file,

1. On the **Applications** page, expand the application (Sample).
2. From the **Actions** menu, to the right of the cube name (Basic), select **Import alias table**.



3. Select **Catalog**, browse to the location of the exported `.alt` file, select it, and click **Select**.
4. Enter a name for the alias table and click **Import**. Do not enter the name of an alias table that already exists for this cube; otherwise, an error is returned.



Clearing and Deleting Alias Tables

You can delete any alias table except for **Default**.

To delete an alias table,

1. Using the Essbase web interface, open the outline for editing, click **Outline Properties**, and go to the **Aliases** tab.
2. Next to the alias table you want to delete, click ✕.
3. Click **Apply and Close**, and save the outline.

If you want to remove individual aliases from an alias table without deleting the alias table,

1. Using the Essbase web interface, open the outline for editing.
2. Right click a member and select **Inspect**.
3. Click the **Aliases** tab.
4. Delete aliases from one or more of the alias tables.
5. Click **Apply and Close**, and save the outline.

Viewing Aliases in MaxL

In a MaxL session, setting an active alias table controls only which aliases display in the current MaxL session.

To view the current alias table in a MaxL session, use query database (**get active alias_table;**). To set the current alias table for a MaxL session, use alter database (**set active alias_table <*name*>;**). To enable viewing aliases in a MaxL session, use alter session (**set dml_output alias on;**)

The following example MaxL session demonstrates how to work with aliases in MaxL shell.

```
MAXL> query database Sample.Basic get active alias_table;

 active_alias_table
+----------------------
 Default

 OK/INFO - 1241044 - Records returned: [1].

MAXL> query database Sample.Basic list alias_table;

 list_of_aliases
+----------------------
 Default
 Long Names
 ChineseNames
 JapaneseNames
 RussianNames
 GermanNames

 OK/INFO - 1241044 - Records returned: [6].

MAXL> query database Sample.Basic list alias_names in alias_table "Long
Names";
```

```
 member                 alias
+----------------------+----------------------
 Qtr1                   Quarter1
 Jan                    January
 Feb                    February
 Mar                    March
 Qtr2                   Quarter2
 Apr                    April
 May                    May
 Jun                    Jun
 Qtr3                   Quarter3
 Jul                    July
 Aug                    August
 Sep                    September
 Qtr4                   Quarter4
 Oct                    October
 Nov                    November
 Dec                    December
 Margin                 Gross Margin
 Sales                  Revenue
 COGS                   Cost of Goods Sold
 Misc                   Miscelleneous

 OK/INFO - 1241044 - Records returned: [20].

MAXL> alter database Sample.Basic set active alias_table "JapaneseNames";

 OK/INFO - 1013273 - Database Sample.Basic altered.

MAXL> query database Sample.Basic get active
alias_table;

 active_alias_table
+----------------------
 JapaneseNames

MAXL> alter session set dml_output alias on;

 OK/INFO - 1056226 - Session altered for user [admin].

MAXL> SELECT {([West].children)} ON COLUMNS, {([100].children)} ON ROWS FROM
Sample.Basic;

 Axis-1              (カリフォルニ (オレゴン州)   (ワシントン州 (ユタ州)          (ネバ
ダ州)
+--------------------+-------------------+-------------------
+--------------------+-------------------+-------------------
 (コーラ)                        3498                  159
679                 275                   -18
 (ダイエットコ              -1587                 338
231                 398                    86
 (コーラ カフ                -912                 -57
#Missing                 459              #Missing

 OK/INFO - 1241150 - MDX Query execution completed.
```

```
MAXL>
```

Two-Pass Calculation Property

You can tag an accounts dimension member as two-pass to improve performance for some block storage Essbase applications.

> **Note:**
>
> Do not use two-pass calculation with hybrid mode cubes. Only use solve order.

By default, Essbase calculates outlines from the bottom up—first calculating the values for the children and then the values for the parent. Sometimes, however, the values of the children may be based on the values of the parent or the values of other members in the outline. To obtain the correct values for these members, Essbase must first calculate the outline and then recalculate the members that are dependent on the calculated values of other members. The members that are calculated on the second pass through the outline are called *two-pass calculations*.

For example, to calculate the ratio between Sales and Margin, Essbase needs first to calculate Margin, which is a parent member based on its children, including Sales. To ensure that the ratio is calculated based on a freshly calculated Margin figure, tag the Margin % ratio member as a two-pass calculation. Essbase calculates the database once and then calculates the ratio member again. This calculation produces the correct result.

Although two-pass calculation is a property that you can give to any nonattribute member, it works only on the following members:

- Accounts dimension members
- Dynamic Calc members

If two-pass calculation is assigned to other members, Essbase ignores it.

References

Two-Pass Calculation

Setting Two-Pass Calculation Properties.

Bottom-Up and Top-Down Calculation

Creating Formulas

You can apply formulas to standard dimensions and members. You cannot set formulas for attribute dimensions and their members. The formula determines how Essbase calculates the outline data. See Develop Formulas for Block Storage Cubes.

To add formulas to a dimension or member, see Understanding Dimension Worksheets and Creating Member Formulas.

Naming Generations and Levels

You can create names for generations and levels in an outline, such as a word or phrase that describes the generation or level. For example, you might create a generation name called Cities for all cities in the outline.

Use generation and level names in calculation scripts or report scripts wherever you need to specify either a list of member names or generation or level numbers. For example, you could limit a calculation in a calculation script to all members in a specific generation.

You can define only one name for each generation or level. When you name generations and levels, follow the same naming rules as for members, as described in Naming Conventions for Dimensions, Members, and Aliases.

To name generations and levels using the Essbase web interface, see Name Generations and Levels for instructions.

Creating UDAs

You can create user-defined attributes (UDA) for members. For example, you might create a UDA called Debit. Use UDAs in the following places:

- Calculation scripts. After you define a UDA, you can query a member for its UDA in a calculation script. For example, you can multiply all members with the UDA Debit by –1 so that they display as either positive or negative (depending on how the data is currently stored). See Develop Calculation Scripts for Block Storage Cubes.
- Data loading. You can change the sign of the data as it is loaded into the database based on its UDA.

To perform a calculation, selectively retrieve data based on attribute values, or provide full crosstab, pivot, and drill-down support in the spreadsheet, create attribute dimensions instead of UDAs. See Comparing Attributes and UDAs.

> **Note:**
>
> On aggregate storage databases, using UDAs to define member groups greatly decreases the execution speeds of Essbase functions. To avoid this performance loss, use attribute dimensions to define member groups.

Rules when creating UDAs:

- You can define multiple UDAs per member.
- You cannot set the same UDA twice for one member.
- You can set the same UDA for different members.
- A UDA name can be the same as a member, alias, level, or generation name. Follow the same naming rules as for members. See Naming Conventions for Dimensions, Members, and Aliases.
- You cannot create a UDA on shared members.
- You cannot create a UDA on members of attribute dimensions.

- A UDA applies to the specified member only. Descendants and ancestors of the member do not automatically receive the same UDA.

See Create User-Defined Attributes.

Adding Comments to Dimensions and Members

You can add comments to dimensions and members in a dimension worksheet in an application workbook that you plan to import to create a cube or in the cube outline. Comments can contain 255 characters maximum.

Member IDs

A member ID is a permanent, unique identifier for a member, separate from its name.

All members in an cube's outline have a member ID. Duplicate and unique member name outlines use member IDs, and so do aggregate storage and block storage outlines.

Having a persistent ID for each member helps Essbase perform efficient metadata restructuring. In unique member name outlines, the member names are all required to be unique, whereas in duplicate member name outlines, they are not. However, for all outline types, all members can be differentiated using their permanent member IDs.

You can assign member IDs yourself; otherwise, they will be auto assigned by Essbase. Member IDs are not changeable once they are assigned. Member IDs can be any alphanumeric string.

All members have member IDs, including attribute members. Member IDs exist for members in aggregate storage and block storage cube outlines (including hybrid mode).

You can assign member IDs during the following operations. If you do not, Essbase auto generates member IDs during these operations for any members that do not have them.

- Building the cube from an application workbook
- Building dimensions using a rule file
- Using batch outline editing to add or update a member
- Adding a member in the outline editor in the Essbase web interface
- Adding a member using API

If you migrate an existing outline using one of the Essbase migration tools, then member IDs are automatically assigned during migration. If you patch Essbase without migrating using one of the Essbase migration tools, then member IDs are assigned on the first successful edit of the outline after you apply the patch.

When Essbase auto generates member IDs, the IDs follow an incremental naming pattern: `id__0`, `id__1`, `id__2`, etc.

7 Working with Attributes

You can use attributes to describe characteristics of Essbase data, such as product size and color. Through attributes, you can group and analyze members of dimensions based on their characteristics.

- Process for Creating Attributes
- Understanding Attributes
- Understanding Attribute Dimensions
- Designing Attribute Dimensions
- Building Attribute Dimensions
- Setting Member Names in Attribute Dimensions
- Calculating Attribute Data
- Non-Aggregating Attributes
- Submitting Data for Valid Attribute Combinations in the Grid
- Suppressing Invalid Attribute Combinations in the Grid

Some information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases. Also see Comparison of Aggregate and Block Storage.

Process for Creating Attributes

1. Create a dimension.

   In an application workbook on the Essbase.Cube Worksheet, or in the outline editor, position the attribute dimensions after all standard dimensions.

2. Tag the dimension as an attribute dimension and set attribute dimension type as text, numeric, Boolean, or date.

   See Creating Attribute Dimensions.

3. Consider whether non-aggregating attributes would be useful in your database. See Non-Aggregating Attributes.
4. Add members to the attribute dimension.
5. Associate a base dimension with the attribute dimension.

   See Understanding the Rules for Attribute Dimension Association.

6. Associate members of the base dimension with members of the attribute dimension.

   See Understanding the Rules for Attribute Member Association.

7. If necessary, set up the attribute calculations.

   See Calculating Attribute Data.

See these topics:

- Understanding the Essbase.Cube Worksheet

- Understanding Cube.Settings Worksheet: Attribute Settings
- Working with Attributes
- Set Attribute Associations

Understanding Attributes

You can use the Essbase attribute feature to retrieve and analyze data not only from the perspective of dimensions, but also in terms of characteristics, or attributes, of those dimensions. For example, you can analyze product profitability based on size or packaging, and you can make more effective conclusions by incorporating market attributes, such as the population of each market region, into the analysis.

Such an analysis could tell you that decaffeinated drinks sold in cans in small markets (populations less than 6,000,000) are less profitable than you anticipated. For more details, you can filter the analysis by specific attribute criteria, including minimum or maximum sales and profits of different products in similar market segments.

A few ways analysis by attribute provides depth and perspective, supporting better-informed decisions:

- You can select, aggregate, and report on data based on common features (attributes).
- By defining attributes as having a text, numeric, Boolean, or date type, you can filter (select) data using type-related functions such as AND, OR, and NOT operators and <, >, and = comparisons.
- You can use the numeric attribute type to group statistical values by attribute ranges; for example, population groupings such as <500,000, 500,000–1,000,000, and >1,000,000.
- Through the Attribute Calculations dimension automatically created by Essbase, you can view sums, counts, minimum or maximum values, and average values of attribute data. For example, when you enter Avg and Bottle into a spreadsheet, Essbase retrieves calculated values for average sales in bottles for all the column and row intersections on the sheet.
- You can perform calculations using numeric attribute values in calculation scripts and member formulas; for example, to determine profitability by ounce for products sized by the ounce.
- You can create crosstabs of attribute data for the same dimension, and you can pivot and drill down for detail data in spreadsheets.

  An attribute crosstab is a report or spreadsheet showing data consolidations across attributes of the same dimension. The crosstab example below displays product packaging as columns and the product size in ounces as rows. At their intersections, you see the profit for each combination of package type and size.

  From this information, you can see which size-packaging combinations were most profitable in the Florida market.

```
Product Year Florida Profit Actual
        Bottle     Can       Pkg Type
        ======     =====     ========
  32      946        N/A        946
  20      791        N/A        791
  16      714        N/A        714
```

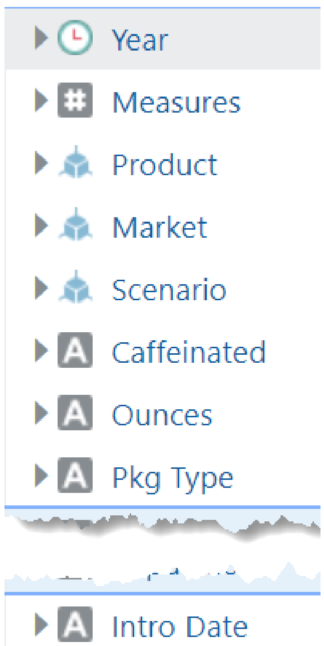
```
    12       241       2,383       2,624
Ounces     2,692       2,383       5,075
```

Understanding Attribute Dimensions

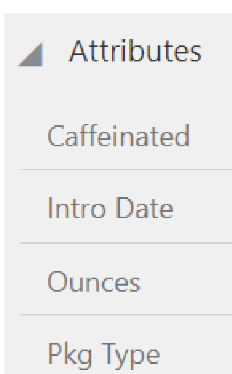
Products have attributes that are characteristics of the products, such as a product's size and packaging. Attribute members reside in attribute dimensions.

An *attribute dimension* has the letter A next to its name in the outline.

The image below shows part of the Sample.Basic outline featuring the Product dimension and some attribute dimensions: Caffeinated, Ounces, Pkg Type, and Intro Date.



In the Properties panel for the Product dimension, you can see which attribute dimensions are associated with the Product dimension.



Attributes are associated with sparse, non-attribute dimensions, called *base dimensions*. In the example above, the Product dimension is the base dimension for the attribute dimensions.

> **Note:**
>
> Attribute dimensions and members are Dynamic Calc (with the exception of non-aggregating attributes), so Essbase calculates attribute information at retrieval time. Attribute data is not stored in the database.

Understanding Members of Attribute Dimensions

Members of an attribute dimension are potential attributes of the members of the associated base dimension. After you associate a base dimension with an attribute dimension, you associate members of the base dimension with members of the associated attribute dimension. The Market dimension member Connecticut is associated with the 6000000 member of the Population attribute dimension. That makes 6000000 an attribute of Connecticut.

Understanding the Rules for Base and Attribute Dimensions and Members

Rules regarding members of attribute dimensions and their base dimensions:

- You can tag only sparse dimensions as attribute dimensions.
- Before you can save an outline to the server, each attribute dimension must be associated with a standard, sparse dimension as its base dimension.
- Attribute dimensions must be the last dimensions in the outline.
- Attribute dimensions have a type setting—text, numeric, Boolean, or date. Text is the default setting. Although assigned at the dimension level, the type applies only to the level 0 members of the dimension. See Understanding Attribute Types.
- If you remove the attribute tag from a dimension, Essbase removes prefixes or suffixes from its member names. Prefixes and suffixes are not visible in the outline. See Setting Prefix and Suffix Formats for Member Names of Attribute Dimensions.
- A base dimension member can have many attributes, but only one attribute from each attribute dimension.

  For example, product 100-10 can have size and packaging attributes, but only one size and only one type of packaging.

You can use attribute values in calculations in the following comparisons:

- > (greater than)
- >= (greater than or equal to)
- < (less than)
- <= (less than or equal to)
- == (equal to)
- <> or != (not equal to)
- IN

Understanding the Rules for Attribute Dimension Association

When you associate an attribute dimension with a standard dimension, the standard dimension is the base dimension for that attribute dimension.

- An attribute dimension must be associated with a sparse standard dimension.
- A standard dimension can be a base dimension for multiple attribute dimensions.
- An attribute dimension can be associated with only one base dimension.

  For example, you might have a Size attribute dimension with members Small, Medium, and Large. If you associate the Size attribute dimension with the Product dimension, you cannot also associate the Size attribute dimension with the Market dimension. Tracking size-related information for the Market dimension requires another attribute dimension with a different name; for example, MarketSize, with the MarketSize attribute dimension associated with the Market dimension.

Understanding the Rules for Attribute Member Association

When you associate a member of an attribute dimension with a member of a base dimension, follow these rules:

- You cannot associate multiple members from the same attribute dimension with the same base dimension member. For example, the Bottle and Can package types cannot both be associated with the product 100-30.
- You can associate members from different attribute dimensions with the same member of a base dimension. For example, a decaffeinated cola product (100-30) sold in 16-ounce bottles has three attributes: Caffeinated:False; Ounces:16; and Pkg Type:Bottle.
- Essbase does not require that each member of a base dimension be associated with a member of an attribute dimension.
- Attributes can be assigned only to level zero members.
- The level 0 members of attribute dimensions are the only members that you can associate with base dimension members.

  For example, in the Population attribute dimension, you can associate only level 0 members such as 3000000, 6000000, and 9000000, with members of the Market dimension. You cannot associate a level 1 member such as Small.

  The name of the level 0 member of an attribute dimension is the attribute value. The only members of attribute dimensions that have attribute values are level 0 members.

  You can use the higher-level members of attribute dimensions to select and group data. For example, you can use Small, the level 1 member of the Population attribute dimension, to retrieve sales in both the 3000000 and 6000000 population categories.

Understanding Attribute Types

Attribute dimensions have a text, numeric, Boolean, or date type that enables different functions for grouping, selecting, or calculating data. Although assigned at the dimension level, the attribute type applies only to level 0 members of the attribute dimension.

- The default attribute type is text. Text attributes enable the basic attribute member selection and attribute comparisons in calculations. When you perform such comparisons, Essbase compares characters. For example, the package type Bottle is less than the package type Can, because B precedes C in the alphabet. In Sample.Basic, Pkg Type is an example of a text attribute dimension.

- The names of level 0 members of numeric attribute dimensions are numeric values. You can include the names (values) of numeric attribute dimension members in calculations. For example, you can use the number of ounces specified in the Ounces attribute to calculate profit per ounce for each product.

  You can also associate numeric attributes with ranges of base dimension values; for example, to analyze product sales by market population groupings—states with 3,000,000 population or less in one group, states with a population between 3,000,001 and 6,000,000 in another group, and so on. See Setting Up Member Names Representing Ranges of Values.
- All Boolean attribute dimensions in a database contain only two members. The member names must match the settings for the database; for example, True and False. If multiple Boolean attribute dimensions exist, specify a prefix or suffix member name format to ensure unique member names; for example, Caffeinated_True and Caffeinated_False. See Setting Boolean Attribute Member Names.
- You can use date attributes to specify the date format—month-day-year or day-month-year—and to sequence information accordingly. See Changing the Member Names in Date Attribute Dimensions. You can use date attributes in calculations. For example, you can compare dates in a calculation that selects product sales from markets established since 10-12-1999.

  Essbase supports date attributes from January 1, 1970, through January 1, 2038.

Comparing Attribute and Standard Dimensions

In general, attribute dimensions and their members are similar to standard dimensions and their members. You can provide aliases and member comments for attributes. Attribute dimensions can include hierarchies, and you can name generations and levels. You can perform the same spreadsheet operations on attribute dimensions and members as on standard dimensions and members; for example, to analyze data from different perspectives, you can retrieve, pivot, and drill down in the spreadsheet.

The followign table describes major differences between attribute and standard dimensions and their members.

Table 7-1 Differences Between Attribute and Standard Dimensions

Functionality	Attribute Dimensions	Standard Dimensions
Storage	Sparse. An attribute dimension's base dimension also must be sparse	Can be dense or sparse
Storage property	Can be Dynamic Calc only. Therefore, not stored in the database. The outline does not display this property.	Can be Store Data, Dynamic Calc, or Label Only
Position in outline	Must be the last dimensions in the outline	Must be above all attribute dimensions in the outline
Partitions	Cannot be defined along attribute dimensions, but you can use attributes to define a partition on a base dimension	Can be defined along standard dimensions
Formulas (on members)	Cannot be associated	Can be associated
Shared members	Not allowed	Allowed

Table 7-1 (Cont.) Differences Between Attribute and Standard Dimensions

Functionality	Attribute Dimensions	Standard Dimensions
Two-pass calculation member property	Not available	Available
Two-pass calculation with runtime formula	If a member formula contains a runtime-dependent function associated with an attribute member name, and the member with the formula is tagged as two-pass, calculation skips the member and issues a warning message. Runtime-dependent functions include: @CURRMBR, @PARENT, @PARENTVAL, @SPARENTVAL, @MDPARENTVAL, @ANCEST, @ANCESTVAL, @SANCESTVAL, and @MDANCESTVAL.	Calculation is performed on standard members with runtime formulas and tagged two-pass.
Two-pass, multiple dimensions: Calculation order	Order of calculation of members tagged two-pass depends on order in outline. The last dimension is calculated last.	Calculation result is not dependent on outline order for members tagged two-pass in multiple dimensions.
Two-pass calculation with no member formula	Calculation skipped, warning message issued. Therefore, member intersection of two-pass tagged members and upper-level members may return different results from calculation on standard dimensions.	Available
Dense Dynamic Calc members in nonexisting stored blocks	Calculations skip dense dimensions if they are on nonexisting stored blocks. For attributes to work on dense members, data blocks for the dense members must exist. When retrieving data on a dense member that has a Dynamic Calc formula and no attributes, Essbase dynamically creates the data block and returns a value. However, if the Dynamic Calc dense member has an attribute, doing a retrieve on the attribute member results in #MISSING, because Essbase skips the dynamic calculation on the dense member and, therefore, the data block is not created. To identify nonexisting stored blocks, export the database or run a query to find out whether the block has data.	Available
UDAs on members	Not allowed	Allowed
Consolidations	For all members, calculated through the Attribute Calculations dimension members: Sum, Count, Min, Max, and Avg. Consolidation operators in the outline are ignored during attribute calculations.	Consolidation operation indicated by assigning the desired consolidation symbol to each member
Member selection facilitated by Level 0 member typing	Available types include text, numeric, Boolean, and date.	All members treated as text
Associations	Must be associated with a base dimension	N/A

Table 7-1 (Cont.) Differences Between Attribute and Standard Dimensions

Functionality	Attribute Dimensions	Standard Dimensions
Spreadsheet drill-downs	List the base dimension data associated with the selected attribute. For example, drilling down on the attribute Glass displays sales for each product packaged in glass, where Product is the base dimension for the Pkg Type attribute dimension.	List lower or sibling levels of detail in the standard dimensions. For example, drilling down on Qtr1 displays a list of products and their sales for that quarter.

Solve Order and Attributes

If hybrid mode is enabled, you can set the solve order for attribute dimensions and their base dimensions, eliminating the need to tag members as two-pass. In hybrid mode, the default calculation order (also known as solve order) matches that of block storage databases, with some enhancements. If you wish to use a non-default solve order, you can set a custom solve order for dimensions and members.

For more information about hybrid mode, see Adopt Hybrid Mode for Fast Analytic Processing.

Understanding Two-Pass Calculations on Attribute Dimensions

The following example, based on the Product dimension in the Sample.Basic database, illustrates how two-pass calculations work on attribute dimensions. Assume member “400–30” is tagged as two-pass.

If member “400-30” has the following member formula:

```
=”400-10”;
```

Essbase executes the formula when performing a retrieve on “400-30.”

If “400-30” has the following member formula:

```
=@CURRMBR("Market");
```

Essbase skips the calculation because the formula includes the @CURRMBR runtime function, which is not allowed, and issues the following error message:

```
Two-pass calc skipped on member [400-30] in attribute calc”
```

If “400-30” does not have a member formula, the same error message is generated because a member tagged as two-pass must have a formula.

Comparing Attributes and UDAs

Attributes and UDAs enable analysis based on characteristics of the data. Attributes provide greater capability than UDAs. The tables in this topic describe the differences between attributes and UDAs in these areas of functionality:

- Data storage

- Data retrieval
- Data conversion
- Calculation scripts

Table 7-2 Data Storage—Comparing Attributes and UDAs

Data storage	Attributes	UDAs
You can associate with sparse dimensions.	Supported	Supported
You can associate with dense dimensions.	Not supported	Supported

Table 7-3 Data Retrieval—Comparing Attributes and UDAs

Data Retrieval	Attributes	UDAs
You can group and retrieve consolidated totals by attribute or UDA value. For example, associate the value High Focus Item to various members of the Product dimension and use that term to retrieve totals and details for only those members.	Supported Simple	Supported More difficult to implement, requiring additional calculation scripts or commands
You can categorize attributes in a hierarchy and retrieve consolidated totals by higher levels in the attribute hierarchy; for example, if each product has a size attribute such as 8, 12, 16, or 32, and the sizes are categorized as small, medium, and large. You can view the total sales of small products.	Supported	Supported More difficult to implement
You can create crosstab views displaying aggregate totals of attributes associated with the same base dimension.	Supported You can show a crosstab of all values of each attribute dimension.	Not supported You can retrieve only totals based on specific UDA values.
You can use Boolean operators AND, OR, and NOT with attribute and UDA values to refine a query. For example, you can select decaffeinated drinks from the 100 product group.	Supported	Supported
Because attributes have a text, Boolean, date, or numeric type, you can use appropriate operators and functions to work with and display attribute data. For example, you can view sales totals of all products introduced after a specific date.	Supported	Not supported

Table 7-3 (Cont.) Data Retrieval—Comparing Attributes and UDAs

Data Retrieval	Attributes	UDAs
You can group numeric attributes into ranges of values and let the dimension building process automatically associate the base member with the appropriate range. For example, you can group sales in various regions based on ranges of their populations—less than 3 million, between 3 million and 6 million, and so on.	Supported	Not supported
Through the Attribute Calculations dimension, you can view aggregations of attribute values as sums, counts, minimums, maximums, and averages.	Supported	Not supported
You can use an attribute in a calculation that defines a member. For example, you can use the weight of a product in ounces to define the profit per ounce member of the Measures dimension.	Supported	Not supported
You can retrieve specific base members using attribute-related information.	Supported Powerful conditional and value-based selections	Supported Limited to text string matches only

Table 7-4 Data Conversion—Comparing Attributes and UDAs

Data Conversion	Attributes	UDAs
Based on the value of a UDA, you can change the sign of the data as it is loaded into the database. For example, you can reverse the sign of all members with the UDA Debit.	Not supported	Supported

Table 7-5 Calculation Scripts—Comparing Attributes and UDAs

Calculation Scripts	Attributes	UDAs
You can perform calculations on a member if its attribute or UDA value matches a specific value. For example, you can increase the price by 10% of all products with the attribute or UDA of Bottle.	Supported	Supported
You can perform calculations on base members whose attribute value satisfies conditions that you specify. For example, you can calculate the Profit per Ounce of each base member.	Supported	Not supported

Designing Attribute Dimensions

Essbase provides multiple ways to design attribute information into a database. Most often, defining characteristics of the data through attribute dimensions and their members is the best approach. The following sections discuss when to use attribute dimensions, when to use other features, and how to optimize performance when using attributes.

Using Attribute Dimensions

For the most flexibility and functionality, use attribute dimensions to define attribute data. Using attribute dimensions provides the following features:

- Sophisticated, flexible data retrieval

  You can view attribute data only when you want to; you can create meaningful summaries through crosstabs; and, using type-based comparisons, you can selectively view only the data that you want to see.
- Additional calculation functionality

  Not only can you perform calculations on the names of members of attribute dimensions to define members of standard dimensions, you can also access five types of consolidations of attribute data—sums, counts, averages, minimums, and maximums.
- Economy and simplicity

  Because attribute dimensions are sparse, Dynamic Calc, they are not stored as data. Compared to using shared members, outlines using attribute dimensions contain fewer members and are easier to read.

See Understanding Attributes.

Using Alternative Design Approaches

In some situations, consider one of the following approaches:

- Hybrid mode. If you implement hybrid mode, you can use non-aggregating attributes if needed, and you can set a custom solve order instead of using two-pass calculation. See Non-Aggregating Attributes. For more information about hybrid mode, see Adopt Hybrid Mode for Fast Analytic Processing.
- UDAs. Although UDAs provide less flexibility than attributes, you can use them to group and retrieve data based on its characteristics. See Comparing Attributes and UDAs.
- Shared members. For example, to include a seasonal analysis in the Year dimension, repeat the months as shared members under the appropriate season; Winter: Jan (shared member), Feb (shared member), and so on. A major disadvantage of using shared members is that the outline becomes large if the categories repeat many members.
- Standard dimensions and members. Additional standard dimensions provide flexibility but add storage requirements and complexity to a database. For guidelines on evaluating the impact of additional dimensions, see Analyzing and Planning.

The table below describes situations in which you might consider an alternative approach to managing attribute data in a database.

Table 7-6 Considering Alternatives to Attribute Dimensions

Situation	Alternative
Analyze attributes of dense dimensions	UDAs or shared members
Perform batch calculation of data	Shared members or members of separate, standard dimensions
Define the name of a member of an attribute dimension as a value that results from a formula	Shared members or members of separate, standard dimensions
Define attributes that vary over time	Members of separate, standard dimensions. For example, to track product maintenance costs over time, the age of the product at the time of maintenance is important. However, using the attribute feature, you could associate only one age with the product. You need multiple members in a separate dimension for each time period that you want to track.
Minimize retrieval time with large numbers of base-dimension members	Batch calculation with shared members or members of separate, standard dimensions.
Perform cross-dimensional analysis with low performance impact	Non-aggregating attributes

Optimizing Outline Performance

Outline layout and content can affect attribute calculation and query performance. For general outline design guidelines, see Designing an Outline to Optimize Performance.

To optimize attribute query performance, consider the following design tips:

- Ensure that attribute dimensions are the only sparse Dynamic Calc dimensions in the outline.
- Locate sparse dimensions after dense dimensions in the outline. Place the most-queried dimensions at the beginning of the sparse dimensions and attribute dimensions at the end of the outline. In most situations, base dimensions are queried most.

See Optimizing Calculation and Retrieval Performance.

Building Attribute Dimensions

To build an attribute dimension, tag the dimension as an attribute and assign the dimension a type. Then associate the attribute dimension with a base dimension. Finally, associate each level 0 member of the attribute dimension with a member of the associated base dimension.

You can view the dimension, attribute value, and attribute type of a specific attribute member using the outline viewer in the Essbase web interface, or by using the query database MaxL statement.

See these topics:

- Understand the Essbase.Cube Worksheet
- Understand Cube.Settings Worksheet: Attribute Settings
- Work with Attributes
- Set Attribute Associations

Setting Member Names in Attribute Dimensions

When you use the attribute feature, Essbase establishes default member names; for example, the system-defined True and False precludes other member names of True and False. You can change these system-defined names for the database. Date attributes and numeric attributes also can be duplicated. To avoid duplicate name confusion, you can establish settings for qualifying member names in attribute dimensions. The outline does not show the fully qualified attribute names, but you can see the full attribute names anywhere you select members, such as when you define partitions or select information to be retrieved.

Define the member name settings before you define or build the attribute dimensions. Changing the settings after the attribute dimensions and members are defined could result in invalid member names.

The following sections describe how to work with the names of members of attribute dimensions:

- Setting Prefix and Suffix Formats for Member Names of Attribute Dimensions
- Setting Boolean Attribute Member Names
- Changing the Member Names in Date Attribute Dimensions
- Setting Up Member Names Representing Ranges of Values
- Changing the Member Names of the Attribute Calculations Dimension

> **Note:**
>
> If you partition an outlines containing attribute dimensions, the name format settings of members described in this section must be identical in the source and target outlines.

Setting Prefix and Suffix Formats for Member Names of Attribute Dimensions

The information in this section does not apply to duplicate member attribute dimensions.

The names of members of Boolean, date, and numeric attribute dimensions are values. It is possible to encounter duplicate attribute values in different attribute dimensions.

- Boolean example

  If you have multiple Boolean attribute dimensions in an outline, the two members of each of those dimensions have the same names, by default, True and False.

- Date example

  If you have multiple date attribute dimensions, some member names in both dimensions could be the same. For example, the date on which a store opens in a certain market could be the same as the date on which a product was introduced.

- Numeric example

  The attribute value for the size of a product could be 12, and 12 also could be the value for the number of packing units for a product. This example results in two members with the name 12.

You can define unique names by attaching a prefix or suffix to member names in Boolean, date, and numeric attribute dimensions in the outline. You can affix the dimension, parent, grandparent, or all ancestors to the attribute name. For example, by setting member names of attribute dimensions to include the dimension name as the suffix, attached by an underscore, the member value 12 in the Ounces attribute dimension assumes the unique, full attribute member name 12_Ounces.

By default, Essbase assumes that no prefix or suffix is attached to the names of members of attribute dimensions.

The convention that you select applies to the level 0 member names of all numeric, Boolean, and date attribute dimensions in the outline. You can define aliases for these names if you want to display shorter names in retrievals.

See Understanding Cube.Settings Worksheet: Attribute Settings.

Setting Boolean Attribute Member Names

When you set the dimension type of an attribute dimension as Boolean, Essbase automatically creates two level 0 members with the names specified for the Boolean attribute settings. The initial Boolean member names in a database are set as True and False. To change these default names, for example, to Yes and No, define the member names for Boolean attribute dimensions before you create Boolean attribute dimensions in the database.

Before you can set an attribute dimension type as Boolean, you must delete all existing members in the dimension.

See Understanding Cube.Settings Worksheet: Attribute Settings.

Changing the Member Names in Date Attribute Dimensions

You can change the format of members of date attribute dimensions. For example, you can use the following date formats:

- mm-dd-yyyy: October 18, 2007 is displayed as 10-18-2007.
- dd-mm-yyyy: October 18, 2007 is displayed as 18-10-2007.

If you change the date member name format, the names of existing members of date attribute dimensions may be invalid. For example, if the 10-18-2007 member exists, and you change the format to dd-mm-2007, outline verification will find this member invalid. If you change the date format, you must rebuild the date attribute dimensions.

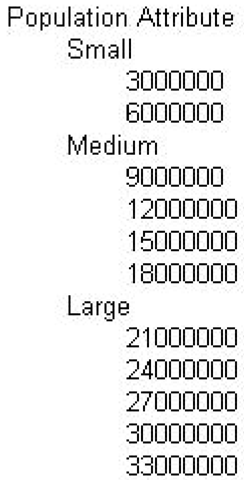
See Understanding Cube.Settings Worksheet: Attribute Settings.

Setting Up Member Names Representing Ranges of Values

Members of numeric attribute dimensions can represent single numeric values or ranges of values:

- Single-value example: the member 12 in the Ounces attribute dimension represents the single numeric value 12; you associate this attribute with all 12-ounce products. The outline includes a separate member for each size; for example, 16, 20, and 32.
- Range of values example: the Population attribute dimension, as shown in Figure 7-1:

Figure 7-1 Population Attribute Dimension and Members



In this outline, the members of the Population attribute dimension represent ranges of population values in the associated Market dimension. The 3000000 member represents populations from zero through 3,000,000; the 6000000 member represents populations from 3,000,001 through 6,000,000; and so on. A setting for the outline establishes that each numeric member represents the top of its range.

You can also define this outline setting so that members of numeric attribute dimensions are the bottoms of the ranges that they represent. For example, if numeric members are set to define the bottoms of the ranges, the 3000000 member represents populations from 3,000,000 through 5,999,999, and the 6000000 member represents populations from 6,000,000 through 8,999,999.

When you build the base dimension, Essbase automatically associates members of the base dimension with the appropriate attribute range. For example, if numeric members represent the tops of ranges, Essbase automatically associates the Connecticut market, with a population of 3,269,858, with the 6000000 member of the Population attribute dimension.

In the dimension build rules file, specify the size of the range for each member of the numeric attribute dimension. In the above example, each attribute represents a range of 3,000,000.

> **Note:**
>
> Oracle recommends that numeric attribute dimension member names contain no more than six decimal positions. Otherwise, because of precision adjustments, an outline may not pass verification.

See Understanding Cube.Settings Worksheet: Attribute Settings.

Changing the Member Names of the Attribute Calculations Dimension

To avoid duplicating names in an outline, you may need to change the name of the Attribute Calculations dimension or its members.

Sum, Count, Min, Max, and Avg—the names of the members that Essbase creates in the Attribute Calculations dimension—are not considered reserved words because you can change these names in the Attribute Calculations dimension and then use the default name in an attribute or standard dimension. Follow these guidelines:

- If the outline is tagged as a duplicate member outline, you can use the default names to name other base or attribute members.
- If the outline is tagged as a unique member outline, you should avoid using Sum, Count, Min, Max, and Avg as member names. For example, if you use Max in a standard dimension and then create an attribute dimension, in which Essbase creates the Max member in the Attribute Calculations dimension, Essbase detects a duplicate name and returns an error message indicating the name is already in use.

Regardless of the name that you use for a Attribute Calculations dimension member, its function remains the same. For example, Count, the second member, always counts.

See Understanding Cube.Settings Worksheet: Attribute Settings.

Calculating Attribute Data

Essbase calculates attribute data dynamically at retrieval time, using members from a system-defined dimension created by Essbase. Using this dimension, you can apply different calculation functions, such as a sum or an average, to the same attribute. You can also perform specific calculations on members of attribute dimensions; for example, to determine profitability by ounce for products sized by the ounce.

The following information assumes that you understand the concepts of attribute dimensions and Essbase calculations, including dynamic calculations. See the following sections.

- Understanding the Attribute Calculations Dimension
- Understanding the Default Attribute Calculations Members
- Viewing an Attribute Calculation Example
- Accessing Attribute Calculations Members in Smart View
- Optimizing Calculation and Retrieval Performance
- Using Attributes in Calculation Formulas
- Understanding Attribute Calculation and Shared Members
- Differences Between Calculating Attribute Members and Non-Attribute (Stored and Dynamic Calc) Members

Understanding the Attribute Calculations Dimension

When you create the first attribute dimension in the outline, Essbase also creates the Attribute Calculations dimension comprising five members with the default names Sum, Count, Min (minimum), Max (maximum), and Avg (average). You can use these members in spreadsheets or in reports to dynamically calculate and report on attribute data, such as the average yearly sales of 12-ounce bottles of cola in the West.

The Attribute Calculations dimension is not visible in the outline. You can see it wherever you select dimension members, such as in Smart View.

The attribute calculation dimension has the following properties:

- System-defined

When you create the first attribute dimension in an application, Essbase creates the Attribute Calculations dimension and its members (Sum, Count, Min, Max, and Avg). Each member represents a type of calculation to be performed for attributes.

See Understanding the Default Attribute Calculations Members.

- Label only

  Like all label only dimensions, the Attribute Calculations dimension shares the value of its first child, Sum.

  See Member Storage Properties.

- Dynamic Calc

  The data in the Attribute Calculations dimension is calculated when a user requests it and is then discarded. You cannot store calculated attribute data in a database.

  See Dynamically Calculating Data Values.

- Not displayed in Outline Editor

  The Attribute Calculations dimension is not displayed in Outline Editor. Members from this dimension can be viewed in spreadsheets and in reports.

There is no consolidation along attribute dimensions. You cannot tag members from attribute dimensions with consolidation symbols (for example, + or -) or with member formulas in order to calculate attribute data. As Dynamic Calc members, attribute calculations do not affect the batch calculation in terms of time or calculation order.

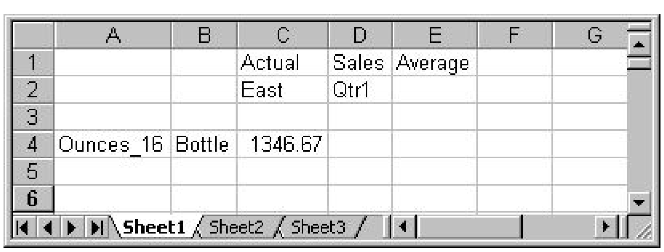
To calculate attribute data at retrieval time, Essbase performs the following tasks:

1. Finds the base-dimension members associated with the attribute-dimension members present in the current query
2. Dynamically calculates the sum, count, minimum, maximum, or average for the attribute-member combination for the current query
3. Displays the results in the spreadsheet or report
4. Discards the calculated values—that is, the values are not stored in the database

> **Note:**
>
> Essbase excludes #MISSING values when calculating attribute data.

For example, as shown in Figure 7-2, a spreadsheet user specifies two members of attribute dimensions (Ounces_16 and Bottle) and an Attribute Calculations member (Avg) in a spreadsheet report. Upon retrieval, Essbase dynamically calculates the average sales values of all products associated with these attributes for the current member combination (Actual -> Sales -> East -> Qtr1):

Figure 7-2 Retrieving an Attribute Calculations Member



	A	B	C	D	E	F	G
1			Actual	Sales	Average		
2			East	Qtr1			
3							
4	Ounces_16	Bottle	1346.67				
5							
6							

See Accessing Attribute Calculations Members in Smart View.

Understanding the Default Attribute Calculations Members

The Attribute Calculations dimension contains five members (Sum, Count, Min, Max, and Avg) that are used to calculate and report attribute data:

- Sum—Calculates a sum, or total, of the values for a member with an attribute or combination of attributes.
- Count—Calculates the number of members with the specified attribute or combination of attributes, for which a data value exists. Count includes only those members that have data blocks in existence. To calculate a count of all members with certain attributes, regardless of whether they have data values, use the @COUNT function in combination with the @ATTRIBUTE function.
- Avg—Calculates a mathematical mean, or average, of the nonmissing values for an specified attribute or combination of attributes (Sum divided by Count).
- Min—Calculates the minimum data value for a specified attribute or combination of attributes.
- Max—Calculates the maximum data value for a specified attribute or combination of attributes.

> **Note:**
>
> Each of these calculations excludes #MISSING values.

You can change these default member names, subject to the same naming conventions as standard members. See Changing the Member Names of the Attribute Calculations Dimension.

Viewing an Attribute Calculation Example

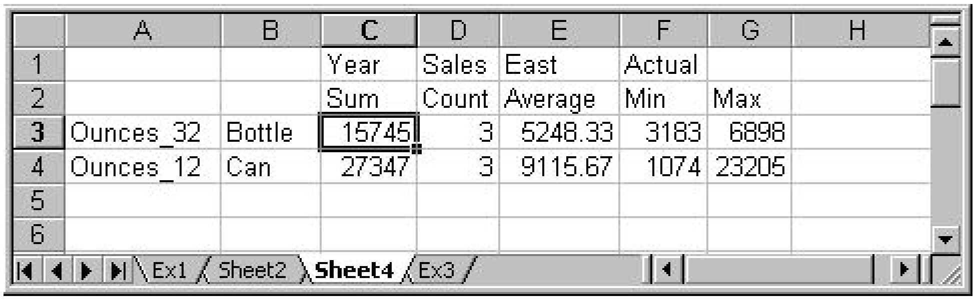
As an example of how Essbase calculates attribute data, consider the following yearly sales data for the East:

Table 7-7 Sample Attribute Data

Base-Dimension Member	Associated Attributes	Sales Value for Attribute-Member Combination
Cola	Ounces_12, Can	23205
Diet Cola	Ounces_12, Can	3068
Diet Cream	Ounces_12, Can	1074
Grape	Ounces_32, Bottle	6398
Orange	Ounces_32, Bottle	3183
Strawberry	Ounces_32, Bottle	5664

Figure 7-3 shows how calculated attribute data might look in a spreadsheet report. You can retrieve multiple Attribute Calculations members for attributes. For example, you can calculate Sum, Count, Avg, Min, and Max for bottles and cans.

Figure 7-3   Sample Spreadsheet with Attribute Data



	A	B	C	D	E	F	G	H
1			Year	Sales	East	Actual		
2			Sum	Count	Average	Min	Max	
3	Ounces_32	Bottle	15745	3	5248.33	3183	6898	
4	Ounces_12	Can	27347	3	9115.67	1074	23205	
5								
6								

Ex1 Sheet2 Sheet4 Ex3

Accessing Attribute Calculations Members in Smart View

You can access members from the Attribute Calculations dimension in Smart View. From the spreadsheet, users can view Attribute Calculations dimension members using any of the following methods:

- Entering members directly into a sheet
- Selecting members from the Query Designer
- Entering members as an EssCell parameter

See *Working with Oracle Smart View for Office*.

Optimizing Calculation and Retrieval Performance

To optimize attribute calculation and retrieval performance, consider the following:

- The calculation order for attribute calculations is the same as for dynamic calculations. For an outline, see Calculation Order for Dynamic Calculation.
- Because Essbase calculates attribute data dynamically at retrieval time, attribute calculations do not affect the performance of the overall (batch) database calculation.
- Tagging base-dimension members as Dynamic Calc may increase retrieval time.
- When a query includes the Sum member and an attribute-dimension member whose associated base member is tagged as two-pass, retrieval time may be slow.
- To maximize attribute retrieval performance, use any of the following techniques:
  - Configure the outline using the tips in Optimizing Outline Performance.
  - Drill down to the lowest level of base dimensions before retrieving data. For example, in Smart View, turn on the Navigate Without Data feature, drill down to the lowest level of the base dimensions included in the report, and then retrieve data.
  - When the members of a base dimension are associated with several attribute dimensions, consider grouping the members of the base dimension according to their attributes. For example, in the Sample.Basic database, you can group all 8-ounce products.

Using Attributes in Calculation Formulas

In addition to using the Attribute Calculations dimension to calculate attribute data, you can use calculation formulas on members of standard or base dimensions to perform specific calculations on members of attribute dimensions; for example, to determine profitability by ounce for products sized by the ounce.

You cannot associate formulas with members of attribute dimensions.

> **Note:**
>
> Some restrictions apply when using attributes in formulas associated with two-pass members. See the rows about two-pass calculations in Understanding Two-Pass Calculations on Attribute Dimensions.

You can use the following functions to perform specific calculations on attributes:

Table 7-8 Functions That Calculate On Attributes

Function	Type of Calculation
@ATTRIBUTE	Generate a list of all base members with a specific attribute. For example, generate a list of members that have the Bottle attribute, and then increase the price for those members.
@ATTRIBUTEVAL @ATTRIBUTEBVAL @ATTRIBUTESVAL	Return the value of the level 0 attribute member that is associated with the base member being calculated. • From a numeric or date attribute dimension (using @ATTRIBUTEVAL) • From a Boolean attribute dimension (using @ATTRIBUTEBVAL) • From a text attribute dimension (using @ATTRIBUTESVAL) For example, return the numeric value of a size attribute (for example, 12 for the member 12 under Ounces) for the base member being calculated (for example, Cola). For an additional example using @ATTRIBUTEVAL in a formula, see Calculating an Attribute Formula.
@TODATE	Convert a date string to numbers for a calculation. For example, use @TODATE in combination with the @ATTRIBUTEVAL function to increase overhead costs for stores opened after a certain date.
@WITHATTR	Generate a list of base dimension members associated with attributes that satisfy the conditions that you specify. For example, generate a list of products that are greater than or equal to 20 ounces, and then increase the price for those products.

Understanding Attribute Calculation and Shared Members

Attribute calculations start at level 0 and stop at the first stored member. Therefore, if your outline has placed a stored member between two shared members in an outline hierarchy, the calculation results may not include the higher shared member.

In the following example, when an attribute calculation is performed, the calculation starts with level 0 Member 2 and stops when it encounters the first stored member, which is Member A. Therefore, Member 1 would not be included in the calculation.

```
Member 1 (stored)
   Member A (stored)
     Member 2 (shared)
Member B (stored)
   Member 1 (shared member whose prototype member is Member 1 above)
```

To avoid unexpected results with attribute calculation, avoid mixing shared and stored members. For this example, if Member 2 were not shared, or Member 1 did not have a corresponding shared member elsewhere in the outline, calculation results would not be as expected.

Differences Between Calculating Attribute Members and Non-Attribute (Stored and Dynamic Calc) Members

The calculation of attribute dimension members is designed to work differently than the calculation of stored or Dynamic Calc members in standard dimensions.

The basis of this difference:

- **Members in standard dimensions:** The value of a parent member is based on aggregating the values of the parent member's child members, whether the child members are level 0 or upper-level members. All child member values contribute to the value of the parent member.
- **Members in attribute dimensions:** For each parent member for which an attribute aggregation is requested, the parent member's descendant list is expanded to include the dependent level 0 data blocks that need to be aggregated to calculate the value of the parent member.

Given these differences, the result of aggregating attribute dimension members might differ from the result of aggregating standard dimension members, if shared members are involved and there are multiple aggregation paths.

To workaround this issue, remove duplicate shared members under the aggregating attribute hierarchy or remodel the outline so that there are not multiple aggregation paths.

Non-Aggregating Attributes

Non-aggregating attributes are supported only in hybrid aggregation mode. If you set an attribute member as non-aggregating, then it will not be rolled up.

Non-aggregating attributes enable your analyses to benefit from the association of a cross-dimensional attribute. You can pivot the attribute across the base dimension to identify valid combinations, while avoiding the performance hit of dynamic aggregation.

Non-aggregating attributes only present values when the base dimension member is explicitly tagged as non-aggregating. If the base member is not tagged, then the top member in the attribute dimension also needs to be included in the query.

Mixing non-aggregating and standard attribute dimensions on the same grid is not supported. If mixed, all the attribute dimensions on the grid will behave as non-aggregating attributes.

Submitting Data for Valid Attribute Combinations in the Grid

Essbase allows submitting data on the grid for valid attribute combinations. A valid attribute combination is the result of an intersection of a dimension member for which an attribute is assigned. Invalid attribute combinations are represented on the grid with #invalid.

An invalid attribute combination is the result of an intersection of a dimension member for which an attribute is not assigned or, if an attribute is assigned to the member, the attribute combination is not within the scope of the grid query or the assigned attribute is incorrect.

This functionality applies to block storage and aggregate storage databases.

See Suppressing Invalid Attribute Combinations in the Grid.

Suppressing Invalid Attribute Combinations in the Grid

An invalid attribute combination is the result of an intersection of a dimension member for which:

- An attribute is not assigned to the member
- The assigned attribute is not within the scope of the grid query
- The attribute assigned to the member is incorrect

On the grid, invalid attribute combinations are represented by #invalid. You can configure Essbase to suppress #invalid values by setting the GRIDSUPPRESSINVALID configuration setting to TRUE. Invalid attribute combinations are suppressed when the row contains all #invalid values. Valid combinations with #MISSING values are not suppressed.

This functionality applies to block storage and aggregate storage databases.

Consider the following outline:

```
Product (Color, Size)
   sku1 (Color: Red; Size: Small)
   sku2 (Color: Blue; Size: Small)
   sku3 (Color: Blue; Size: Medium)
Supplier
Geo (Dynamic Calc)
Customer (Dynamic Calc)
YearTime (Dynamic Calc)
Measures Accouts
Size Attribute [Type: Text]
   Small
   Medium
...Large
Color Attribute [Type: Text]
   Red
   Blue
   Green
```

The Size attribute dimension values are Small, Medium, and Large. The Color attribute dimension values are Red, Blue, and Green. In the Product dimension, each member is assigned a Color and Size attribute:

- Member sku1 is assigned the color Red and the size Small attributes

- Member sku2 is assigned the color Blue and the size Small attributes
- Member sku3 is assigned the color Blue and the size Medium attributes

When drilling down on the Product, Color, and Size dimensions, all combinations—including invalid combinations—are displayed on the grid. As shown, the values in the Supplier column are either #MISSING (these are valid combinations) or #invalid (these are invalid combinations):

Table 7-9 Suppress Invalid Attribute Combinations: Grid 1

			Supplier
Small	Red	sku1	#MISSING
Small	Red	sku2	#invalid
Small	Red	sku3	#invalid
Small	Blue	sku1	#invalid
Small	Blue	sku2	#MISSING
Small	Blue	sku3	#invalid
Small	Green	sku1	#invalid
Small	Green	sku2	#invalid
Small	Green	sku3	#invalid
Medium	Red	sku1	#invalid
Medium	Red	sku2	#invalid
Medium	Red	sku3	#invalid
Medium	Blue	sku1	#invalid
Medium	Blue	sku2	#MISSING
Medium	Blue	sku3	#invalid

When Essbase is configured to suppress invalid attribute combinations, the grid shows only valid attribute combinations, which are the combinations with a value of #MISSING, as shown:

Table 7-10 Suppress Invalid Attribute Combinations: Grid 2

			Supplier
Small	Red	sku1	#MISSING
Small	Blue	sku2	#MISSING
Medium	Blue	sku2	#MISSING

Invalid combinations are not suppressed when there are values other than #invalid in a row. Consider this example of a grid, in which some rows have all #invalid values and some rows have #invalid and #MISSING values:

Table 7-11 Suppress Invalid Attribute Combinations: Grid 3

		Red	Blue	Green
		Supplier	Supplier	Supplier
Small	sku1	#MISSING	#invalid	#invalid

Table 7-11 (Cont.) Suppress Invalid Attribute Combinations: Grid 3

		Red	Blue	Green
Small	sku2	#invalid	#MISSING	#invalid
Small	sku3	#invalid	#invalid	#invalid
Medium	sku1	#invalid	#invalid	#invalid
Medium	sku2	#invalid	#invalid	#invalid
Medium	sku3	#invalid	#MISSING	#invalid
Large	sku1	#invalid	#invalid	#invalid
Large	sku2	#invalid	#invalid	#invalid
Large	sku3	#invalid	#invalid	#invalid

Essbase suppresses all of the rows that contain the #invalid value only, which leaves the rows that contain a mixture of #invalid and #MISSING values, as shown:

Table 7-12 Suppress Invalid Attribute Combinations: Grid 4

		Red	Blue	Green
		Supplier	Supplier	Supplier
Small	sku1	#MISSING	#invalid	#invalid
Small	sku2	#invalid	#MISSING	#invalid
Medium	sku3	#invalid	#MISSING	#invalid

Consider this outline:

```
Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
Measures Accounts (Label Only)
Product {A, B, Caffeinated, Intro Date, Ounces, Pkg Type}
   P1 (+) {A: aa1; B: True}
   P2 (+) {B: True}
   P3 (+) {A: aa3}
   100 (+) (Alias: Colas)
   200 (+) (Alias: Root Beer)
   300 (+) (Alias: Cream Soda)
   400 (+) (Alias: Fruit Soda)
   Diet (~) (Alias: Diet Drinks)
Market {Population}
Scenario (Label Only)
Caffeinated Attribute [Type: Boolean]
Ounces Attribute [Type: Numeric]
Pkg Type Attribute [Type: Text]
Population Attribute [Type: Numeric]
Intro Date Attribute [Type: Date]
A Attribute [Type: Text]
   aa1
   aa2
   aa3
B Attribute [Type: Boolean]
```

```
True
False
```

The A attribute dimension values are aa1, aa2, and aa3. The B attribute dimension values are True and False. In the Product dimension, the P1, P2, and P3 members are assigned one or two attributes:

- Member P1 is assigned the A aa1 attribute and the B True attribute
- Member P2 is assigned the B True attribute
- Member P3 is asssigned the A aa3 attribute

Consider this example of a grid, in which there is only one valid attribute combination—member P1, which is assigned the A aa1 attribute. The attributes assigned to members P2 (True) and P3 (aa3) are not represented in the grid; therefore, those attribute combinations are invalid.

Table 7-13 Suppress Invalid Attribute Combinations: Grid 5

		Market	Scenario
		Year	
		Measures	
aa1	P1	#MISSING	
aa1	P2	#invalid	
aa1	P3	#invalid	
aa2	P1	#invalid	
aa2	P2	#invalid	
aa2	P3	#invalid	

When Essbase is configured to suppress invalid attribute combinations, the grid shows only the valid attribute combination for member P1:

Table 7-14 Suppress Invalid Attribute Combinations: Grid 6

		Market	Scenario
		Year	
		Measures	
aa1	P1	#MISSING	

8 Working with Typed Measures

You can use typed measures to extend the analytical capabilities of Essbase. In addition to numeric values, measures can also be associated with text- or date-typed values.

About Typed Measures

Typed measures extend the analytical capabilities of Essbase. In addition to numeric values, measures can also be associated with text- or date-typed values.

Text measures are tagged as “text” in whichever dimension measures are represented. They enable cell values to contain one of an enumerated list of text labels. These labels are defined, at the outline level, using a mapping artifact called a text list.

Date measures are tagged as “date” in the dimension where measures are represented. Date measures enable cell values in the form of a formatted date.

The following general guidelines apply to both text and date measures:

- Add them to the existing measures dimension; for example, Accounts.
- Do not consolidate them. By default, text and date measures are assigned the never consolidate operator (^).
- If the measure is not designed to be consolidated, queries should be made at the same level at which data was loaded.
- After you enable an outline to support typed measures, it cannot be reverted back to an outline that does not support typed measures.
- If you need to enable your outline to support typed measures,
  1. In the Essbase web interface, open the outline for editing.
  2. Click **Outline Properties**.
  3. On the General tab, change the value for **Typed Measures Enabled** from False to True.
  4. Click **Apply and Close**.
  5. Click **Save**, and close the outline editor tab.
- Text and date measures functionality applies to both aggregate storage and block storage applications.

Working with Text Measures

Text measures extend the analytical capabilities of Essbase beyond numerical data to text-based content. Storage and analysis of textual content can be useful when a cell needs to have one of a finite list of textual values; for example, a product may be sold in five different colors. The color is a text measure whose value must be one of those five colors. Any color not represented in the finite list would be considered by Essbase to be out of range.

You create text measures at the database level. Text measures are made possible by your mapping of a set of text strings to corresponding numeric IDs. These mappings are contained in database-level text list objects that you create.

- Text Measures Workflow
- Text List Objects and Text List Members

Text Measures Workflow

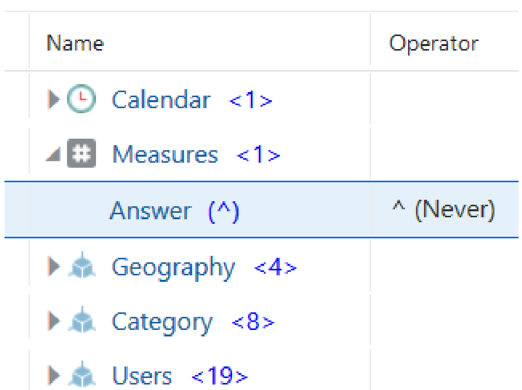
Use one of the following workflows to implement text measures for a cube. Text measures are enabled by default for an outline, but to use them, you must follow the implementation steps in either the Essbase web interface or the application workbook you use to build a cube.

Text Measures Workflow in Essbase web interface

The following workflow uses the Essbase web interface.

1. Create a measure in the Accounts dimension that will be the textual measure. In the member properties, set its consolidation operator to ^ (no consolidation).

   For example, the following textual measure, **Answer**, is used in the Facility Rating cube, which you can download from the Files catalog under All Files > Gallery > Applications > Facility Rating.

2. From the outline properties, go to edit mode and add a text list object to store the available text values for the textual measure. Map each text value to a numeric value, so that Essbase can work with the text values. Also map Missing and Out of Range to numeric values.

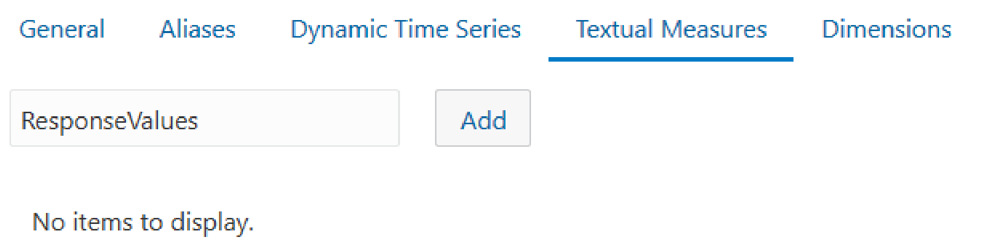
   a. Open the cube outline in the Essbase web interface.

   b. Click **Edit**, and then click **Outline Properties**.

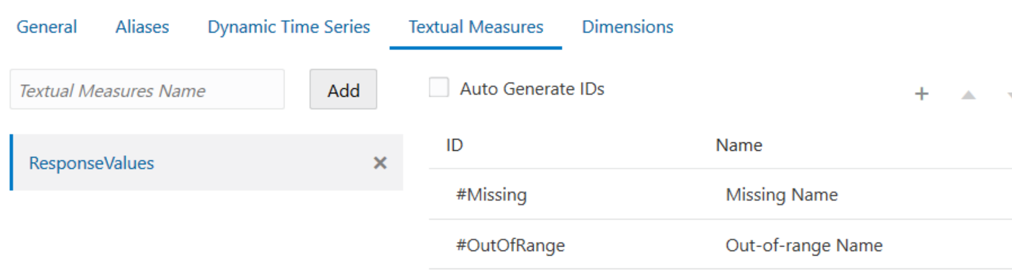
   c. Click the **Textual Measures** tab.

d. Enter a name for a text list object and click Add.

For example, a text list object named **ResponseValues** is used in the Facility Rating cube.

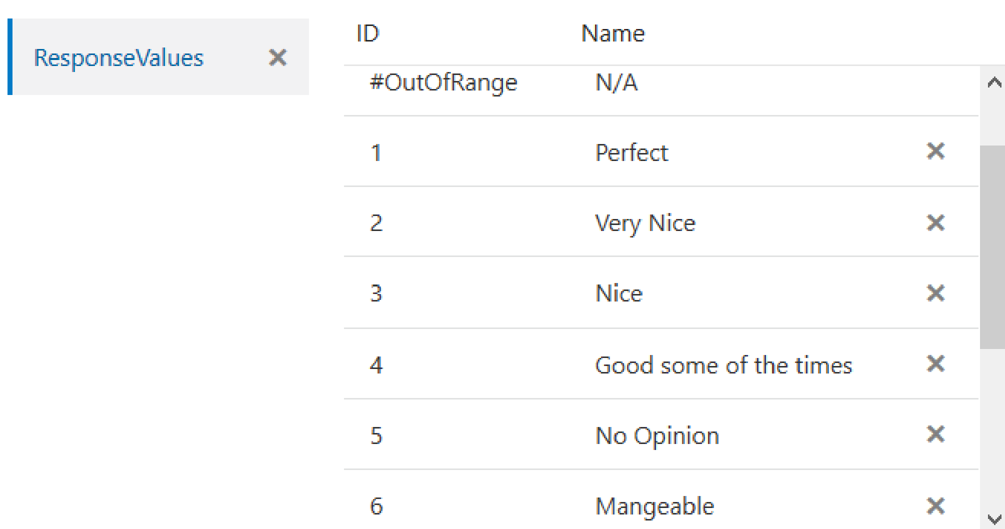


Essbase automatically adds IDs for handling missing and out of range values.

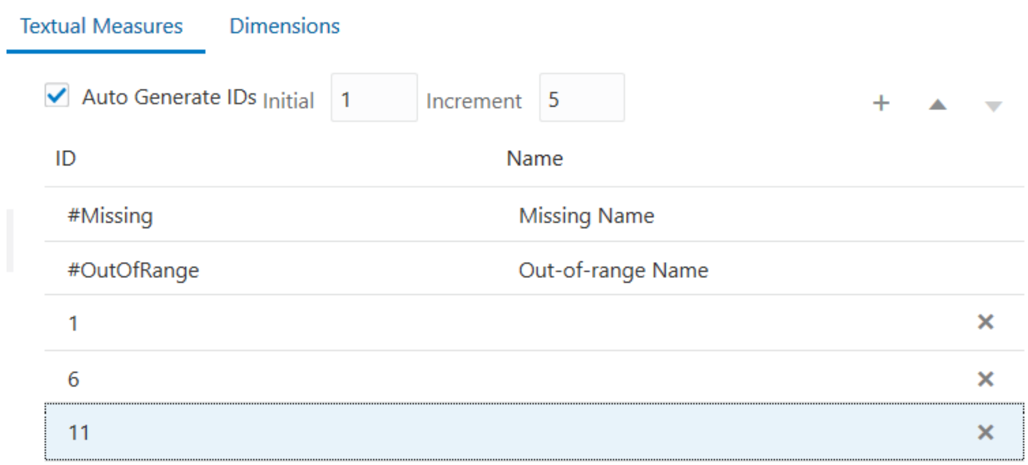


e. Edit the missing and out of range ID and mapping name as needed. Add additional numeric values and map them to text values. Each numeric value can have only one text value mapped to it.

For example, the text list object named **ResponseValue** contains numeric values mapped to text values; these are used for survey response answers in the Facility Rating cube.

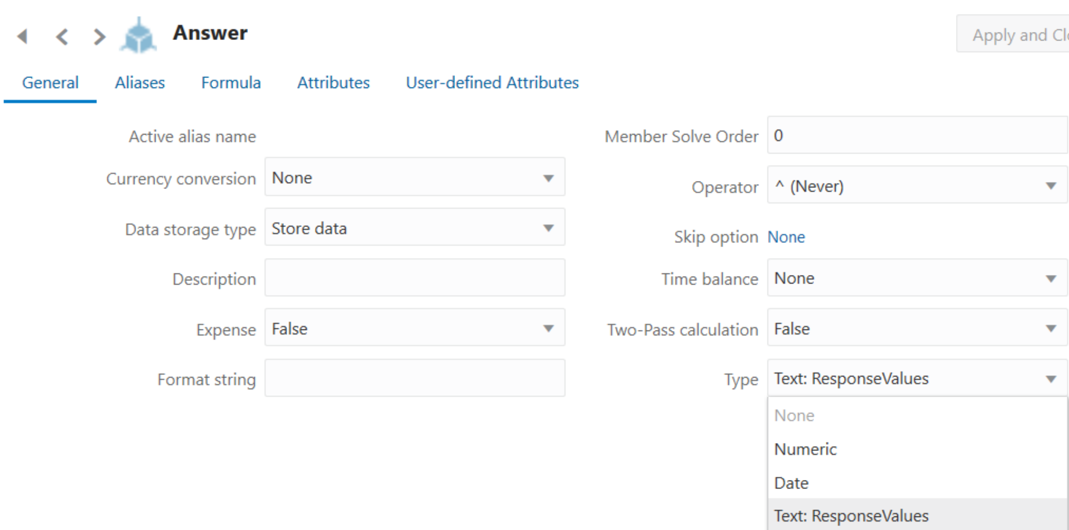


Optionally, you can auto generate the IDs, selecting an increment, as in the example below. There is no need for the IDs to be in any particular order. Don't forget to assign names to each ID.



f. Click **Apply and Close** to save the text list object with the outline.

3. Still in outline editing mode, return to the textual measure in the Accounts dimension (in our example, the member named **Answer**) and click **Inspect**. In the member properties, define the measure as type Text, associated with the name of the text list object you created.



4. Click **Apply and Close** to save the type setting for the textual measure.

5. Save the outline.

Text Measures Workflow using Application Workbooks

As an alternative to using the Essbase web interface, you can also build a cube with a textual measure implementation from an application workbook.

To do this, include a Cube.TypedMeasures worksheet in the workbook you use to build the cube. In the Cube.TypedMeasures worksheet, fill out what you want to use for the text list name (in this example, ResponseValues), provide the associated textual measure member name (in this case, Answer), and define the text list object (the section that maps IDs to text values).

Text List Properties

List Name	ResponseValues
Associated Members	Answer

ID	**Text**
#Missing	Blank
#OutOfRange	N/A
1	Perfect
2	Very Nice
3	Nice
4	Good some of the times
5	No Opinion
6	Mangeable
7	Needs Improvement
8	Bad
9	Very Bad
10	Not Usable

Settings | Cube.Generations | **Cube.TypedMeasures**

As an easy way to implement text measures from your application workbook,

1. Download the Facility Rating gallery template workbook, available from the Files catalog under All Files > Gallery > Applications > Facility Rating.
2. Copy the Cube.TypedMeasures worksheet from the template into your own application workbook.
3. Make modifications as needed to the values for **List Name**, **Associated Members**, IDs, and their associated text values.
4. Build the cube from the workbook, either by importing it or using Cube Designer.

See also: Performing Database Operations on Text and Date Measures.

Text List Objects and Text List Members

A cell that corresponds to a text measure can have one of a finite list of up to 1024 valid text values. Internally, Essbase needs to store the text values as numbers. Therefore, a mapping of text values to numeric values is required. You define the mapping between the text and numeric values by creating a text list object. A text list object consists of a list of text values and a numeric value that corresponds to each text value.

For example, you can create a text list object called “Customer Satisfaction Level” with the contents shown below. The **Name** column contains the possible text values for a text measure, and the **ID** column represents the internal numeric value used by Essbase.

Table 8-1 Example of a Text List Object

Name	ID
Missing	#MISSING
N/A	#OUTOFRANGE
High	1
Medium	2
Low	3

Each text value must map to a unique numeric value. Any text value that does not map to an integer in the text list object is considered by Essbase to be invalid.

The first two IDs, #MISSING and #OUTOFRANGE, are for handling cases where the textual data is invalid or empty. For example, if a user attempted to load an unmapped value such as “Average” to a text measure, the cell value would not be updated, and would display as #MISSING in a subsequent query. If a user loads a numerical cell value which is unmapped, the subsequent query would return N/A.

Aside from #MISSING and #OUTOFRANGE , all of the other IDs must be integers.

The internal numeric value of the "#OutOfRange" ID is the constant `-0.000000000000011`.

Working with Date Measures

Date measures enable members to be associated with date-type values. The ability to process dates in the measures dimension can be useful for types of analysis that are difficult to represent using the Time dimension.

For example, an application that analyzes asset depreciation may need to track acquisition dates for a series of capital assets. The company has been in business for fifty years, so the acquisition dates span too large a period to allow for feasible Time dimension modeling (the Time dimension only covers five years).

In addition to their ability to represent values spanning large time periods, date measures can also capture date values with smaller granularity than is captured in the Time dimension, such as hours and minutes.

- Implementing Date Measures
- Functions Supporting Date Measures

Implementing Date Measures

Date measures are supported for Essbase aggregate and block storage cubes. To implement date measures, first enable them for the cube. Then, select a date format, and add the date measures to the outline in the accounts dimension.

Use the following workflow to enable and use date measures:

1. In the outline properties, enable typed measures.
2. In the outline properties, select a date format (for example, `yyyy-mm-dd`). All date measures in the outline must use the same format.
3. Create a date measure in the outline (in the Accounts dimension), and in the member properties, define it as type Date.

For example, in ASOsamp.Basic, you can enable typed measures for the outline, select a date format, and add a measure named IntroDate defined as type Date.

The date values are stored internally as numeric values, although you load them into Essbase as formatted date strings. When queried, date measures are displayed according to the selected date format.

See Performing Database Operations on Text and Date Measures.

The following configuration setting is useful for tuning data exports of date measures: MAXDATE

Functions Supporting Date Measures

Some MDX and calculation functions help you perform Essbase calculations based on calendar dates or date-type strings.

The following MDX functions are useful for calculations based on date measures.

- DateDiff
- DatePart
- DateRoll
- FormatDate
- GetFirstDate
- GetLastDate
- Todate
- TodateEx
- Today

The following calculation functions are useful for calculations based on date measures.

- @DATEDIFF
- @DATEPART
- @DATEROLL
- @FORMATDATE
- @TODATEEX
- @TODAY

The DATEFORMAT Report Writer command.

Performing Database Operations on Text and Date Measures

You can perform these common database operations when using text and date measures:

- Loading, Clearing, and Exporting Text and Date Measures
- Consolidating Text and Date Measures
- Retrieving Data With Text and Date Measures
- Limitations of Text and Date Measures

Loading, Clearing, and Exporting Text and Date Measures

To load data to text or date measures, follow the same procedure as for loading data to members with numeric measures. The input data should contain formatted date values, or text values corresponding to the text list object that is associated with the text measure.

If you attempt to load text values that are not present in the text list object associated with that member, Essbase issues a warning message.

In aggregate storage cubes, values can only be loaded at the input level; this restriction applies equally to text and date measures. In block storage cubes, text and date values can be loaded at any level.

You can clear, lock and send, and export text or date values just as you perform those operations on numeric values.

Format of Textual Measures in Data Exports and Free-Form Data Imports

You can load text or date values with or without rule files. When a rule file is not used (in a free-form data load), in some cases, you must distinguish text or date values from member names by prefixing them with the string `#Txt:` and enclosing the prefix and the text values in double quotation marks, for example `"#Txt:Highly Satisfied"`.

> **Note:**
>
> The entire string, including the prefix must be enclosed in the quotation marks. For example, `#Txt:"Highly Satisfied"` is incorrect and returns an error.

Here is an example of a line of data in a free-form data load file. This is also how an data export file is formatted, when you export data from an aggregate storage cube with textual measures:

```
"100-10" "New York" "Cust Index" "#Txt:Highly Satisfied"
```

The text value `Highly Satisfied` is the text list object name, prefixed with the marker `#Txt:` to differentiate it from member names such as "New York".

The `#Txt:` prefix is also applicable when loading date measures, #Missing values, and values that would be #OutOfRange.

Exported textual measures are marked in the export file with `#Txt:` as a prefix before the text list object name. For example,

```
"100-10" "New York" "Cust Index" "#Txt:Highly Satisfied"
```

When re-importing textual measures into the cube using a free-form data load with no rule file, the `#Txt:` markers are required to distinguish the text list object names from the rest of the data.

Additional Aggregate Storage Guidelines for Loading Text and Date Values

Use the following guidelines when loading text and date values into an aggregate storage cube. These guidelines will help eliminate invalid aggregations.

- Load values at the input level.
- Use Replace mode.

> **Note:**
>
> Replace mode is set when committing the buffer. In MaxL, use the **override values** grammar of the **import data** statement. In the Essbase Java API, use the *commitType* parameter of the IEssCube.loadBufferTerm method. In the C API, use the *ulCommitType* field of the EssLoadBufferTerm function.

- Use a single load buffer to load all values associated with date/text measures.
- Use the aggregate_use_last aggregation method.

> **Caution:**
>
> The aggregate_use_last method has significant performance impact, and is not intended for large data loads. If your data load is larger than one million cells, consider separating the numeric data into a separate data load process (from any typed measure data). The separate data load can use aggregate_sum instead.

  Aggregate_use_last is set when creating the load buffer. In MaxL, see the PROPS terminal that is part of the **initialize load_buffer** grammar in the **alter database** statement. In the Essbase Java API, use the *duplicateAggregationMethod* parameter of the IEssCube.loadBufferInit method. In the C API, use the *ulDuplicateAggregationMethod* field of the EssLoadBufferInit function.

- Avoid loading #MISSING values to text/date measures in incremental data load mode. When a #MISSING value is loaded to a cell with a non-Missing value in incremental load, it is replaced with a zero value. The zero value may not have the same meaning as the #MISSING value for date/text measures. Use full data load if you need to load #MISSING values to date/text measures.

If mixed (numeric and text or date) data are being loaded, either ensure that Replace mode is sufficient for your numeric data, or create a separate data load process for the numeric data.

Consolidating Text and Date Measures

By default, text measures are assigned the ^ operator (never consolidate). Text and date measures are not consolidated to higher level members along non Accounts dimensions.

If you tag a text or date measure with an operator other than ^, it will be consolidated along other dimensions based on its internal numeric value. This is not recommended for aggregate storage cubes, because only the + operator is supported for consolidation, and the aggregated values likely will not have any validity for text or date measures. Additionally, Essbase does not translate out-of-range values to #OUTOFRANGE during consolidation.

For block storage cubes, you can write calculation scripts that consolidate text measures in a custom fashion. You might want to consolidate text measures when they represent ranking measures. For example, consider a text list named "CustomerSatisfaction", which contains mappings such as Excellent=5, Good=4, Fair=3, Poor=2, Bad=1. The values are loaded at level 0. You can consolidate values to parent levels by taking an average of values at child levels. For example, the value of “CustomerSatisfaction” at [Qtr1] is the average of values at [Jan], [Feb], [Mar].

Retrieving Data With Text and Date Measures

Text or date measures can be queried in the same way as numerical measures, using Smart View, MDX, Report Writer, or the Analyze view in the Essbase web interface. The corresponding cells are displayed with the appropriate text values or formatted date values.

For example, the following Smart View grid is from the Facility Rating cube, which you can download from the Files catalog under All Files > Gallery > Applications > Facility Rating. The **Answer** member is a textual measure within the Measures dimension.

		Answer	**Answer**
		USA	**USA**
		FY2018-Jan	**FY2018-Feb**
User1	**Cafeteria**	No Opinion	Nice
User1	**Commute**	Very Bad	No Opinion
User1	**Conference Rooms**	Nice	Mangeable
User1	**IT Equipment**	Perfect	Mangeable
User1	**Kitchen**	Very Nice	Good some of the times
User1	**Office**	Mangeable	Very Nice
User1	**Parking**	Needs Improvement	No Opinion
User1	**Temperature Control**	Mangeable	No Opinion
User1	**Category**	Blank	Blank
User2	**Cafeteria**	Good some of the times	Perfect
User2	**Commute**	Good some of the times	Bad
User2	**Conference Rooms**	Very Nice	Very Nice

The MDX function EnumValue and the calculation function @ENUMVALUE are designed for getting the numeric value of text measures. These functions can be useful in MDX scripts, calc scripts, or formulas when you need to do operations based on the numeric value of a member rather than its text value.

The following Report Writer commands are designed to work with numeric data, and are not supported for text or date measures:

- RESTRICT
- TOP
- BOTTOM
- SORT* commands
- CALCULATE COLUMN
- CALCULATE ROW

Limitations of Text and Date Measures

An outline restructure does not restructure text lists. If the mapping of numeric to text values in a text list is changed, the change will be reflected in the text data already present in the database for that text list. Therefore, when adding items to a text list, add them to the top or bottom of the list so as to avoid altering the mapping numbers of existing text list items.

Text and date measures are not supported across partitions.

Shared members inherit the text or date type of the prototype member.

Working with Format Strings

Using format strings, you can format the values (cell contents) of Essbase database members in numeric type measures so that they appear, for query purposes, as text, dates, or other types of predefined values. The resultant display value is the cell's formatted value (FORMATTED_VALUE property in MDX).

The underlying real value is numeric, and this value is unaffected by the associated formatted value. Format strings enable you to display more meaningful values in place of raw numeric values. For example, using a text based formatted value, you might display data cells as “High,” “Medium,” and “Low.”

Text and date type values are additionally supported using the built-in text and date measure types. Format strings add more flexibility to your implementation, in that you can apply format strings to members in multiple dimensions, whereas with text and date measures, you can only apply one or the other to a single measures dimension. You can apply format strings to numeric dimensions; you do not have to type the dimension as text or date.

Format strings can be applied to either aggregate storage or block storage databases.

Format strings can be defined on the following members:

- All members in Measures dimension
- Members associated with explicit formula strings on other dimensions

Topics:

- Implementing Format Strings
- MDX Format Directive
- Functions Supporting Format Strings
- Limitations of Format Strings

Implementing Format Strings

Format strings are supported for aggregate and block storage databases. You implement format strings at the database level.

Use the following workflow to enable and use format strings:

1. In the outline properties, enable typed measures.
2. In the Accounts dimension, create a measure whose members you want to format, and in its member properties, edit the Associate Format String field to create an MDX format directive. For the syntax to create an MDX format directive, see MDX Format Directive.

MDX Format Directive

A format string is defined by the following syntax:

```
format_string_expression = MdxFormat (string_value_expression)
```

string_value_expression is a valid MDX string value expression as described in the MDX specification (see MDX Grammar Rules).

Most MDX expressions can be used to specify format strings; however, format strings cannot contain references to values of data cells other than the current cell value being formatted. The current cell value can be referenced using the MDX CellValue function.

Essbase treats members with invalid format strings as if there is no format string defined. Outlines can be saved with invalid format strings. Essbase generates a warning if a query consists of a member with an invalid format string.

If a member is not associated with a format string, default format rules are applied. The default format rules format a cell based on whether the measure is numeric, text, or date type. For numeric measures, the default formatted value is the text version of that number. For text measures, the default formatted value is the text value based on the associated text list object. For date values, the default format is a date string formatted according the date format string defined in the outline properties.

Functions Supporting Format Strings

The following MDX functions can be useful when applying format strings to a measure. Format strings are applied as MDX expressions, in both aggregate and block storage databases.

- EnumText returns the text list label associated with the internal numeric value.
- EnumValue returns the internal numeric value for a text list label.
- CellValue returns the internal numeric value of the current cell.
- NumToStr converts a value to a decimal string.

The @ENUMVALUE calculation function can be useful when writing calculation scripts for a block storage database that has text measures or format strings. This function returns the text list label associated with the internal numeric value.

The MaxL alter session **set dml_output** statement has a clause **set formatted_value on | off**. By default, formatted values are displayed in queries, but this statement can be used to turn off the display of formatted values.

Limitations of Format Strings

Format strings are not supported across partitions.

Shared members cannot have separate format strings; they inherit the format string of the prototype member.

Maximum length of a format string: 256 characters.

9

Designing and Building Currency Conversion Applications

This chapter explains how to use the Essbase currency conversion feature. These topics apply only block storage (and hybrid) cubes, and are not applicable to aggregate storage cubes.

About Currency Conversion

Building a currency conversion application is one way you can translate financial data from one currency into another.

Currency conversion facilitates comparisons among countries and enables consolidation of financial data from locations that use different currencies.

For example, consider an organization that analyzes profitability data from the UK, reported in pounds, and from Japan, reported in yen. Comparing local currency profitability figures would be meaningless. To understand the relative contribution of each country, you must convert pounds into yen, yen into pounds, or both into another currency.

As another example, reporting total profitability for North America requires standardization of the local currency values that constitute the North America total. Assuming that the U.S., Mexico, and Canada consolidate into Total North America, the profitability total is meaningless if data is kept in local currencies. The Total North America sum is meaningful only if local currencies are converted to a common currency before consolidation.

Topics in this section refer to a sample currency application named Sample_Currency with two cubes, Interntl and Xchgrate. You can recreate these samples from the gallery in the file catalog, from a compressed Lifecycle Management (LCM) file. See Building Currency Conversion Applications and Performing Conversions for instructions.

About the Sample Currency Application

The Sample_Currency application is similar to the Sample Basic application, but expands the business model to global markets.

The Sample_Currency application builds on the business scenario introduced in Case Study: Designing a Single-Server, Multidimensional Database as the Beverage Company (TBC) expands its business outside the U.S. TBC adds the following markets:

- Three locations in Canada: Toronto, Vancouver, and Montreal
- Four locations in Europe: the UK, Germany, Switzerland, and Sweden

In addition, TBC adds a new member, U.S., a consolidation of data from the U.S. regions: East, West, South, and Central.

Data for each TBC market location is captured in local currency. U.S. dollar values are derived by applying exchange rates to local values.

TBC must analyze actual data in two ways:

- Actuals are converted at actual exchange rates.
- Actuals are converted at budget exchange rates to analyze variances due to exchange rates.

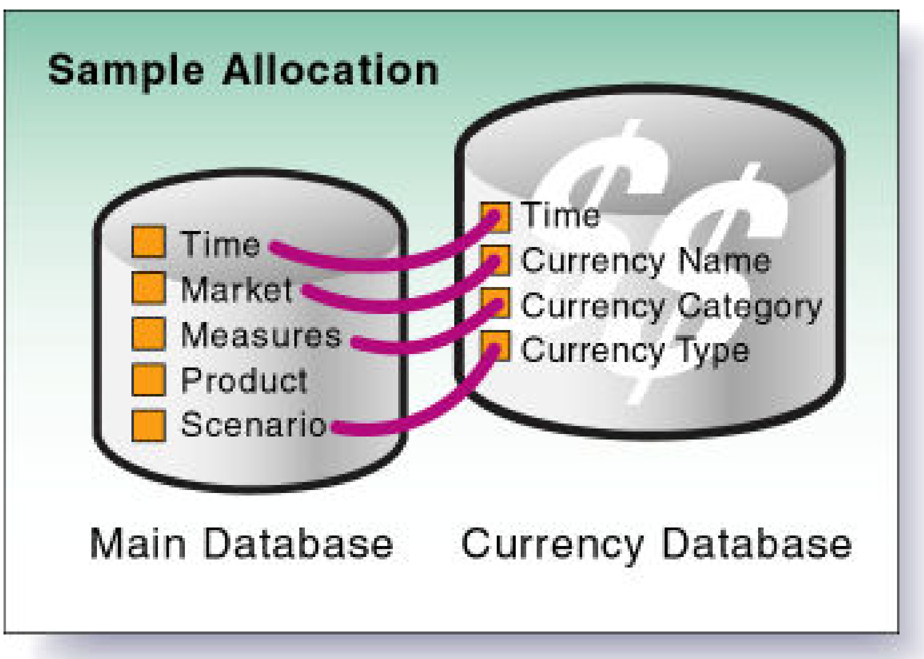
After all actuals are processed, budget data is converted with budget exchange rates.

The TBC currency application consists of the main database (Interntl) and the currency database (Xchgrate).

Structure of Currency Applications

In currency conversion application, the main database (cube) is divided into at least two slices. One slice handles input of the local data, and another slice holds a copy of the input data converted to a common currency.

Essbase holds the exchange rates required for currency conversion in a separate currency cube. The currency cube outline, automatically generated by Essbase from the main cube after you assign the necessary tags, typically maps a given conversion ratio onto a section of the main cube. After the currency cube is generated, it can be edited like any other Essbase cube.



Main Cube

To enable Essbase to generate the currency cube outline automatically, you modify dimensions and members in the main outline. In the Sample currency application, the main cube is Interntl.

The main cube outline can contain from 3 to *n* dimensions. At minimum, the main cube must contain the following dimensions:

- A dimension tagged as time. Tagging a dimension as time generates a dimension in the currency cube that is identical to the time dimension in the main cube. In the Sample_Currency.Interntl cube, the time dimension is Year.
- A dimension tagged as accounts. Tagging a dimension as accounts and assigning currency categories to its members creates a dimension in the currency cube that contains members for each of the individual currency categories. Category assignment enables the application of different exchange rates to various accounts or measures. In the Sample_Currency.Interntl cube, the accounts dimension is Measures.

  Each descendant of a member inherits the currency category tag of its ancestor. A member or sub-branch of members also can have its own category.

  For example, profit and loss (P&L) accounts may use exchange rates that differ from the rates used with balance sheet accounts. In addition, some accounts may not require conversion. For example, in the Sample_Currency.Interntl cube, members such as Margin% and Profit% require no conversion. You tag members not to be converted as No Conversion. The No Conversion tag is not inherited.
- A market-related dimension tagged as country. Tagging a dimension as country and assigning currency names to individual countries creates a member in the currency cube for each currency. In the Sample_Currency.Interntl cube, the Market dimension is tagged as country. The currency name for this dimension is USD (U.S. dollars), because all local currencies must be converted to USD, the company's common currency.

  Because multiple members can have the same currency name, the number of currency names is typically less than the total number of members in the dimension. As shown below, the Sample_Currency.Interntl cube uses only six currency names for the 15 members in the Market dimension. Each of the children of the member Europe uses a different currency and, therefore, must be assigned an individual currency name. However, the U.S. dimension and its four regional members all use the same currency. The same is true of the Canada member and its three city members. When the children of a given member share a currency, you must define a currency name for only the parent member.

Dimensions and Members	Currency Name
Market - Country	USD (US Dollar)
U.S.	
East	
West	
South	
Central	
Canada	CND (Canadian dollar)
Toronto	
Vancouver	
Montreal	
Europe	GBP (British pound)
UK	EUR (Euro)
Germany	CHF (Swiss franc)
Switzerland	SEK (Swedish krona)
Sweden	

When preparing a main cube outline for currency conversion, you can create an optional currency partition to tell Essbase which slice of the cube holds local currency data and which holds data to be converted. The dimension that you tag as currency partition contains members for both local currency values and converted values. Local currency data is converted to common currency data using currency conversion calculation scripts. In the Sample.Interntl cube, the Scenario dimension is the currency partition dimension.

Currency Cube

By assigning currency tags to members in the main cube outline, you enable Essbase to generate the currency cube automatically. In the Sample_Currency application, the currency cube is Xchgrate.

In the currency cube, all level 0 members must be stored, non-dynamic-calc members. This means that the cube the currency cube is generated from must also have all level 0 members as stored, non-dynamic-calc members.

A currency cube always consists of the following three dimensions, with an optional fourth dimension:

- A dimension tagged as time, which is typically the same as the dimension tagged as time in the main cube. This allows the currency cube to track currency fluctuations over time and to accurately convert various time slices of the main cube. In the Sample.Xchgrate cube, the dimension tagged as time is Year.

  Each member of the time dimension in the main cube must be defined in the currency cube. Values by time period in the main cube usually are converted to the exchange rates of their respective time period from the currency cube (although you can convert data values against the exchange rate of any period).

- A dimension tagged as country, which contains the names of currencies relevant to the markets (or countries) defined in the main cube. Each currency name defined in the main cube must also exist in the currency cube. The currency names define the country-to-exchange rate mapping when conversion occurs.

  In the Sample_Currency.Xchgrate cube, the country dimension is CurName. The following table lists the currency names in the CurName dimension:

Dimensions and Members	Alias Name
CurName - Country	U.S. dollar
USD	Canadian dollar
CND	British pound
GBP	Euro
EUR	Swiss franc
CHF	Swedish krona
SEK	

- A dimension tagged as accounts, which enables the application of various rates to members of the dimension tagged as accounts in the main cube. The categories defined for the accounts dimension in the main cube are used to form the members in the accounts dimension of the currency cube. For example, it may be necessary to convert Gross Profit and Net Profit using one category of rates, while other accounts use a different set of rates.

  In the Sample_Currency.Xchgrate cube, the dimension tagged as accounts is CurCategory, and the account categories included are P&L (Profit & Loss) and B/S (Balance Sheet).

- A currency cube, which typically includes an optional currency type dimension, which enables different scenarios for currency conversion. Typically, an application has different exchange rates for different scenarios, such as actual, budget, and forecast. To convert data between scenarios, select which type of rate to use.

The currency type dimension is created when you generate the currency outline and is not directly mapped to the main cube. Therefore, member names in this dimension need not match member names of the main cube.

In the Sample_Currency.Xchgrate cube, the currency type dimension is CurType. CurType includes actual and budget scenarios.

Conversion Methods

When designing a currency conversion application, consider whether your business requires analysis and reporting on local currencies, or only on converted currencies.

Essbase supports two conversion methods:

- **Overwrite local values with converted values**

  Some applications require that only converted values be stored in the main cube. Local values are entered, and the conversion operation overwrites local values with common currency values. This method assumes that there is no requirement for reporting or analyzing local currencies.

  Because this operation overwrites data, load local values and recalculate the data each time you perform a conversion. This method is useful only for single (not ongoing) conversions.

- **Keep local and converted values**

  Most applications require that data be stored in both local and common currency (converted) values. This method permits reporting and analyzing local data, and data modifications and recalculations are easier to control. To use this method, you must define a currency partition dimension in the main cube.

Either of these methods may require a currency conversion at report time. Report time conversion enables analysis of various exchange rate scenarios without actually storing data in the database. The Currency Conversion module enables performance of ad hoc conversions in report scripts.

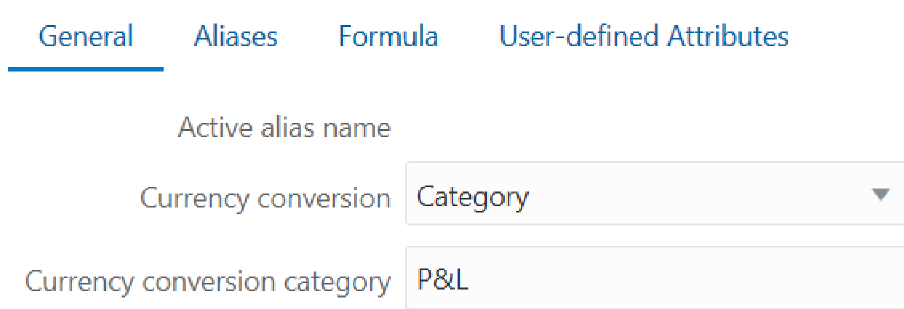
Building Currency Conversion Applications and Performing Conversions

This topic describes how to build a currency conversion application and convert currencies. The instructions use a main cube, Sample.Interntl, and a currency cube, Sample.Xchgrate as examples. You can recreate these to follow along with the instructions.

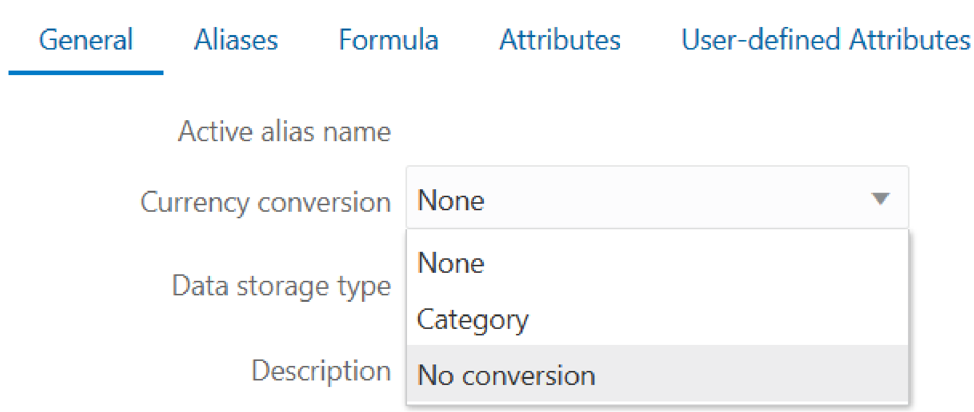
To build a currency conversion application and perform conversions,

1. Build the sample cubes used in this documentation.
   a. In the Essbase web interface, click **Files** and navigate to `gallery/Technical/Currency Conversion`.
   b. Copy `LCM_Sample_Currency.zip` to your own directory under `users`.
   c. Click **Jobs**.
   d. From the **New Job** menu, select **Import LCM**.
   e. Select the `LCM_Sample_Currency` zip file from your user directory.
   f. Click OK to build the Sample_Currency.Interntl cube using Lifecycle Management (LCM).

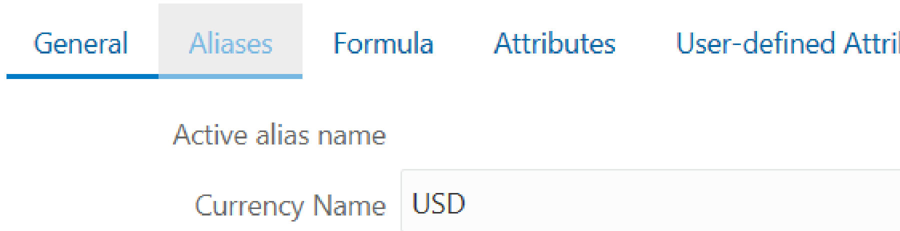
2. Open the main cube's outline and prepare it for currency conversion. The Interntl cube is already prepared, if you imported it, but this topic uses it as an example anyway. If you are working with another cube, open the outline for editing, if you need to make any of the changes indicated in this section.
   a. Click **Applications** and refresh the list.
   b. Expand the application (in this example, Sample_Currency).
   c. To the right of the cube name (in this example, Interntl), select **Outline** from the actions menu.
   d. In the outline viewer, confirm that the outline contains:
      - a dimension tagged as time
      - a dimension tagged as accounts
      - a dimension tagged as country
   e. Assign a currency category to the accounts dimension (and to members within this dimension) to handle different categories of exchange rates. Child members inherit the dimension category unless you specify them differently.



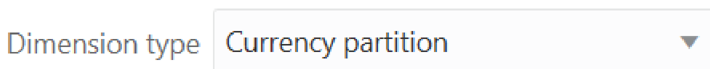
   f. Tag any members that should not be converted as **No Conversion**. The No Conversion tag is not inherited. **None** does not indicate no conversion; rather, it indicates that the conversion category is unspecified (thus inherited).



   g. Assign currency names to members in the country dimension. For the top-level member, assign the currency name of the base currency (for example, USD) to ensure that the base currency is inherited down the country dimension hierarchy. When the children of a member share a single currency, you only need to define a currency name for the parent.

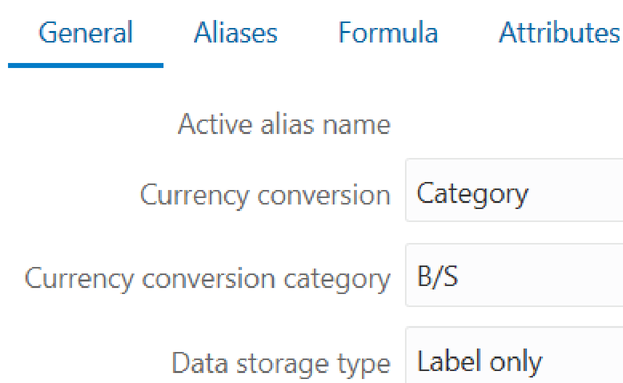


h. Optional: Tag one dimension as a currency partition, to tell Essbase which part of the cube holds local data and which holds base data.



The dimension you tag as the currency partition contains members for both local and base values. This dimension holds the data that users input in their own currencies. The local data is converted to the base data using currency conversion calculation scripts.

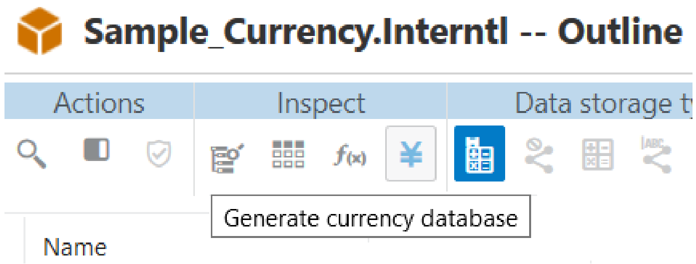
i. Apply the Label Only tag to dimensions and members that do not store data.



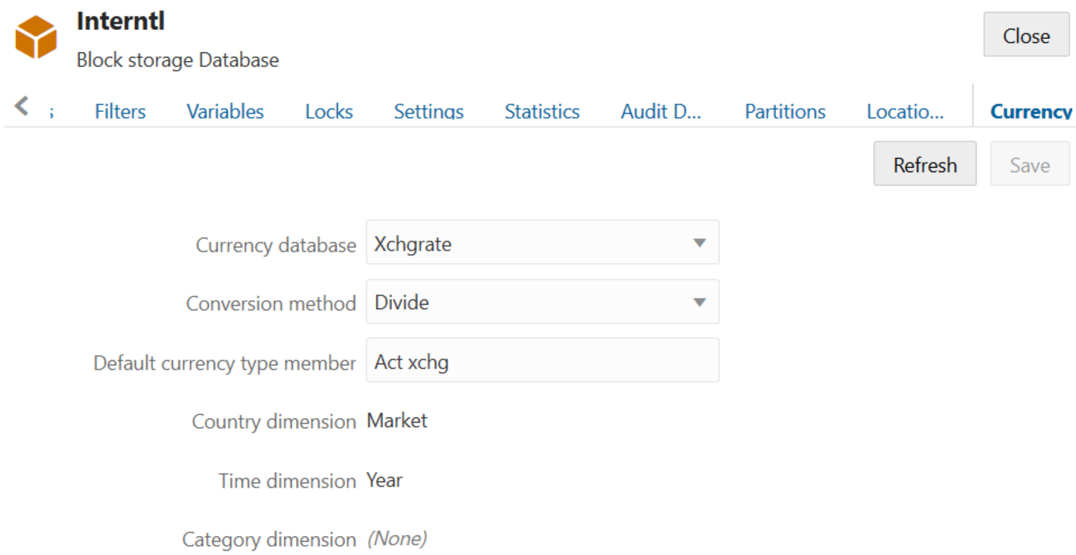
j. Save the outline, if you made changes.

k. If you are using the samples described in this section, the currency cube Sample_Currency.Xchgrate already exists.

If you are working on your own main cube, and a currency cube does not exist yet, generate it now. From the main cube outline viewer, in the Inspect group, click the yen symbol, and provide a name for the currency cube.



3. Link the main and currency cubes, if they are not already linked. The following image of the cube inspector shows that Sample_Currency.Interntl already has a linked currency cube named Xchgrate:



4. Calculate the currency conversion. Using the CCONV command in a calculation script, convert data values from a local currency to a common, converted currency. For example, you might convert data from a variety of currencies into USD. You can convert the data values back to the original, local currencies using the CCONV TOLOCALRATE command.
5. Track currency conversions. Use the CCTRACK application configuration setting to control whether Essbase tracks which currency partitions have been converted, and the exchange rates used.
6. Troubleshoot currency conversions, if needed.

   If you receive an error linking the currency cube to the main cube, ensure that the main cube meets these criteria:

   - A dimension is tagged as time.
   - A dimension is tagged as accounts.
   - The accounts dimension has a currency conversion category defined for the categories of accounts that you wish to convert.
   - All members of the accounts dimension that should not be converted are tagged as No Conversion.
   - A market dimension is tagged as country.
   - The country dimension is assigned a currency name, even if each member in the dimension is individually assigned a currency name.

- Each member of the country dimension has an associated currency name or inherits the currency defined at the dimension level.

Also ensure that the currency cube meets these criteria:

- A dimension is tagged as time.
- The time dimension contains each member that is defined in the time dimension in the main cube. Make sure that each member is included and spelled correctly.
- If you use Dynamic Time Series members in the main cube, you must use them in the currency cube.
- A dimension is tagged as accounts. The accounts dimension in the currency cube contains the account categories defined in the main cube.
- A dimension is tagged as country and contains the names of the currencies used in the dimension tagged as country in the main cube. Make sure that each currency is included and spelled correctly.
- Make sure that the currency cube is started.

10
Designing Partitioned Applications

A partition in Essbase is the region of a database that is shared with another database. Partitioning can provide many benefits. Carefully review the design requirements to help you determine whether partitioning can meet your objectives.

- Understanding Partitioning
- Partition Design Requirements
- Replicated Partitions
- Transparent Partitions
- Federated partitions - see Integrate Essbase with Autonomous Database Using Federated Partitions

Understanding Partitioning

A partition is the region of a database that is shared with another database. An Essbase partitioned application can span multiple servers, processors, or computers.

See:

- Partition Types
- Parts of a Partition
- Data Sources and Data Targets
- Attributes in Partitions
- Version and Encoding Requirements

Partition Types

The following types of partitions are supported in Essbase:

Table 10-1 Partition Types

Partition Type	Description	Applies To
Replicated	A copy of a portion of the data source that is stored in the data target. See Replicated Partitions.	Block storage cubes Aggregate storage cubes
Transparent	Allows users to access data from the data source as though it were stored in the data target. The data is, however, stored at the data source, which can be in another application or Essbase cube, or on another Essbase Server. See Transparent Partitions.	Block storage cubes Aggregate storage cubes

Table 10-1 (Cont.) Partition Types

Partition Type	Description	Applies To
Federated	Federated partitions enable you to integrate Essbase cubes with Autonomous Data Warehouse, to combine Essbase's analytical power with the fast aggregation of Autonomous Database. This chapter does not discuss federated partitions; instead, see Integrate Essbase with Autonomous Database Using Federated Partitions.	Block storage cubes Aggregate storage cubes

Use the information in the following table to help you choose which type of partition to use:

Table 10-2 Features Supported by Partition Type

Feature	Replicated	Transparent
Up-to-the-minute data		x
Reduced network traffic	x	
Reduced disk space		x
Increased calculation speed	x	
Smaller databases		x
Improved query speed	x	
Invisible to end users	x	x
Easier to recover	x	
Ability to query data based on its attributes		x
Ability to use front-end tools that are not distributed OLAP-aware	x	x
Easy to perform frequent updates and calculations		x
Ability to update data at the data target		x
Perform batch updates and simple aggregations	x	

Parts of a Partition

Partitions contain the following parts.

Figure 10-1 Parts of a Partition

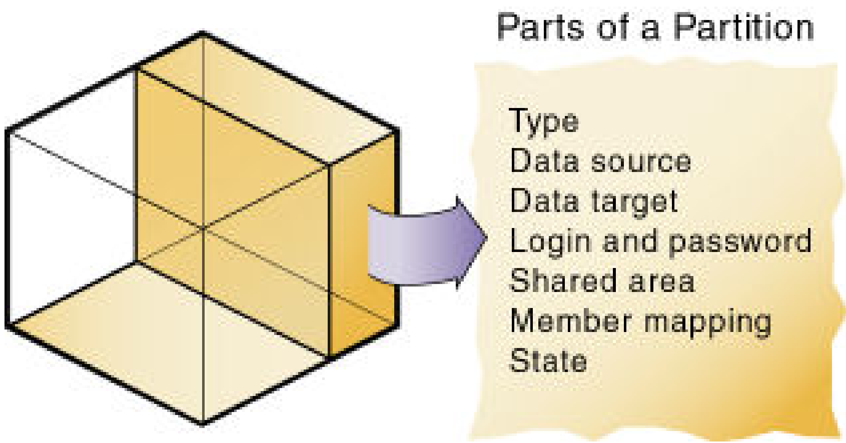


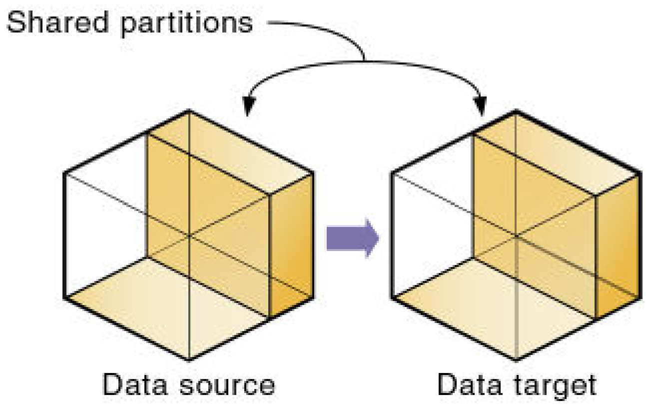
Table 10-3 Parts of a Partition

Part	Description
Type of partition	A flag indicating whether the partition is replicated or transparent.
Data source information	The server, application, and database name of the data source.
Data target information	The server, application, and database name of the data target.
Login and password	The login and password information for the data source and the data target. This information is used for internal requests between the two databases to execute administrative and end-user operations.
Shared areas	A definition of one or more areas, or regions, shared between the data source and the data target. To share multiple noncontiguous portions of a database, define multiple areas in a single partition. This information determines which parts of the data source and data target are shared so that Essbase can put the proper data into the data target and keep the outlines for the shared areas synchronized.
Member mapping information	A description of how the members in the data source map to members in the data target. Essbase uses this information to determine how to put data into the data target if the data target and the data source use different names for some members and dimensions.
State of the partition	Information about whether the partition is up-to-date and when the partition was last updated.

Data Sources and Data Targets

Partitioned databases contain at least one data source (the primary site of the data) and at least one data target (the secondary site of the data). One database can serve as the data source for one partition and the data target for another partition. When defining a partition, you map cells in the data source to their counterparts in the data target.

Figure 10-2 Data Source and Data Target



An Essbase database can contain many partitions, as well as data that is not shared with any other Essbase database. You can define partitions between the following databases:

- Different databases in different applications, as long as each database uses the same language and the same Unicode-related mode.

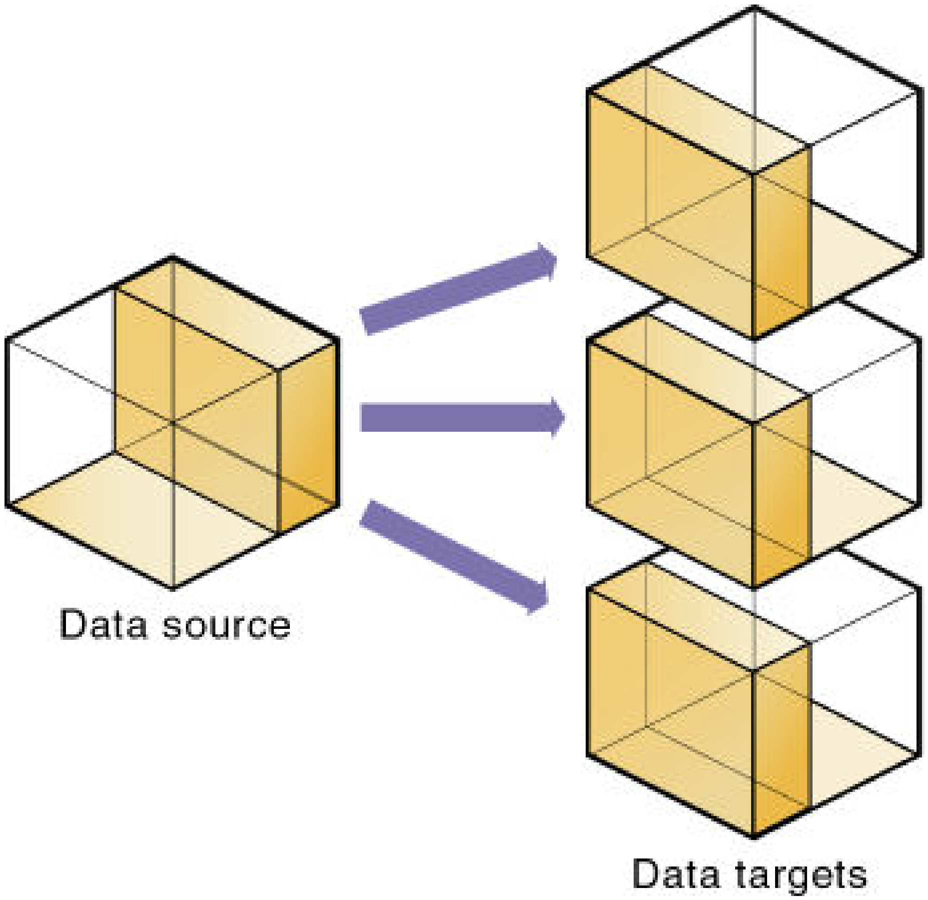
  The applications can be on the same computer or different computers.

- Different databases in one block storage application.

  This practice is not recommended, because the full benefits of partitioning databases are realized when each database is in a separate application.

One database can serve as the data source or data target for multiple partitions. To share data among many databases, create multiple partitions, each with the same data source and a different data target, as illustrated here:

Figure 10-3 Data Shared at Multiple Targets



The following table lists the combinations of block storage and aggregate storage databases as data target and data source that are supported by each partition type:

Table 10-4 Combinations of Data Sources and Data Targets Supported by Partition Type

Source	Target	Replicated	Transparent
Block storage	Block storage	Yes	Yes
Aggregate storage	Block storage	No	Yes
Aggregate storage	Aggregate storage	No	Yes
Block storage	Aggregate storage	Yes	Yes

Attributes in Partitions

For block storage databases, you can use attribute functions for partitioning on attribute values, but you cannot partition an attribute dimension. Use attribute values to partition a database to access members of a dimension according to their characteristics.

For example, in the Sample.Basic database, you cannot partition the Pkg Type attribute dimension, but you can create a partition that contains all the members of the Product dimension that are associated with either or both members (Bottle and Can) of the Pkg Type dimension. If you create a partition that contains members associated with Can, you can access data only on Product members that are packaged in cans; namely, 100-10, 100-20, and 300-30.

> **Note:**
>
> Retrieving data on attribute members of block storage databases may result in missing data. See the entry for "Dense Dynamic Calc members in nonexisting stored blocks" in Comparing Attribute and Standard Dimensions.

You can use the @ATTRIBUTE and @WITHATTR calculation functions to define partitions.

For example, to extract data on all members of the Product dimension that are associated with the Caffeinated attribute dimension, you can create a partition such as @ATTRIBUTE (Caffeinated). But you cannot partition the Caffeinated attribute dimension.

Based on the previous example, this partition is correct:

```
Source                    Target
@ATTRIBUTE(Caffeinated)   @ATTRIBUTE(Caffeinated)
```

This partition is incorrect:

```
Source         Target
Caffeinated    Caffeinated
```

Version and Encoding Requirements

Version and encoding requirements for transparent and replicated partitions:

- Version: Both ends (the source and target) of the partition must be on the same release level.
- Encoding: Both ends of the partition must be in Unicode-mode.

Partition Design Requirements

Use the information in this section to carefully design partitions before implementing them.

- Benefits of Partitioning
- Partitioning Strategies
- Guidelines for Partitioning a Database
- Guidelines for Partitioning Data
- Security for Partitioned Databases

Benefits of Partitioning

Partitioning can provide the following benefits:

- For block storage databases, data synchronization across multiple databases

Essbase tracks changes made to data values in a partition and provides tools for updating the data values in related partitions.

- Ability for user navigation between databases with differing dimensionality

  When users drill across to the new database, they can drill down to more-detailed data.

Partitioning Strategies

Based on user requirements, select a partitioning strategy:

- Partition applications from the top down.

  Use *top-down partitioning* to split a database onto multiple processors, servers, or computers, which can improve the scalability, reliability, and performance of databases. To achieve the best results with top-down partitioning, create a separate application for each partitioned database.

- Partition applications from the bottom up.

  Use *bottom-up partitioning* to manage data flow between multiple related databases, which can improve the quality and accessibility of the data in databases.

- Partition databases according to attribute values associated with base dimensions (a standard dimension associated with one or more attribute dimensions).

  Use this strategy to extract data based on the characteristics of a dimension, such as flavor or size.

> **Note:**
>
> You cannot partition attribute dimensions. See Attributes in Partitions.

Guidelines for Partitioning a Database

Use the following information to help you determine whether to partition a database.

- Partition a database when:
  - The data should be closer to the people who are using it.
  - A single failure would be catastrophic.
  - It takes too long to perform calculations after new data is loaded, and you want to improve performance by spreading calculations across multiple processors or computers.
  - Users want to see the data in different application contexts, and you want to control how users navigate between databases.
  - You need to synchronize information from different sources.
  - You plan to add new organizational units that would benefit from having their own databases.
  - Users must wait as other users access the database.
  - You want to save disk space by giving users access to data stored in a remote location.
  - You want to reduce network traffic by replicating data in several locations.

  - You need to control database outlines from a central location.
  - You need client write-back functionality on an aggregate storage database.
- Do not partition a database when:
  - You have disk space, network bandwidth, and administrative resource concerns.
  - You perform complex allocations where unit level values are derived from total values.
  - You are required to keep all databases online at all times.

    Keeping databases online can be a problem if you have databases in several time zones, because peak user load may differ between time zones. Using transparent partitions exacerbates this problem, but using replicated partitions might help.
  - Databases are in different languages or Unicode-related modes.

    Essbase can partition databases only if each database uses the same language, or each database uses the same Unicode or non-Unicode mode.

Guidelines for Partitioning Data

When designing a partitioned database, use the following information to help you determine which data to include in each partition:

- Which database should be the data source and which the data target? The database that “owns” the data, where the data is updated and where most of the detail data is stored, should be the data source.
- Are some parts of the database accessed more frequently than others?
- What data can you share among sites?
- How granular must the data be at each location?
- How frequently is the data accessed, updated, or calculated?
- What are the available resources: disk space, CPUs, and network resources?
- How much data must be transferred over the network? How long does it take?
- Is the data stored in one or multiple locations?
- Is the data accessed in one or multiple locations?
- Is there information in separate databases that should be accessed from a central location? How closely are groups of data related?

Security for Partitioned Databases

Users accessing partitions may need to view data stored in multiple databases. Set up security so that users do not view or change inappropriate data.

1. Grant users access to the source and target cubes.
2. Restrict user access by setting up filters (on the source and target cubes) to the partitioned slices only.

   See Controlling Access to Database Cells Using Security Filters.
3. If you are creating a replicated partition, determine whether users can make changes to a replicated partition at the data target. The update setting (which allows or disallows updates) overrides user filters that allow users to update data.

When creating replicated partitions using the create replicated partition MaxL statement, if you do not specify the **update allow** grammar, replicated partitions cannot be updated by default.

Replicated Partitions

A replicated partition is a copy of a portion of the data source that is stored in the data target. Some users can then access the data in the data source while others access it in the data target.

For example, in the Samppart and Sampeast sample applications, the DBA at The Beverage Company (TBC) created a replicated partition between the East database and the Company database containing Actual, Budget, Variance, and Variance%. Users in the eastern region now store their budget data locally. Because they do not have to retrieve this data live from corporate headquarters, response times are faster, and they have more control over the downtimes and administration of local data.

Changes to the data in a replicated partition flow from the data source to the data target. Changes made to replicated data in the data target do not flow back to the data source. If users change the data at the data target, Essbase overwrites their changes when the DBA updates the replicated partition.

When a replicated partition is defined, the DBA can select a setting to prevent the data in the replicated portion of the data target from being updated. The update setting (which allows or disallows updates) takes precedence over access provided by security filters and is also honored by batch operations, such as data load and calculation.

Use a replicated partition to achieve any of the following goals:

- Decrease network activity
- Decrease query response times
- Decrease calculation times
- Recover more easily from system failures

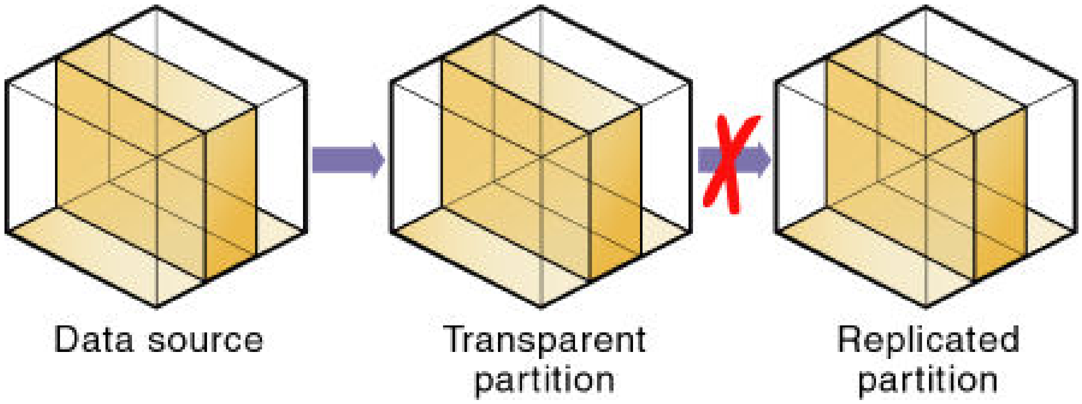
Rules for Replicated Partitions

Replicated partitions must follow these rules:

- You must be able to map the shared replicated areas of the data source and data target outlines, although the shared areas need not be identical. You must tell Essbase how each dimension and member in the data source maps to each dimension and member in the data target.

  The data source and data target outlines for the non-shared areas do not have to be mappable.
- Because none of the areas that you use as a replicated partition target can come from a transparent partition source, you cannot create a replicated partition on top of a transparent partition, as shown in the illustration:

Figure 10-4 Invalid Replicated Partition



- The cells in the data target of a replicated partition cannot come from two data sources; the cells in one partition must come from one database. To replicate cells from multiple databases, create a different partition for each data source.

  The cells in a data target can be the data source for a different replicated partition. For example, if the Samppart.Company database contains a replicated partition from the Sampeast.East database, you can replicate the cells in Sampeast.East into a third database, such as Sampwest.West.
- You cannot use attribute members to define a replicated partition. For example, associated with the Market dimension, the Market Type attribute dimension members are Urban, Suburban, and Rural. You cannot define a partition on Urban, Suburban, or Rural, because a replicated partition contains dynamic data, not stored data. Therefore, an attempt to map attributes in replicated partitions results in an error message. However, you can use the WITHATTR command to replicate attribute data.

Advantages of Replicated Partitions

- Because data is stored closer to end users, in the data target, replicated partitions can decrease network activity, resulting in improved retrieval times for users.
- The data is more easily accessible to all users. Some users access the data at the data source, others at the data target.
- Failures are not as catastrophic. Because the data is in multiple places, if one database fails, only the users connected to that database are unable to access the information. Data is still available at and can be retrieved from the other sites.
- Local DBAs can control the downtime of their local databases. For example, because users in the eastern region are accessing their own replicated data instead of the Company database, DBAs can bring down the Company database without affecting users in the eastern region.
- Because only the relevant data is kept at each site, databases can be smaller. For example, users in the eastern region can replicate only the eastern budget information, instead of accessing a larger company database containing budget information for all regions.

Disadvantages of Replicated Partitions

- You need more disk space because data is stored in multiple locations.
- Because the DBA must manually refresh data regularly, users may not see the latest version of the data.

Performance Considerations for Replicated Partitions

To improve the performance of replicated partitions, follow these guidelines:

- Do not replicate members that are dynamically calculated in the data source, because Essbase must probe the outline to find dynamically calculated members and their children to determine how to perform the calculation.
- Do not replicate derived data from the data source. Instead, replicate the lowest practical level of each dimension and perform the calculations on the data target after you complete the replication.

  For example, to replicate the database along the Market dimension:

  - Define the shared area as the lowest-level members of the Market dimension that you care about, for example, East, West, South, and Central and the level 0 members of the other dimensions.
  - After you complete the replication, calculate the values for Market and the upper-level values in the other dimensions at the data target.

    Sometimes you cannot calculate derived data at the data target. In that case, replicate it from the data source. For example, you cannot calculate derived data at the data source if the data meets any of the following criteria:

    - Requires that data outside the replicated area be calculated.
    - Requires calculation scripts from which you cannot extract only the portion to be calculated at the data target.
    - Is being replicated onto a computer with little processing power, such as a laptop.
- Partitioning along a dense dimension takes longer than partitioning along a sparse dimension. When Essbase replicates data partitioned along a dense dimension, it must access every block in the data source and then create each block in the data target during the replication operation.
- You cannot replicate data into a member that is dynamically calculated at the data target. Essbase does not load or replicate into Dynamic Calc members, because these members do not contain data until a user requests it at runtime. Essbase avoids sending replicated data for both dynamic dense and dynamic sparse members on the replication target, because this data is not stored on the data target.

  To replicate only the data values that have changed instead of the entire partition, see Populating or Updating Replicated Partitions.

Replicated Partitions and Port Usage

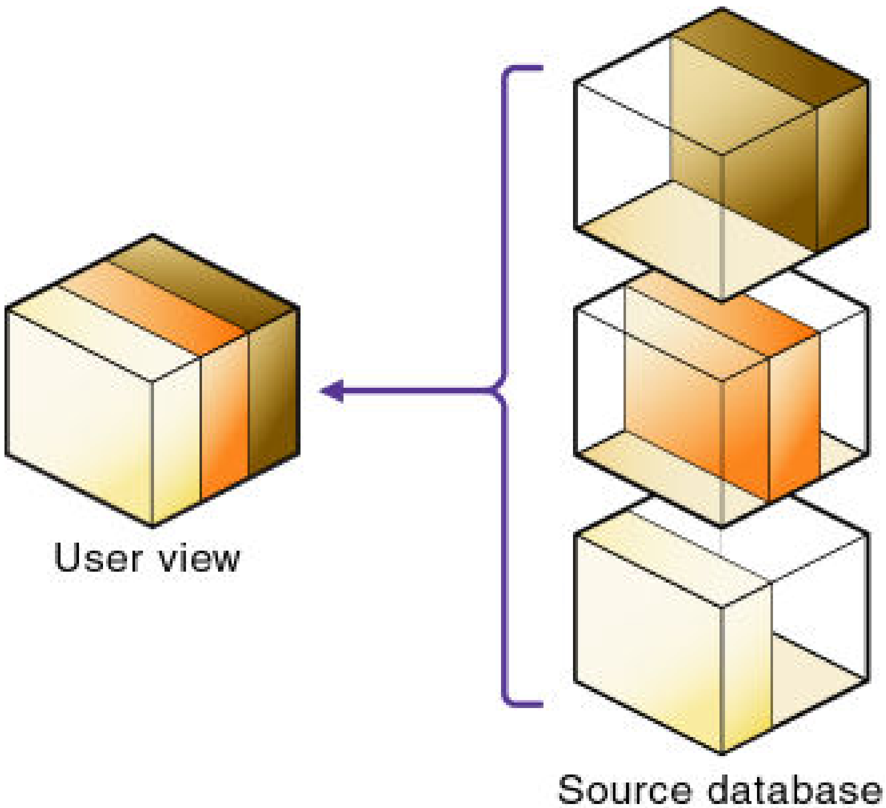
With replicated partitions, users connect to the target database only. When data is updated on the target database, the process of replicating data from the source database to the target database utilizes one port and this connection is based on the user name declared in the partition definition (partition user).

> **Note:**
>
> Because of the short-term nature of replication, replicated partitions and ports are rarely a problem.

Transparent Partitions

A transparent partition allows users to manipulate data that is stored remotely as if it were part of the local database. The remote data is retrieved from the data source each time that users at the data target request it. Users do not need to know where the data is stored, because they see it as part of their local database.

Figure 10-5 Transparent Partitions



Because data is retrieved directly from the data source, users see the latest version. When they update the data, their updates are written back to the data source. This process means that other users at the data source and the data target have immediate access to those updates.

With a transparent partition, users at the data source and at the data target may notice slower performance as more users access the source data.

For example, the DBA at TBC can use a transparent partition to calculate each member of the Scenario dimension on a separate computer. This process reduces the elapsed time for the calculation while providing users with the same view of the data.

Use a transparent partition to achieve the following goals:

- Show users the latest version of the data
- Allow users at the data target to update data
- Decrease disk space

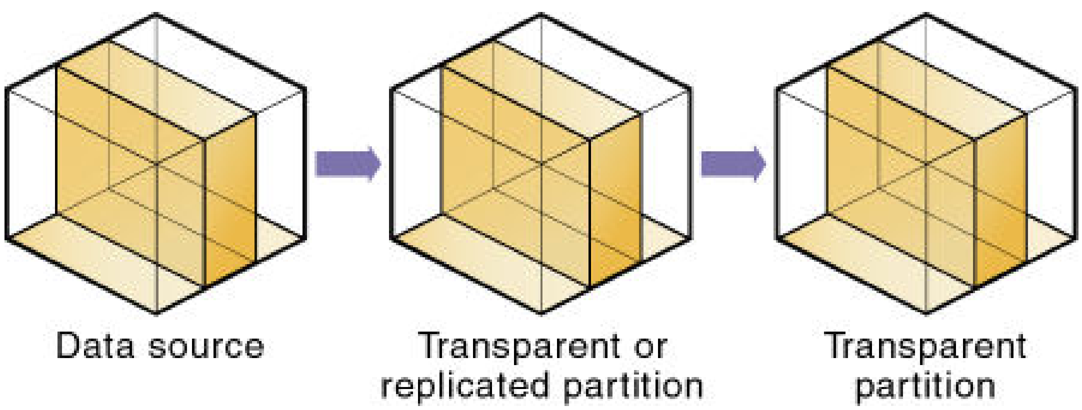
When you create a transparent partition, the data slice in the target cube is cleared to #MISSING, as the data is expected to be stored on the source cube. It remains cleared even if you delete the partition.

Rules for Transparent Partitions

Transparent partitions must follow these rules:

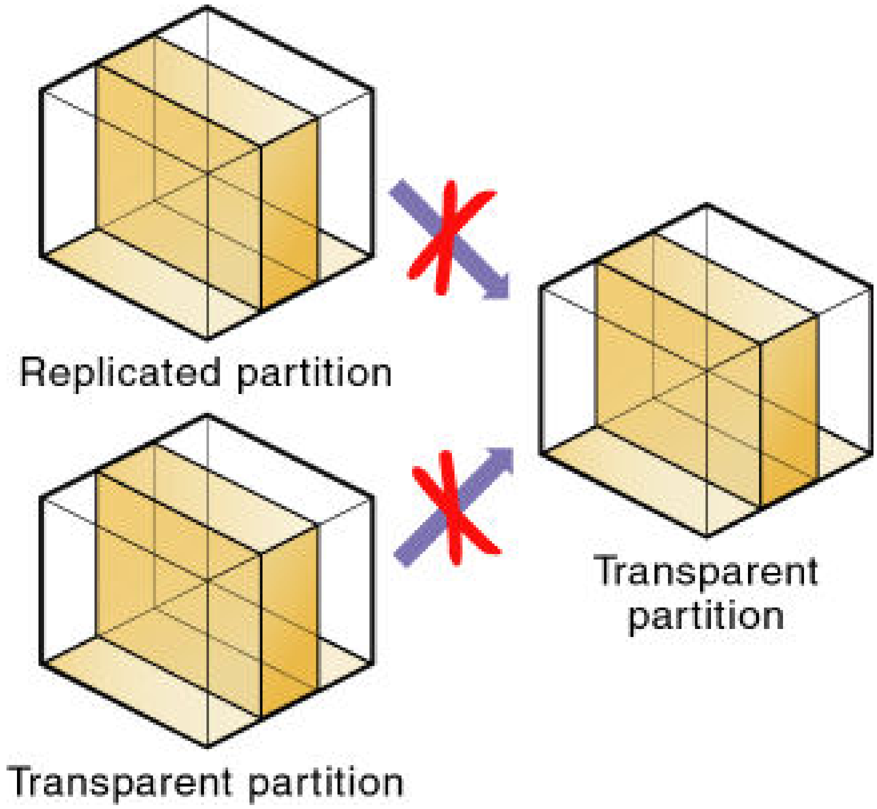
- The shared transparent areas of the data source and data target outlines need not be identical, but you must be able to map the dimensions in them. You must tell Essbase how each dimension and member in the data source maps to each dimension and member in the data target.
- The data source and data target outlines for the nonshared areas need not be mappable, but attribute associations must be identical. Otherwise, users can get incorrect results for some retrievals. For example, if product 100-10-1010 is associated with the Grape Flavor attribute on the source, but product 100-10-1010 is not associated with Grape on the target, the total of sales for all Grape flavors in New York is incorrect.
- The partition definition must contain only stored members. You cannot use attribute dimensions or members to define a transparent partition. For example, the Market Type attribute dimension, which is associated with the Market dimension, has members Urban, Suburban, and Rural. You cannot define a partition on Urban, Suburban, or Rural.
- If a cell is mapped from the data source to an aggregate storage database as the target, all the cell's dependents must also be mapped to the same partition definition.
- You can create a transparent partition on top of a replicated partition. In other words, you can create a transparent partition target using a replicated partition source, as shown in this illustration:

Figure 10-6 Valid Transparent Partition



- As illustrated below, you cannot create a transparent partition on top of multiple other partitions. In other words, you cannot create a transparent partition target from multiple sources because each cell in a database must be retrieved from only one location—either the local disk or a remote disk.

Figure 10-7 Invalid Transparent Partition



- Carefully consider any formulas you assign to members in the data source and data target.

Advantages of Transparent Partitions

Transparent partitions can solve many database problems, but transparent partitions are not always the ideal partition type.

- You need less disk space, because you are storing the data in one database.
- The data accessed from the data target is always the latest version.
- When the user updates the data at the data source, Essbase makes those changes at the data target.
- Individual databases are smaller, so they can be calculated more quickly.
- The distribution of the data is invisible to the end user and the end user's tools.
- You can load the data from either the data source or data target.

Disadvantages of Transparent Partitions

If these disadvantages are too serious, consider using replicated partitions instead.

- Transparent partitions increase network activity, because Essbase transfers the data at the data source across the network to the data target. Increased network activity results in slower retrieval times for users.
- Because more users are accessing the data source, retrieval time may be slower.

- If the data source fails, users at both the data source and the data target are affected. Therefore, the network and data source must be available whenever users at the data source or data target need them.
- You can perform some administrative operations only on local data. For example, if you archive the data target, Essbase archives only the data target and not the data source. The following administrative operations work only on local data in block storage databases:
  - CLEARDATA calculation command
  - DATACOPY calculation command
  - EXPORT command
  - VALIDATE command
  - BEGINARCHIVE and ENDARCHIVE commands
- When you perform a calculation on a transparent partition, Essbase performs the calculation using the current values of the local data and transparent dependents. Essbase does not recalculate the values of transparent dependents, because the outlines for the data source and the data target may be so different that such a calculation is inaccurate. To calculate all partitions, issue a CALC ALL command for each individual partition, and then perform a CALC ALL command at the top level using the new values for each partition.

  Consider this example:
  - The data target outline contains a Market dimension with East, West, South, and Central members
  - The data source outline contains an East dimension with New York and New Jersey members

  If you tried to calculate the data target outline, you would assume that East was a level 0 member. In the data source, however, East is derived by adding New York and New Jersey. Any calculations at the data target, however, would not know this information and could not reflect changes made to New York and New Jersey in the data source. To perform an accurate calculation, therefore, calculate East in the data source and then calculate the data target.
- Formulas assigned to members in the data source may produce calculated results that are inconsistent with formulas or consolidations defined in the data target, and vice versa.

Performance Considerations for Transparent Partitions

To improve the performance of transparent partitions, consider the following guidelines when creating the partition:

- Partitioning along dense dimensions in a transparent partition can greatly slow performance, because dense dimensions are used to determine the structure and contents of data blocks. If a database is partitioned only along a dense dimension at the target, Essbase must compose data blocks by performing network calls for the remote data in the transparent partition and to the disk I/O for the local portion of the block.

  To improve performance, consider including one or more sparse dimensions in the area definition so that the number of blocks required is limited to combinations with the sparse members.
- Basing transparent partitions on the attribute values of a dimension can increase retrieval time, because attributes are associated with sparse dimensions. In such cases, partitioning at a level higher than the level that is associated with attributes improves retrieval time. For example, in the Product dimension of the Sample.Basic database, if children 100-10,

200-10, and 300-10 (level 0) are associated with attributes, then partition their parents 100, 200, and 300 (level 1) for better retrieval performance.

- Loading data into the data source from the data target can greatly slow performance. If possible, load data into the data source locally.
- Retrieval time is slower because users access the data over the network.
- When the transparent partition target is an aggregate storage database, you can specify the maximum size of the request grid and the response grid, using the MAX_REQUEST_GRID_SIZE and MAX_RESPONSE_GRID_SIZE configuration settings.
- Partitioning base dimensions can greatly slow performance.
- See Performance Considerations for Transparent Partition Calculations.

Calculating Transparent Partitions

When calculating local data that depends on remote data, Essbase must use bottom-up calculation. Be sure you are using an optimized calculator cache on the target cube.

When you perform a calculation on a transparent partition, Essbase performs the calculation using the current values of the local data and transparent dependents. When calculating local data that depends on remote data, Essbase performs a bottom-up calculation. The bottom-up calculation can be done only if the calculator cache on the target database is used properly. Refer to Bottom-Up and Top-Down Calculation.

Increasing the memory assigned to the calculator cache greatly improves calculation performance with transparent partitions. When a calculation is started, a message in the application log file indicates whether the calculator cache is enabled or disabled on the target database. Using the calculator cache on the target database reduces the number of blocks that are requested from the data source during calculation. Reducing the blocks requested, in turn, reduces the network traffic that is generated by transferring blocks across the network.

Performance Considerations for Transparent Partition Calculations

Calculating data on the target of a transparent partition can slow performance when Essbase must retrieve dependent blocks across the network from the source before calculating. To optimize transparent calculations, use Dynamic Calc, manage the calculator cache, avoid top down formulas, and avoid complex formulas on area-defining members.

Performance with transparent calculations also may slow if Essbase must perform a top-down calculation on any portion of the data target that contains top-down member formulas. When the data target does not contain top-down member formulas, Essbase can perform a bottom-up calculation on the data target, which is much faster.

When Essbase performs the calculation on the data source, it can always perform a bottom-up calculation. For a comparison of top-down and bottom-up calculations, refer to Bottom-Up and Top-Down Calculation.

Consider using these calculation alternatives:

- If you are absolutely sure that a target partition calculation script does not involve access to remote data, you can use the SET REMOTECALC OFF calculation command in the calculation script to stop retrieval efforts from the source partition.
- Implement Dynamic Calc members as parents of the transparent data so that the data is calculated on the fly when it is retrieved. This process reduces the batch processing time. Essbase performs the calculation only when users request it.

- Implement a replicated layer between the low-level transparent data and high-level local data.

Consider these performance strategies:

- Keep the partition fully within the calculator cache area, which means that any sparse members in the partition definition must be contained within the calculator cache. For example, in the Sample.Basic cube, if a partition definition includes @IDESC(East), all descendants of East must be within the calculator cache.
- Enable the calculator cache, and assign a sufficient amount of memory to it.
- Do not use complex formulas on any members that define the partition. For example, in Sample.Basic, assigning a complex formula to New York or New Jersey (both children of East) forces Essbase to use the top-down calculation method. Refer to Bottom-Up and Top-Down Calculation.

Transparent Partitions and Member Formulas

If the data target and data source outlines are identical except for different member formulas, ensure that the partition definition produces the calculation results you want.

For example, suppose that the data source and data target outlines both contain a Market dimension with North and South members, and children of North and South. On the data target, Market is calculated from the data for the North and South members (and their children) on the data source. If any of these members on the data source contains member formulas, these formulas are calculated, affecting the calculated value of Market on the data target. These results may be different from how the Market member are calculated from the North and South members on the data target, where these formulas may not exist.

Ensure that any formulas you assign to members in the data source and data target produce the results you want.

Transparent Partitions and Port Usage

One port is used for every unique user and machine combination. If a user defines several transparent partitions on one server, using the same user name, then only one port is occupied.

In a transparent partition, when a user (user1) drills into an area in the target that accesses source data, user1 is using the user name declared in the partition definition (partition user) to access the data from the source database. This access causes the use of an additional port because different users (user1 and partition user) are connecting to the application.

If a second user (user2) connects to the target database and drills down to access source data, user2 also uses the user name declared in the partition definition (partition user). Because the partition user is already connected to the source database, an additional port is not needed for the partition user, as long as user2 is accessing the same source database.

11 Creating and Maintaining Partitions

To share data across multiple Essbase cubes, you can connect them using a replicated or transparent partition. Replicated and transparent partition definitions identify a source and target database and help you manage the flow between them. To integrate Essbase cubes with Autonomous Data Warehouse, use federated partitions.

Partition creation requires Database Manager permissions or higher. After you have created a partition, load and calculate the database that contains the partition. Loading and calculating the partition may require you to change existing rules files and calculation scripts.

For information about using the Essbase web interface to create and maintain partitions, see Link Cubes Using Partitions and XREF/XWRITE.

- Choosing a Partition Type
- Setting up a Partition Data Source
- Setting the User Name and Password
- Defining a Partition Area
- Mapping Members in Partitions
- Validating Partitions
- Populating or Updating Replicated Partitions
- Viewing Partition Information
- Partitioning and SSL
- Troubleshooting Partitions

Choosing a Partition Type

Decide which type of partition to create:

- Replicated
- Transparent
- Federated

See Partition Types.

Setting up a Partition Data Source

You define partitions in the context of the data target. You must be logged in on the target cube. Select your data source, which can be another cube, or a federated source.

Single-Instance Partitioning – All source cubes and the target cube are in the same Essbase instance. You select a source cube from the list of available cubes in the instance. There is no need to specify any host connection information, as everything within the same instance is inferred.

Cross-Instance Partitioning – At least some source cubes are from a different Essbase instance than the target cube. To access these sources from the target, you must first create a connection, and select it while creating the partitions.

See also:

- Link Cubes Using Partitions and XREF/XWRITE
- Use Connections and Datasources

Setting the User Name and Password

You must specify a user name and password for Essbase to use when it communicates between the data source and the data target. The user name and password must be identical on the data source and the data target. Essbase uses this user name and password to:

- Transfer data between the data source and the data target for replicated and transparent partitions. Local security filters apply to prevent end users from seeing privileged data.
- Synchronize database outlines for all partition types.

See Security for Partitioned Databases.

Defining a Partition Area

You can define or edit the areas of the data source to share with the data target in a partition. An area is a subcube within an Essbase cube, and a partition comprises one or more areas. For example, an area could be all Measures at the lowest level for Actual data in the Eastern region.

When you define a replicated area, ensure that the data source and data target contain the same number of cells. The shape of the two partitions must match. For example, if the area in the data source covers 18 cells, the data target should contain an area covering 18 cells into which to put those values. The cell count does not include the cells of attribute dimensions.

> **Note:**
>
> Use member names instead of their aliases to create area definitions. Although Essbase validates the aliases, the partitions will not work.

You define partition areas when you create the partition. For instructions, see the topic that pertains to your partition type:

- Create a Transparent Partition
- Create a Replicated Partition
- Create a Federated Partition

Mapping Members in Partitions

To create a partition, Essbase must be able to map all shared data source members to data target members. Oracle recommends that data source member names and data target member names are the same to reduce maintenance requirements for the partition, especially when the partition is based on member attributes.

If the data source and data target contain the same number of members and use the same member names, Essbase automatically maps the members. You need only validate, save, and test the partitions. If Essbase cannot map automatically, you must map manually.

Map data source members to data target members in any of the following ways:

- Enter or select member names manually. (When you type a duplicate member name, type the qualified member name and enclose it in double quotation marks; for example, `"[State].[New York]"`
- Import the member mappings from an external data file.
- Create area-specific mappings.

Topics in this section:

Mapping Members with Different Names

If the data source outline and data target outline contain different members, or if the members have different names in each outline, you must map the data source members to the data target members. In the following example, the first two member names are identical, but the third member name is different:

```
Source       Target
Product      Product
   Cola         Cola
Year         Year
   1998         1998
Market       Market
   East         East_Region
```

Because you know that East in the data source corresponds to East_Region in the data target, map East to East_Region. Then, all references to East_Region in the data target point to East in the data source. For example, if the data value for Cola, 1998, East is 15 in the data source, the data value for Cola, 1998, East_Region is 15 in the data target.

Mapping Data Cubes with Extra Dimensions

The number of dimensions in the data source and data target may vary. The following example illustrates a case where there are more dimensions in the data source outline than in the data target outline:

```
Source       Target
Product      Product
   Cola         Cola
Market       Market
   East         East
Year
   1999
```

```
   1998
   1997
```

You can map member 1997 of the Year dimension to Void in the data target. First, define the areas of the data source to share with the data target:

Source	Target
@DESCENDANTS(Market), 1997	@DESCENDANTS(Market)

Then, map the data source member to Void in the data target:

Source	Target
1997	Void

"Void" is displayed automatically; manually entering "Void" may cause errors.

If you do not include at least one member from the extra dimension in the area definition, you will receive an error message when you attempt to validate the partition.

> **Note:**
>
> When you map a member from an extra dimension, the partition results reflect data only for the mapped member. In the above example, the Year dimension contains three members: 1999, 1998, and 1997. If you map member 1997 from the data source to the data target, the partition results reflect Product and Market data only for 1997. Product and Market data for 1998 and 1999 will not be extracted.

The following example illustrates a case where the data target includes more dimensions than the data source:

```
Source        Target
Product       Product
   Cola          Cola
              Market
                 East
Year          Year
   1997          1997
```

In such cases, first define the shared areas of the data source and the data target:

Source	Target
@IDESCENDANTS(Product)	@IDESCENDANTS(Product), East

You can then map member East from the Market dimension of the data target to Void in the data source:

Source	Target
Void	East

If member East from the Market dimension in the data target is not included in the target areas definition, you will receive an error message when you attempt to validate the partition.

Mapping Shared Members

When you create a replicated or transparent partition using a shared member, use the prototype member names in the mapping. Essbase maps the prototype member from the data source.

Mapping Attributes Associated with Members

You must accurately map attribute dimensions and members from the data source to the data target to ensure that the partition is valid.

You can map attributes in transparent partitions. See Attributes in Partitions.

In the following example, the outline for the data source contains a Product dimension with a member 100 (Cola). Children 100-10 and 100-20 are associated with member TRUE of the Caffeinated attribute dimension, and child 100-30 is associated with member FALSE of the Caffeinated attribute dimension.

The data target outline has a Product dimension with a member 200 (Cola). Children 200-10 and 200-20 are associated with member Yes of the With_Caffeine attribute dimension, and child 200-30 is associated with No of the With_Caffeine attribute dimension.

First define the areas to be shared from the data source to the data target:

```
Source              Target
@DESCENDANTS(100)   @DESCENDANTS(200)
@DESCENDANTS(East)  @DESCENDANTS(East)
```

Then map the attributes:

```
Source              Target
100-10              200-10
100-20              200-20
100-30              200-30
Caffeinated         With Caffeine
Caffeinated_True    With_Caffeine_True
Caffeinated_False   With_Caffeine_False
```

If you map attribute Caffeinated_True to attribute With_Caffeine_No, you receive an error message during validation. You must associate caffeinated cola from the data source to caffeinated cola in the data target.

An attribute dimension or an attribute member can exist in the outline of the data source but not in the outline of the data target, or in the outline of the data target but not in the outline for the data source. For example:

```
Source        Target
Caffeinated
   True
   False
```

In such cases, you have the following choices:

- Create the Caffeinated attribute dimension and its members in the outline of the data target and associate them with the Product dimension. You can then map the attributes from the data source to the data target.
- Map the Caffeinated attribute dimension in the data source to Void in the data target.

Creating Advanced Area-Specific Mappings

If you can map all of the members in your data source to their counterparts in the data target using standard member mapping, you need not perform advanced area-specific mapping.

If, however, you need to control how Essbase maps members at a more granular level, you may need to use area-specific mapping, which maps members in one area to members in another area only in the context of a particular area map.

Use area-to-area mapping to do the following:

- Map data differently depending on where it is coming from.
- Map multiple members in the data source to a single member in the data target.

Because Essbase cannot determine how to map multiple members in the data source to a single member in the data target, you must logically determine how to divide your data until you can apply one mapping rule to that subset of the data. Then use that rule in the context of area-specific mapping to map the members.

Example 11-1 Example 1: Advanced Area-Specific Mapping

The data source and data target contain the following dimensions and members:

```
Source        Target
Product       Product
   Cola          Cola
Market        Market
   East          East
Year          Year
   1998          1998
   1999          1999
              Scenario
                 Actual
                 Budget
```

The data source does not have a Scenario dimension. Instead, it assumes that past data is actual data and future data is forecast, or budget, data.

You know that 1998 in the data source should correspond to 1998, Actual in the data target and 1999 in the data source should correspond to 1999, Budget in the data target. So, for example, if the data value for Cola, East, 1998 in the data source is 15, the data value for Cola, East, 1998, Actual in the data target should be 15.

Because mapping works on members, not member combinations, you cannot simply map 1998 to 1998, Actual. Define the area (1998 and 1998, Actual) and then create area-specific mapping rules for that area.

Because the data source does not have Actual and Budget members, you also must map these members to Void in the data target.

Example 11-2 Example 2: Advanced Area-Specific Mapping

You also can use advanced area-specific mapping if the data source and data target are structured very differently but contain the same kind of information.

This strategy works, for example, if your data source and data target contain the following dimensions and members:

```
Source      Target
Market      Customer_Planning
   NY            NY_Actual
   CA           NY_Budget
                CA_Actual
                CA_Budget
Scenario
   Actual
   Budget
```

You know that NY and Actual in the data source should correspond to NY_Actual in the data target and NY and Budget in the data source should correspond to NY_Budget in the data target. So, for example, if the data value for NY, Budget in the data source is 28, the data value for NY_Budget in the data target should be 28.

Because mapping works on members, not member combinations, you cannot simply map NY, Actual to NY_Actual. Define the area (NY and Actual, and NY_Actual) and then create area-specific mapping rules for that area.

Because the data target does not have NY and CA members, you must also map these members to Void in the data target so that the dimensionality is complete when going from the data source to the data target.

Validating Partitions

When you create a partition, validate it to ensure its accuracy before you use it. Database Manager permissions or higher are required. After you validate, save the partition definition. If necessary, you can edit an existing partition.

When Essbase validates a partition definition, it checks on the Essbase Server for the data source and the data target to ensure that:

- The area definition is valid (contains no syntax errors).
- The specified data source members are valid and map to valid members in the data target.
- All connection information is correct; that is, the server names, database names, application names, user names, and password information.
- For replicated and transparent partitions, a replication target does not overlap with a replication target; a replication target does not overlap with a transparent target; and a transparent target does not overlap with a transparent target.
- For replicated and transparent partitions, the cell count for the partition is the same on the data source and the data target.
- For replicated and transparent partitions, the area dimensionality matches the data source and the data target.
- You must validate a transparent partition that is based on attribute values to ensure that the results are complete. Essbase does not display an error message when results are incomplete.

Populating or Updating Replicated Partitions

How frequently you update replicated partitions depends on how current data needs to be. Essbase tracks when the data source was last changed and when the data target was last updated. The administrator of either the source site or target site can be responsible for replicating data.

Essbase also tracks which cells in a partition are changed:

- Faster—Only the cells that have changed since the last replication
- Slower—All cells

  The slower update is useful under certain conditions; for example, updating all cells to recover lost data at the target.

Follow these guidelines:

- Unless you update all cells, replication does not update target data when the source data has not changed since the last replication.
- By default, Essbase replicates #MISSING cells.
- If you do not want to replicate #MISSING cells, you can use the DISABLEREPLMISSINGDATA configuration setting.
- If you deleted data blocks on the data source, Essbase updates all data cells at the data target, even if you choose to update only changed cells. You can delete data blocks at the data source using any of these methods:
  - Using the CLEARDATA command in a calculation script
  - Using “Clear combinations” in your rules file during a data load
  - Restructuring the database keeping only level 0 or input data
  - Deleting sparse members

You can replicate:

- All data targets connected to a data source.

  For example, if you replicate all data targets connected to the Sampeast.East database, Essbase updates the Budget, Actual, Variance, and Variance % members in the Samppart.Company database:
- From all data sources connected to a data target.

  For example, if you replicate from all data sources connected to the Samppart.Company database, Essbase pulls the Budget, Actual, Variance, and Variance % members from the Sampeast.East database and updates them in the Samppart.Company database.

To update a replicated partition, you can use the **refresh replicated partition** MaxL statement.

Viewing Partition Information

To view information about a partition, you can use the **display partition** MaxL statement.

For information about using the Essbase web interface for partitioning, see Link Cubes Using Partitions and XREF/XWRITE.

Partitioning and SSL

The following considerations apply when partitioning in secure (SSL) mode:

- The partition source and target must have the same security protocol; for example, both or neither use SSL.
- To enable Essbase to use SSL connectivity, you must set ENABLESECUREMODE to TRUE.
- Consider setting CLIENTPREFERREDMODE to SECURE.

  If CLIENTPREFERREDMODE is not set, or is set to FALSE, but ENABLESECUREMODE is set to TRUE, you can securely create and refresh partitions in MaxL by adding `:secure` to the HOST-NAME string. For example:

  ```
  login esbuser esbpassword on "localhost:6423:secure";
  ```

Troubleshooting Partitions

The following table lists common problems that you may encounter when using partitions.

Table 11-1 Troubleshooting Problems with Partitions

Symptom	Possible Cause	Solution
When replicating to multiple data targets, some are not replicated.	The connection between the data source and one of the data targets was lost during the replication operation.	Retry the replication operation. If one database is unavailable, replicate into only the available databases.
A new or recently changed partition is validated and saved but does not function.	The partition may have a circular dependency. If database A is the source for database B, database B cannot be source for database A for the same slice.	Edit the partition definition to remove the circular dependency.
When you try to access a partition, you cannot connect to it.	Someone has deleted, renamed, or moved the application containing the database to which you are trying to connect.	Edit the partition having problems to specify the new application name or location.
Essbase overwrites user edits.	Users are changing data at a replicated partition that you overwrite each time you update the partition.	Set the partition to disallow user updates, or explain to users why their data disappears.
Data is confusing.	Your partition may not be set up correctly.	Check your partition to ensure that you are partitioning the data that you need.

12
Overview of Data Load and Dimension Build

Essbase **data load** is the process of moving numeric data from external sources, such as files or relational databases, into an Essbase cube. Essbase **dimension build** is the process of populating the cube outline with new or updated metadata that classifies the data into logical business categories.

- Data Load and Dimension Build Workflow
- Sources of Data
- Load Rules
- Security and Multiple-User Considerations

The information in this chapter applies to block storage and aggregate storage cubes. Because some rule file options and data source requirements vary for aggregate storage cubes, also refer to:

- Build Dimensions in Aggregate Storage Cubes
- Load Data into Aggregate Storage Cubes

Data Load and Dimension Build Workflow

To load data or metadata (members) to Essbase, you start by identifying a source of data. In most cases, you need to design a load rule. Finally, you perform the data load or dimension build using a job, a CLI command, or an API.

The following workflow applies when you load data values or dimensions and members into an Essbase cube:

1. Establish access to an external source of data. The source may be an external database, a file you upload to the Essbase server, or part of a formatted Excel file (application workbook).

   See Sources of Data.

2. Define a load rule. Load rules are a set of operations you tell Essbase to perform on data and metadata to manage how it loads into the cube.

   See Load Rules.

3. Perform the data load or dimension build. One way is by using Jobs; see Execute Jobs for details. Alternatively, you can use CLI; see CLI Command Reference.

Sources of Data

During a data load or dimension build, Essbase reads the source of data from the top, proceeding left to right through records, fields, and delimiters. Sources of data may contain data values and metadata, and can be text files, relational tables, or export files from cubes.

Source data contains the information that you want to load into the Essbase cube. A source of data can contain data values as well as metadata. Metadata is information about members, such as names, aliases, formulas, and other properties.

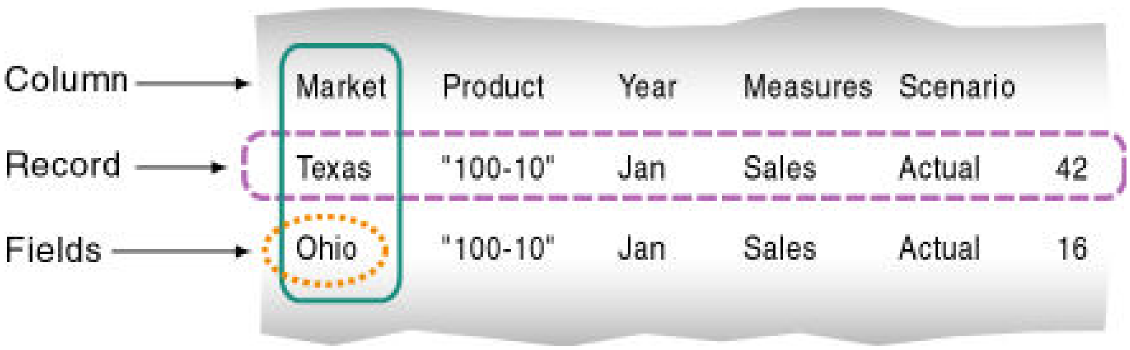
- Items in a Source of Data
- Supported Source Data Types

Items in a Source of Data

During a data load, Essbase reads the source of data starting at the top and proceeding from left to right. A source of data contains records, fields, and field delimiters.

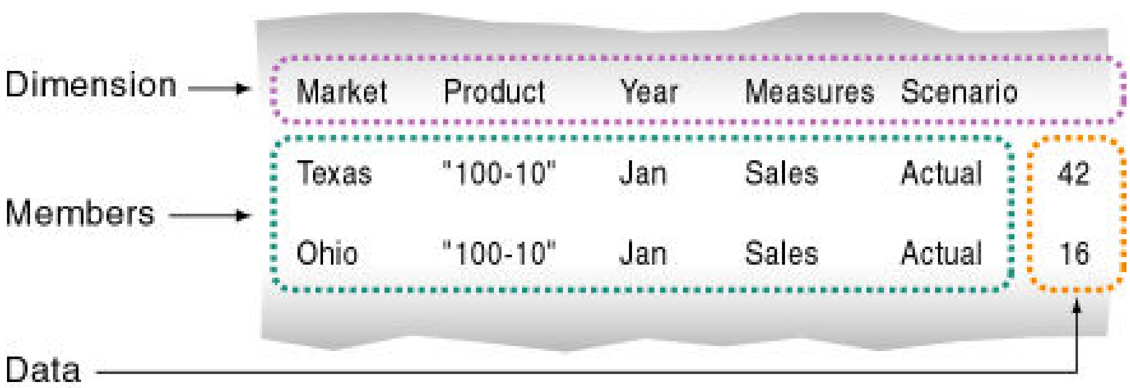
- A record is a structured row of related fields.
- A field is an individual value.
- A delimiter indicates that a field is complete and that the next character in the record starts another field. In the illustration, the default delimiter appears as a space.

Figure 12-1 Records and Fields



Sources of data can contain dimension fields, member fields, member combination fields, and data fields.

Figure 12-2 Kinds of Fields



- Dimension fields identify the dimensions of the cube, such as Market. Use dimension fields to tell Essbase the order of the dimensions in the source. In the illustration above, the dimension fields are Market, Product, Year, Measures, and Scenario. Fields in the Market column, such as Texas, are members of the Market dimension, and fields in the Product column, such as 100-10, are members of the Product dimension. If the dimension fields are not present in the source, you can identify them using the load rule.

- Member fields identify the members or member combinations of the specified dimensions. Use member fields to tell Essbase to which members to map new data values, or which members to add to the outline. In the illustration above, Texas, 100-10, Jan, Sales, and Actual are member fields.
- Data fields contain the numeric data values that are loaded into the intersections of the members of the cube. Each data value must map to a dimension member. In the illustration above, for example, 42 is the data value that corresponds to the intersection of Texas, 100-10, Jan, Sales, and Actual.

  You can specify information in the header and in an individual record. One member from every standard dimension on the outline must be associated with every data value being loaded into Essbase. In the following example, 100 is the data value associated with Jan, Actual, Cola, East, Sales, and 200 is the data value associated with Jan, Actual, Cola, West, Sales.

  ```
  Jan, Actual
  Cola  East  Sales  100
  Cola  West  Sales  200
  Cola  South  Sales  300
  ```

Data fields are used only for data loading; dimension builds ignore data fields.

Valid Dimension Fields

In a data load, if the source does not identify every dimension in the Essbase outline, the load rule must identify the missing dimensions. For example, the Sample.Basic cube has a dimension for Year (where Level 0 of year is Month). If several sources of data arrive with monthly numbers from different regions, the month itself might not be specified in the sources. (Here, all sources contain the same month's data, and we have only one load rule file.) You must specify the month in the header of the load rule.

A dimension field must contain a valid dimension name. If you are not performing a dimension build, the dimension must already exist in the outline. If you are performing a dimension build, the dimension name can be new, but the new name must be specified in the load rule.

Valid Member Fields

A member field can contain the name of a valid member or an alias. In Figure 12-2, for example, Texas and Ohio are valid members of the Market dimension. A blank member field inherits the member name from the previous record. Essbase must know how to map each member field of the source to a member of the outline.

To be valid, a member field must meet the following criteria:

- Contains or inherits a valid member name or member property. See Use the Source Data to Work with Member Properties. If you are not performing a dimension build, the member must already exist in the outline. If you are performing a dimension build, the member can be new.
- Maps to a dimension. Either the source or the rules file must specify which dimension each member field maps to.
- Maps either to a single member name, such as Jan (of the Year dimension) or to a member combination, such as Jan, Actual (of the Year and Scenario dimensions).
- Is within quotation marks as needed. This is necessary for member names that contain the same character as the file delimiter.

When a rules file is not used (for example, if loading data from an Essbase export file), blank dimension and member fields are ok. When Essbase encounters a blank field in such cases, it uses the last dimension or member name encountered for that column.

> **Note:**
>
> As it processes each data load record, Essbase does not check to ensure that a member specified in a member field belongs to the dimension specified for the dimension field. Essbase loads the data value to the data cell identified by the member combination in the record. In Figure 12-2, for example, if the second record reversed Jan and Sales (Texas, '100-10', Sales, Jan, Actual, 42), Essbase would load 42 to the correct data cell.

Valid Data Fields

If you are performing a dimension build, skip this section. Data fields are ignored during a dimension build.

The source data and/or rule file must contain enough information for Essbase to determine where to put the data. A data field contains the data value for its intersection in the cube.

In a data field, Essbase accepts numbers and their modifiers, with no spaces or separators between them, and the text strings #MI and #MISSING, as listed in the table below.

Table 12-1 Valid Data Field Modifiers

Valid Modifiers	Examples
Currency symbols: • Dollar $ • Euro € • Yen ¥	$12 is a valid value. $ 12 is not a valid value because there is a space between the dollar sign and the 12.
Parentheses around numbers to indicate a negative number	(12)
Minus sign before numbers. Minus signs after numbers are not valid.	-12
Decimal point	12.3
Large numbers with or without commas	1,345,218 and 1345218 are valid values.
#MI or #MISSING to represent missing or unknown values	#MI

If the source contains a member field for every dimension and one field that contains data values, you must define the field that contains data values as a data field in the rule file. To read the following source into the Sample.Basic cube, for example, define the last field as a data field.

```
Jan     Cola     East     Sales     Actual     100
Feb     Cola     East     Sales     Actual     200
```

If the source contains blank fields for data values, replace them with #MI or #MISSING. If there is no value in the data field (or the value is #MISSING), Essbase won't replace current values in the cube with empty values.

Valid Delimiters

You must separate fields from each other with delimiters. If you are loading data without a rule file, you must use spaces to delimit fields.

If you are using a rule file, delimiters can be any of the following:

- Tabs (default)
- Spaces
- New lines
- Carriage returns
- Commas

Extra Delimiters Without a Rule File

In sources of data that are loaded without a rule file, Essbase ignores extra delimiters. In the following example, the fields are separated by spaces. Essbase ignores the extra spaces between the fields.

```
East    Cola    Actual    Jan    Sales    10
East    Cola    Actual    Feb    Sales    21
East    Cola    Actual    Mar    Sales    30
```

Extra Delimiters with a Rule File

In sources of data that are loaded with a rule file, Essbase reads extra delimiters as empty fields. For example, if you try to use a rule file to load the file below into the Sample.Basic cube, the load fails. Essbase reads the extra comma between East and Cola in the first record as an extra field. Essbase then puts Cola into Field 3. In the next record, however, Cola is in Field 2. Essbase expects Cola to be in Field 3 and stops the data load.

```
East,,Cola,Actual,Jan,Sales,10
East,Cola,Actual,Feb,Sales,21
East,Cola,Actual,Mar,Sales,30
```

To resolve the problem, delete the extra delimiter from the source.

Valid Formatting Characters

Essbase views some characters in the source data as formatting characters only. Essbase ignores the characters listed below:

Table 12-2 Valid Formatting Characters

Formatting Character	Description
`==`	Multiple equal signs, such as for double underlining
`--`	Multiple minus signs, such as for single underlining

Table 12-2 (Cont.) Valid Formatting Characters

Formatting Character	Description
_ _	Multiple underscores
==	Multiple IBM PC graphic double underlines (ASCII character 205)
_ _	Multiple IBM PC graphic single underlines (ASCII character 196)

Ignored fields do not affect the data load or dimension build.

For example, Essbase ignores the equal signs in the following source file, and loads the other fields normally.

```
East  Actual  "100-10"
      Sales   Marketing
      =====   =========
Jan   10      8
Feb   21      16
```

Supported Source Data Types

Sources of data you can use for Essbase data load and dimension build include text files (flat files), relational tables or databases such as Oracle Database, exported files from Essbase cubes, and other Essbase cubes.

Essbase supports many types of source data, including:

- Text files (flat files) from text backups or external sources
- Relational data sources
- Essbase export files (export files do not need a rules file to load)
- Other sources. See Use Connections and Datasources to learn more about setting up connectivity to source data.

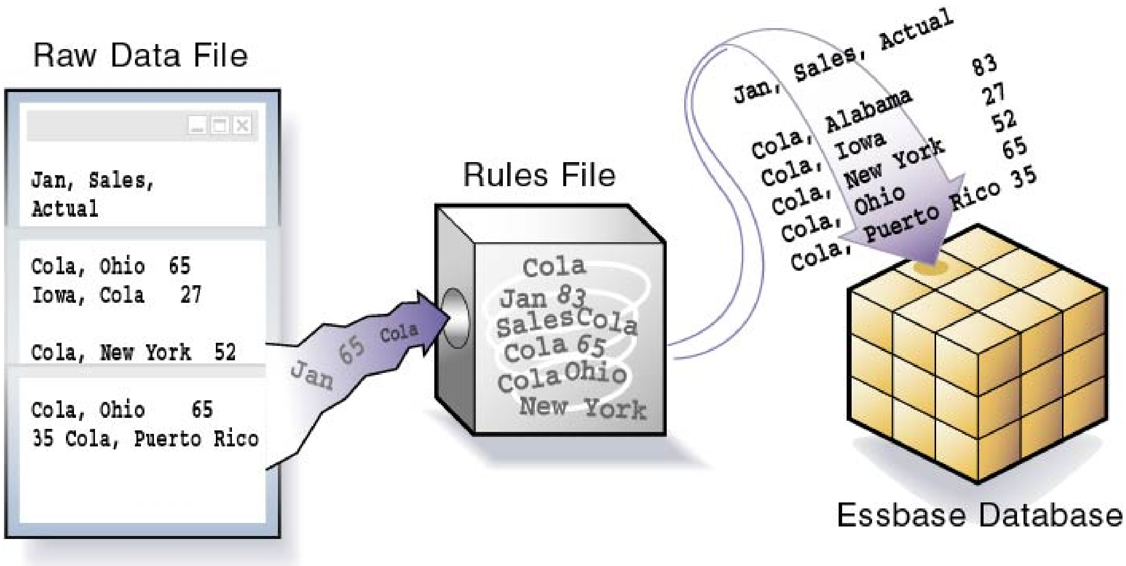
You can import or build cubes through Cube Designer using specially formatted, Microsoft Excel-based application workbook files.

You cannot use regular Excel files as source data for loading to a cube.

Load Rules

Load rules help you define operations that Essbase should perform to process a source of data during a data load or dimension build. Use rules to map data values to the cube, or to map dimensions and members to the outline.

Figure 12-3 Loading Data Through Load Rules



A load rule defines which build method to use, whether data values or members are sorted or in random order, and how to transform data values or members before loading them.

Essbase reads the data values or members in the source, changes them based on the rules, and loads the changed data values into the cube and the changed members into the outline. Essbase does not change the source data. You can reuse a rule file with any source of data that requires the same set of rules.

Load Rule Operations

Rule files enable you to make the following adjustments as you load data or members to your cube:

- Pull from the external source using a SQL query
- Add dimensions and members to the outline
- Change existing dimensions and members in the outline
- Ignore some fields or strings in the source data
- Change the order of fields by moving, joining, splitting, or creating fields
- Map the data in the source to the cube by changing strings
- Change the data values in the source by scaling data values or by adding data values to existing data values in the data source
- Set header records for missing values

- Reject an invalid record and continue the data load

Data Load/Dimension Build Success Criteria

When building dimensions, it is best to create a separate rule file for each dimension.

To load a data value successfully, Essbase must know its dimensionality; in other words, Essbase must encounter one member from each dimension before encountering the data value. For example, in Figure 12-2, Essbase loads the data value 42 into the database with the members Texas, 100-10, Jan, Sales, and Actual. If Essbase encounters a data value before a member of each dimension is specified, it stops loading the data.

The simplest way to format a record (which is analogous to a row in the source data) is to include a member from each dimension, followed by a data field, as illustrated below:

```
Sales "100-10" Ohio Jan Actual 25
Sales "100-20" Ohio Jan Actual 25
Sales "100-30" Ohio Jan Actual 25
```

The number of fields in each record in a rule file must match (note how each record above has 6 fields).

A source of data must contain all of the following and nothing else.

- One or more valid members from each dimension. A member name must be enclosed in quotation marks if it contains any of the following:
  - Spaces
  - Numeric characters (0–9)
  - Dashes (minus signs, hyphens)
  - Plus signs
  - Ampersands (&)
- One or more valid data values. See Valid Data Fields.
- Valid delimiters. See Valid Delimiters.

Use the load rule to help you format the source data as you load it (there is no need to edit the source manually). If the source data is already perfectly formed (for example, if it is an unaltered data export from Essbase), then you may not need a load rule, but such situations are rare. Dimension builds always require a load rule.

Situations that Do and Do Not Need a Load Rule

To load data to Essbase, you typically need a load rule, unless the source data maps perfectly to the cube (for example, if it has been exported from the cube).

You need a load rule if the source data does not map perfectly to the cube or if you are performing any of the following tasks:

- Loading data from an external source
- Building dimensions
  - Adding dimensions and members to the cube
  - Changing existing dimensions and members in the cube
- Changing the data in any way, including the following:

- Ignoring fields or strings in the source data
- Changing the order of fields by moving, joining, splitting, or creating fields
- Mapping the data in the source data to the cube by changing strings
- Changing the data values in the data source by scaling data values or by adding data values to existing data values in the source
- Setting header records for missing values
- Rejecting an invalid record and continuing the data load

You do not need a load rule if you are performing a data load and the data source maps perfectly to the database, as described in Sources of Data that Do Not Need a Load Rule.

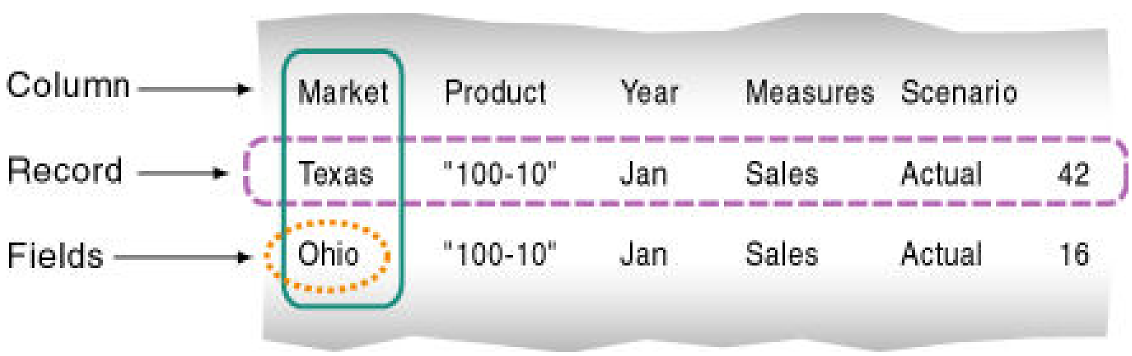
> **Note:**
>
> If you are using a load rule, the number of fields in each record in the rule must match.

Sources of Data that Do Not Need a Load Rule

To load data to an Essbase cube, you typically need a load rule, unless the source data maps perfectly to the cube. However, if a source of data contains all of the information required, you can load the data source directly in a free-form data load with no rule.

If you are performing a dimension build, skip this section. Dimension builds always require load rules.

To load a data value successfully, Essbase must encounter one member from each dimension before encountering the data value. For example, Essbase loads the data value 42 into the database with the members Texas, 100-10, Jan, Sales, and Actual.



If Essbase encounters a data value before a member of each dimension is specified, it stops loading the data.

To map perfectly, a source of data must contain all of the following and nothing else:

- One or more valid members from each dimension. A member name must be enclosed in quotation marks if it contains any of the following:
  - Spaces
  - Numeric characters (0–9)

  - Dashes (minus signs, hyphens)
  - Plus signs
  - Ampersands (&)

    If you are performing a data load without a rule file, when Essbase encounters an invalid member field, it stops the data load even if the Abort on Error flag is not set to true. Essbase loads all fields read before the invalid field into the database, resulting in a partial load of the data values.

- One or more valid data values. See Valid Data Fields.

  If the data source contains blank fields for data values, replace the blank fields with #MI or #MISSING. Otherwise, the data values may not load correctly.

- Valid delimiters. See Valid Delimiters.

The fields in the data source must be formatted in an order that Essbase understands. The simplest way to format a record is to include a member from each dimension and a data field, as illustrated below:

```
Sales "100-10" Ohio Jan Actual 25
Sales "100-20" Ohio Jan Actual 25
Sales "100-30" Ohio Jan Actual 25
```

An incorrectly formatted data source will not load. You can edit the data source using a text editor and fix the problem. If you must perform many edits (such as moving several fields and records), use a load rule.

The following sections describe more complicated ways to format free-form sources.

Ranges of Member Fields in the Source Data

You can express Essbase member names as ranges within a dimension. A range is a series of values. A source of data may contain ranges from multiple dimensions at a time. Learn how Essbase processes ranges during a data load, if no load rule is used.

If you are performing a dimension build, this information is not applicable, as dimension builds always require a rule. If you are performing a data load, this information applies when no rule is used (but it is also helpful to know about ranges when you design a data load rule).

You can express member names as ranges within a dimension. For example, Sales and COGS form a range in the Measures dimension. Ranges of member names can handle a series of values.

A source of data may contain ranges from multiple dimensions at a time. In the example below, Jan and Feb form a range in the Year dimension and Sales and COGS form a range in the Measures dimension.

```
Actual    Texas    Sales    COGS
            Jan    Feb    Jan    Feb
"100-10"    98     89     26     19
"100-20"    87     78     23     32
```

Notice that Sales is defined for the first two columns and COGS for the last two columns.

Automatic Ranges

When Essbase encounters multiple members from the same dimension with no intervening data fields, it sets up a range for that dimension. The range stays in effect until Essbase encounters another member name from the same dimension, at which point Essbase replaces the range with the new member or new member range.

The following example contains a range of Jan to Feb in the Year dimension. It remains in effect until Essbase encounters another member name, such as Mar. When Essbase encounters Mar, the range changes to Jan, Feb, Mar.

```
Texas    Sales
                     Jan    Feb    Mar
Actual    "100-10"   98     89     58
          "100-20"   87     78     115
```

Out of Range Data Values

When Essbase encounters a member range, it assumes that there is a corresponding range of data values. If the data values are not in the member range, the data load stops. Essbase loads any data fields read before the invalid field into the cube, resulting in a partial data load.

The following example contains more data fields than member fields in the defined range of members. The data load stops when it reaches the 10 data field. Essbase loads the 100 and 120 data fields into the cube.

```
Cola   Actual   East
        Jan    Feb
Sales   100    120    10
COGS    30     34     32
```

Repeated Members in a Range

Structure ranges in the source data so that Essbase interprets them correctly. If a member appears more than once in a range, Essbase ignores the repeated members.

The first table below shows repeated members for Actual, Budget, Sales, and COGS and two ranges: Actual to Budget and Sales to COGS. Essbase ignores the repeated instances of Actual, Budget, Sales, and COGS (for example, in the second line, the repeated Actual and Budget are ignored):

```
Cola   East
      Actual   Budget   Actual   Budget
      Sales   Sales    COGS    COGS
Jan   108     110      49      50
Feb   102     120      57      60
```

For Actual, the first member of the first range, Essbase maps data values to each member of the second range (Sales and COGS). Essbase then proceeds to the next value of the first range, Budget, similarly mapping values to each member of the second range. As a result, Essbase interprets the file as shown below:

```
Cola   East
      Actual         Budget
```

```
      Sales    COGS    Sales    COGS
Jan   108      110     49       50
Feb   102      120     57       60
```

Columns in the Source Data

Learn how Essbase processes columns during a data load, if no load rule is used. Columns of source data you load to the cube may be symmetric or asymmetric.

If you are performing a dimension build, this information is not applicable, as dimension builds always require a rule. If you are performing a data load, this information applies when no rule is used (but it is also helpful to know about columns when you design a data load rule).

Sources of data typically contain columns of fields. Essbase supports loading data from symmetric or asymmetric columns.

Symmetric columns

Symmetric columns have the same number of members under them. In the following example, each dimension column has one column of members under it. For example, Product has one column under it (100-10 and 100-10) and Market has one column under it (Texas and Ohio).

```
Product     Measures    Market    Year   Scenario
"100-10"    Sales       Texas    Jan    Actual     112
"100-10"    Sales       Ohio     Jan    Actual     145
```

The columns in the following file are also symmetric, because Jan and Feb have the same number of members under them:

```
                                  Jan                Feb
                           Actual   Budget   Actual   Budget
"100-10"    Sales   Texas  112      110      243      215
"100-10"    Sales   Ohio   145      120      81       102
```

Asymmetric columns

Asymmetric columns have different numbers of members under them. In the following example, the Jan and Feb columns are asymmetric because Jan has two columns under it (Actual and Budget) and Feb has one column under it (Budget):

```
                           Jan      Jan      Feb
                           Actual   Budget   Budget
"100-10"    Sales   Texas  112      110      243
"100-10"    Sales   Ohio   145      120      81
```

If a file contains asymmetric columns, label each column with the appropriate member name.

The example above is valid because the Jan label is now over Actual and Budget. It is clear to Essbase that both columns map to Jan.

The following example is not valid because the column labels are incomplete. The Jan label must appear over the Actual and Budget columns.

```
                             Jan                Feb
                             Actual    Budget   Budget
"100-10"    Sales    Texas   112       110      243
"100-10"    Sales    Ohio    145       120      81
```

Security and Multiple-User Considerations

Because Essbase supports concurrent users reading and updating the cube, users may be active when you are attempting to perform dimension builds, data loads, and calculations.

- Security Issues

  The security system prevents unauthorized users from changing the cube. Only users with write access can load data values or add dimensions and members. Write access can be provided globally or by using filters.

- Multi-User Data Load Issues

  You can load data values while multiple users are connected. Essbase uses a block locking scheme for handling multi-user issues. When you load data values, Essbase does the following:

  – Locks the block it is loading into so that no one can write to the block.

  – Updates the block.

- Multi-User Dimension Build Issues

  You cannot build dimensions while other users are reading or writing to the cube. After you build dimensions, Essbase restructures the outline and locks the cube for the duration of the restructure operation.

13 Design Rules for Data Load or Dimension Build

Design load rules to control how data and metadata are loaded into an Essbase cube or outline. Whether the data comes from a file or an external source, rules help you shape and modify records you load into Essbase.

Data load rules contain sets of operations to shape data from an external source as you load it into Essbase. Similarly, dimension build rules create or modify the dimensions and members in an outline dynamically from the external source.

There are many ways to build Essbase dimensions and load data. You initiate these operations while building cubes from application workbooks, by running jobs, or by streaming. Whichever method you use, rules are essential for shaping how Essbase should filter, map, and transform the external data and metadata that flows into your cube.

When you load dimensions and data into Essbase, the external source data is always left unchanged.

A separate rule is required for each unique, non-Essbase source of data.

If you're only re-importing data from native Essbase export files, you do not need to use a rule.

- General Process for Creating Rules
- Generate Load Rules with Application Workbooks
- Define Load Rules in Essbase Web Interface
- Define Rules that Query External Sources
- Export Load Rules for Offline Editing

General Process for Creating Rules

To create data load or dimension build rules that define how data and metadata are loaded into Essbase, start by providing Essbase information about the source data, such as where it is located and how it is shaped. Specify delimiters, build method, and source property information.

Dimension Build Rules Workflow

To create a dimension build rule, use the following workflow:

1. In the **Scripts** tab of the cube inspector, initiate the process to create a new rule.

   See Create New Dimension Build Rule.
2. If a file is the source of data, set the file delimiters for the source data.

   See Set File Delimiters.
3. If an external database is the source of data, set up connectivity to the source.

   See Define Rules that Query External Sources.

4. If you are creating a new dimension, name/add it as you define the rule.

   See Create New Dimension Build Rule.

5. Select the build method.

   See Select a Build Method.

6. If necessary, change or set the properties of members and dimensions you are building.

   See Set Dimension Build Operational Instructions.

7. If necessary, perform record, field, or data to make changes during the load operation.

8. Set field type information, including field type, field number, and dimension.

   See Set Dimension Build Field Type Information.

9. Validate and save the rules file.

   See Validate Rules.

Data Load Rules Workflow

1. In the **Scripts** tab of the cube inspector, initiate the process to create a new rule.

   See Create New Data Load Rule.

2. Set the file delimiters for the source data.

   See Set File Delimiters.

3. Map each rule field to the source data, and define field properties.

   See Define Data Load Field Properties.

4. If necessary, set record, field, and data operations to change the data in the source data during loading.

5. Validate and save the rule.

Generate Load Rules with Application Workbooks

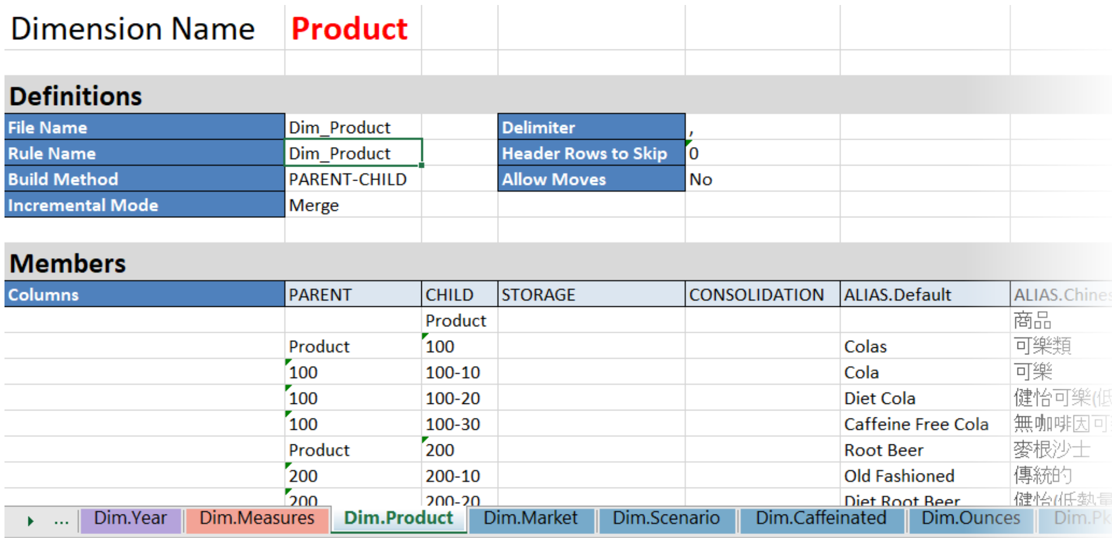
If you build a cube from an application workbook, Essbase creates dimension build and data load rules for you. The editable dimension and data worksheets contain the properties that define how dimensions and data should be loaded.

When you deploy a cube from an application workbook, the dimension and data worksheets each build and execute one load rule.

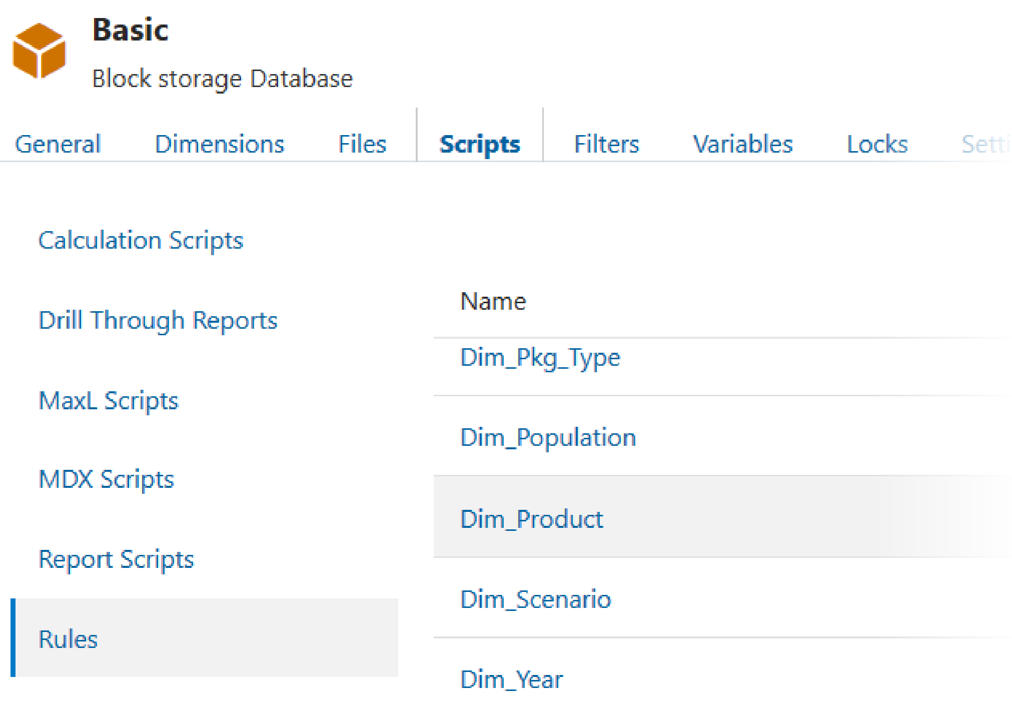
Dimension Worksheets and Rules

The worksheet pictured below defines the Product dimension of the Sample Basic cube. The Definitions section shows the rule name, build method, delimiter used, and instructions for handling header rows. It also specifies the Incremental Mode as Merge, which means that a dimension build run using this rule will add new members while retaining existing members (this option is the same as Update Option in the dimension properties). Moves are not allowed.

This application workbook, which you can use to easily build the Sample Basic cube, is available in the gallery section of the Essbase server file catalog.



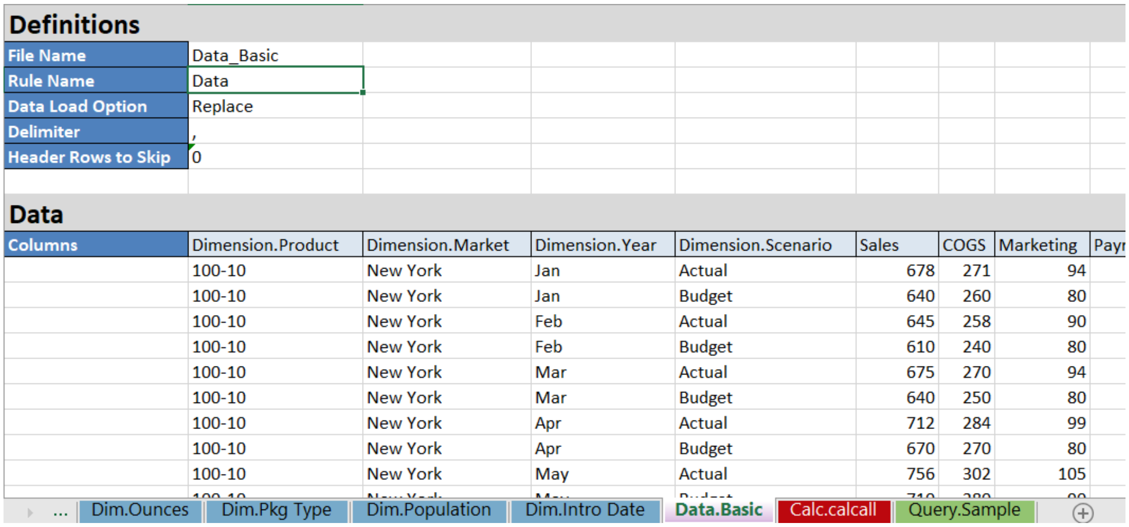
When you build the cube, a dimension build rule for each dimension worksheet is added to the cube directory, and you can edit these in the Essbase web interface if you need to make changes.



Data Worksheets and Rules

When you build a cube from an application workbook, the data worksheets also build and run one data load rule for each data worksheet.

The data worksheet pictured below defines the data load rule for Sample Basic. The Definitions section shows the rule name, data load option, delimiter used, and instructions for handling header rows.



Definitions

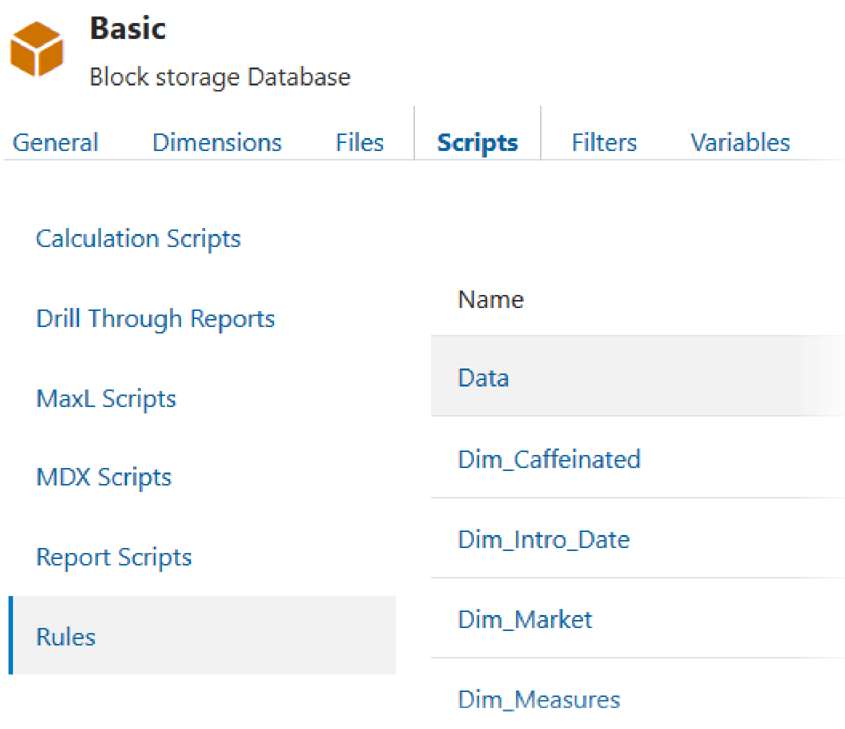
File Name	Data_Basic
Rule Name	Data
Data Load Option	Replace
Delimiter	,
Header Rows to Skip	0

Data

Columns	Dimension.Product	Dimension.Market	Dimension.Year	Dimension.Scenario	Sales	COGS	Marketing	Payr
	100-10	New York	Jan	Actual	678	271	94	
	100-10	New York	Jan	Budget	640	260	80	
	100-10	New York	Feb	Actual	645	258	90	
	100-10	New York	Feb	Budget	610	240	80	
	100-10	New York	Mar	Actual	675	270	94	
	100-10	New York	Mar	Budget	640	250	80	
	100-10	New York	Apr	Actual	712	284	99	
	100-10	New York	Apr	Budget	670	270	80	
	100-10	New York	May	Actual	756	302	105	

... Dim.Ounces | Dim.Pkg Type | Dim.Population | Dim.Intro Date | Data.Basic | Calc.calcall | Query.Sample

The data load rule is added to the cube directory, and you can edit it in the Essbase web interface if you need to make changes.



See Also

- Understand Dimension Worksheets
- Understand Data Worksheets

Define Load Rules in Essbase Web Interface

The administrative Essbase web interface includes an advanced load rules editor with built-in data previews, and a centralized Jobs interface from which to run dimension build and data load jobs using the rules you defined.

Learn how to use the rule editor to create dimension build and data load rules, including how to set delimiters, work with fields and records in your data, find a build method that works best with your data, and use properties to define how your data and dimensions are loaded.

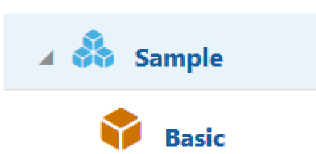
- Access and Edit Rules
- Create New Dimension Build Rule
- Create New Data Load Rule
- Set File Delimiters
- Set Dimension Build Field Type Information
- Select a Build Method
- Set Dimension Build Operational Instructions
- Set Dimension Build Global Properties and General Source Properties
- Use the Source Data to Work with Member Properties
- Define Data Load Field Properties
- Perform Operations on Records
- Perform Operations on Fields
- Perform Operations on Data
- Validate Rules

Access and Edit Rules

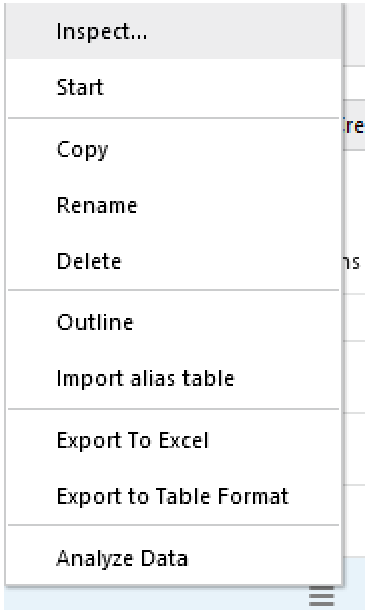
Use the rule editor in the Essbase web interface to access and edit a dimension build rule.

Prerequisite: Import a cube from the gallery. The following instructions assume you have built or imported Sample Basic.

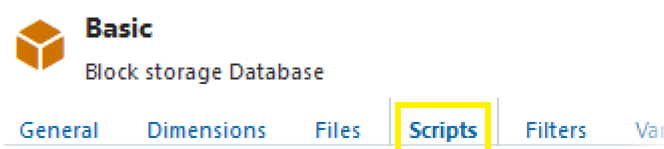
1. On the **Applications** page, expand the application (Sample).



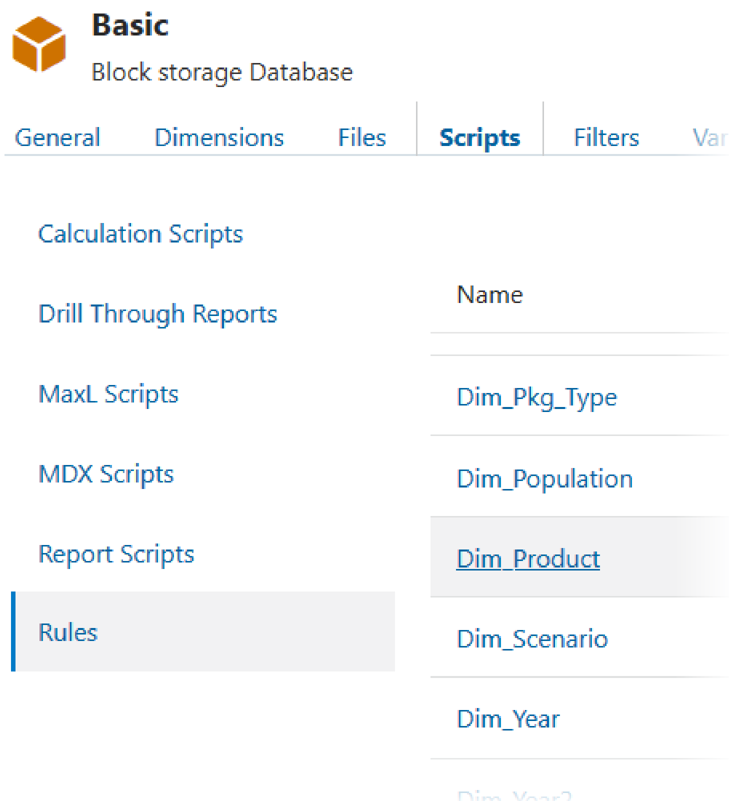
2. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



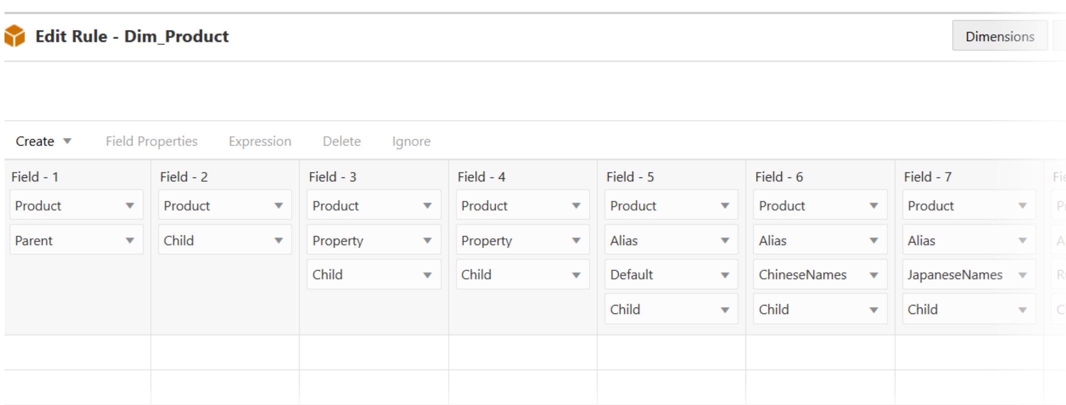
3. Select the **Scripts** tab, and then click **Rules**.



4. Click **Dim_Product**, which is the name of a dimension build rule associated with Sample Basic.



The Dim_Product rule opens in the rule editor.



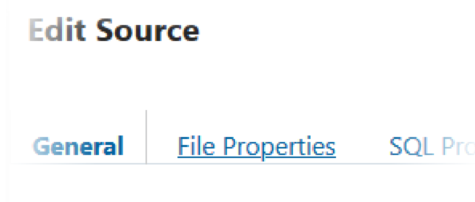
Fields are defined with metadata that identifies the expected format of the source data file. These definitions are already present, because this rule was created automatically by the application workbook when you imported the cube.

All fields are associated with the Product dimension. This rule uses the parent-child build method. Field 1 is designed to expect a parent member. Field 2 expects a child of Field 1. Fields 3 and 4 are reserved for identifying member properties of the child member in Field 2. Fields 5-9 are for aliases associated with the child.

No preview data is visible until you associate this rule with a data file. That is why there are empty rows (records) beneath the field definitions.

5. Now associate a source data file with the rule, to preview it and ensure that the field definitions map correctly to the records in the source file. Click **Source Properties**.

6. Click **File Properties**.



7. For **File**, navigate to the source data file for building the Product dimension. In the Catalog, it is `Dim_Product.txt`, located in the current cube directory for Sample Basic.
8. Select `Dim_Product.txt` from the list, and click **Select**.

   The file has no header record, and is comma-delimited, so you can leave all the defaults as-is, and click **Preview Data**.

   > **Note:**
   >
   > You can increase the number of records that will display in the preview by incrementing **Preview Data Count** on the File Properties tab. The maximum preview size is 500 records.

   The Dim_Product rule is populated with preview data mapped to the fields.

Field - 1	Field - 2	Field - 3	Field - 4	Field - 5	Field - 6
Product	Product	Product	Product	Product	Product
Parent	Child	Property	Property	Alias	Alias
		Child	Child	Default	ChineseNa
				Child	Child
	Product				商品
Product	100			Colas	可樂類
100	100-10			Cola	可樂
100	100-20			Diet Cola	健怡可樂(低
100	100-30			Caffeine Free Cola	無咖啡因可
Product	200			Root Beer	麥根沙士
200	200-10			Old Fashioned	傳統的

Create New Dimension Build Rule

Create and test a dimension build rule using the rule editor in the Essbase web interface. Build dimensions in an Essbase cube using data from a flat file or from a variety of external sources. This example uses a flat file.

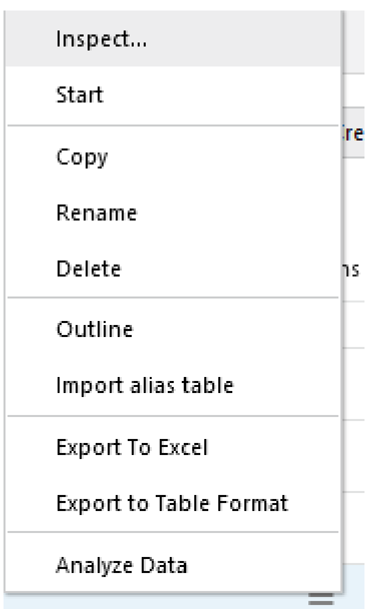
It is very efficient to modify existing rules, such as those that are created for you when you deploy from application workbooks. However, if you are creating a new rule in the Essbase web interface, this topic illustrates a sample flow for creating it, with the goal of adding and editing dimension members in an Essbase cube.

Prerequisites:

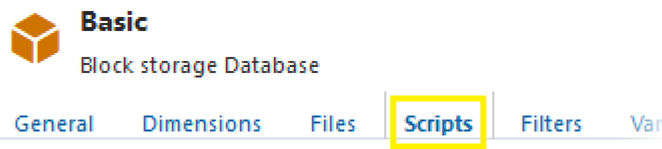
- Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase server.
- Create a tab-delimited data file like the following, name it `add_product.txt`, and upload it to the Sample Basic cube directory.

```
500     500-10
500     500-10
500     500-20
500     500-20
500     500-20
```

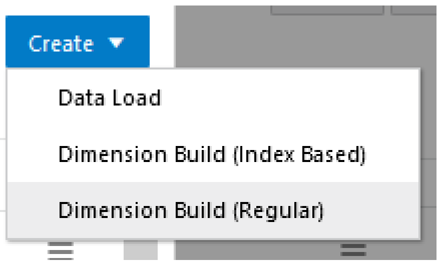
1. On the **Applications** page, expand the application (Sample).
2. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



3. Select the **Scripts** tab, and then click **Rules**.

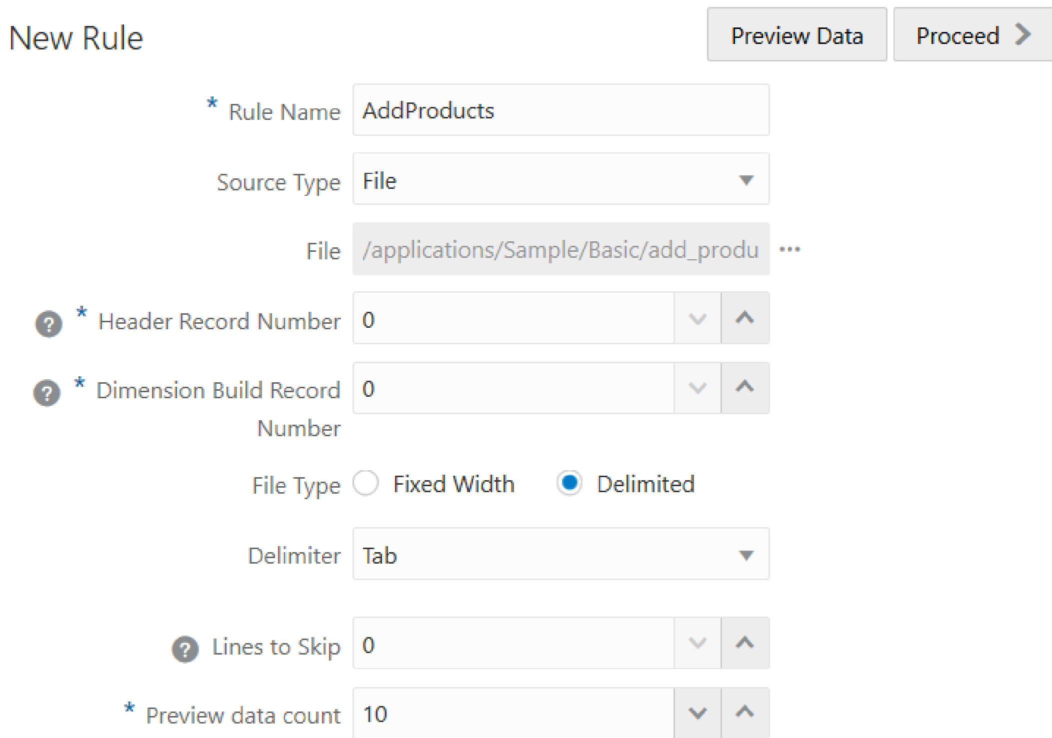


4. Click **Create** and choose **Dimension Build (Regular)**.



> **Note:**
>
> Regular dimension build rules have validation requirements pertaining to the order of fields. For example, the properties of generation 1 must appear after the Generation 1 column. Other generations must appear in order of Parent/Child. If fields are out of order, the Verify action will display an error message.
>
> For index-based rules (sometimes called BPM rules), fields can be in any order; for example, Generation 2 can come before Generation 1, and a property of Generation 1 can come before the actual Generation 1 column. An index-based rule does not offer column operations (join, split, and move).
>
> You can change the type of dimension build rule by modifying the end of the URL in the Essbase web interface when the rule is open for editing. Regular type rules have a URL ending in `&type=regular`, and index-based rules have a URL ending in `&type=index`.
>
> Data load rules do not have different types. Similar validation rules exist as for the Regular dimension build rules.

5. In the New Rule dialog,

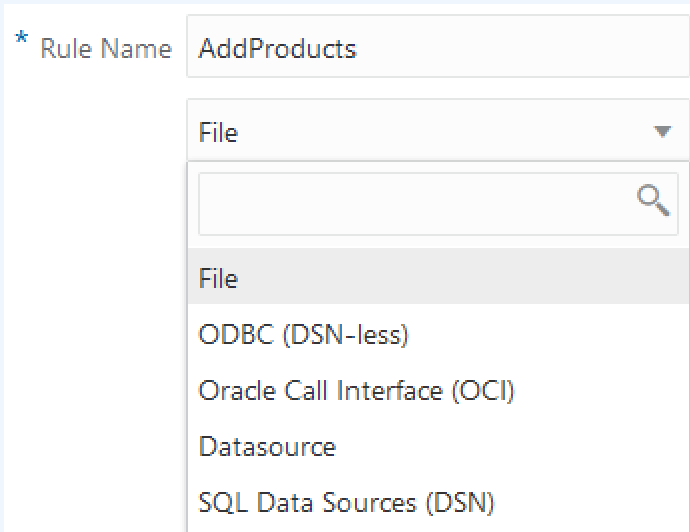


a. Enter a rule name.

b. For **Source Type** select **File**, as this example workflow is based on the assumption of using a flat data file. Select the data file you uploaded. For example, click **Catalog** and navigate to `add_product.txt`.

Note:

Additional options besides **File** are available to use as the **Source Type**. Use these when your dimension build source data is an external source rather than a flat file.



When you are using an external source of data to build dimensions, you must complete certain prerequisites, depending on which type of connectivity you have to the source of data.

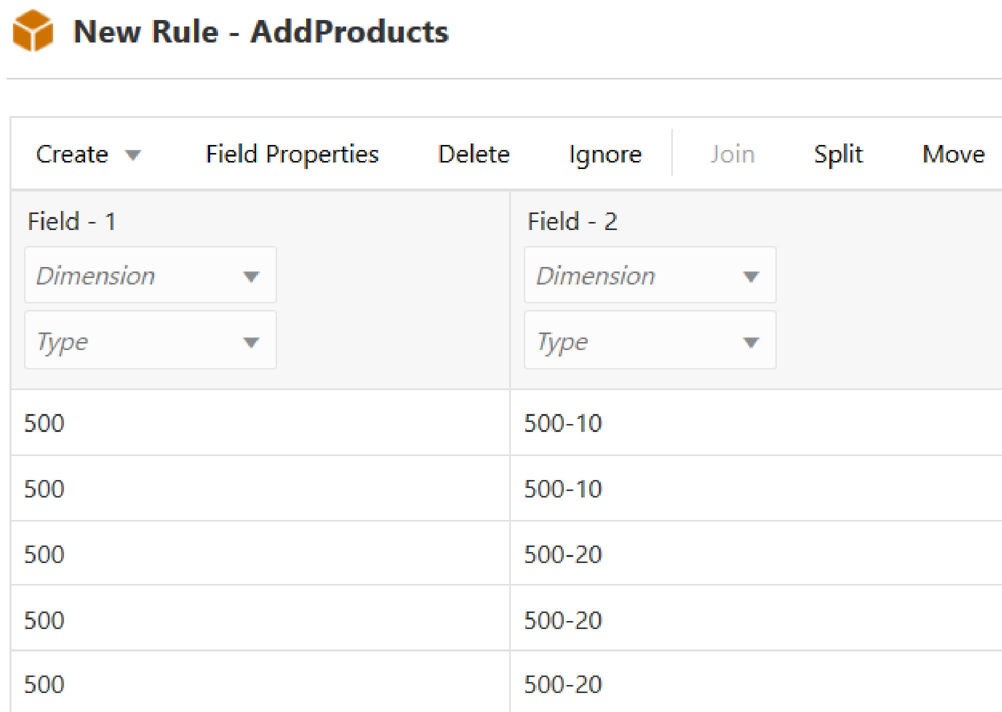
- If your connectivity has been predefined using a connection and Datasource established in Essbase by an application manager or service administrator, choose **Datasource** as the Source Type for the dimension build rule. To ensure your connectivity is ready, see Access External Data Using a Connection and Datasource to understand the prerequisite steps.
- If your connectivity depends on ODBC drivers configured on the Essbase server, choose **ODBC (DSN-less)** or **SQL Data Sources (DSN)** as the Source Type for the dimension build rule. To ensure your connectivity is ready, see Access Data Using ODBC Connectivity to understand the prerequisite steps.
- If your connectivity depends on a connection string that includes `OCI` syntax, choose **Oracle Call Interface (OCI)** as the Source Type for the dimension build rule. To ensure your connectivity is ready, see Access Oracle Database Using Oracle Call Interface to understand the prerequisite steps.

c. Leave the Record Number fields blank.

d. As the sample data file is tab delimited, change the Delimiter value to Tab.

e. Click **Proceed**.

f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.

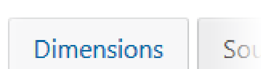


Each row in the source data is a *record*, and corresponds with the rows you see in the rule preview data. The data is top-down, as you can tell by the parent-child order that begins each record (for example, 500 is a general product category, and 500-10 gets more specific about the product type).

6. In a dimension build rule, all fields must map to a dimension. Typically, one dimension is used for all fields in a dimension build rule. You already have dimensions to associate because you are building on Sample Basic. However, if you create a new cube, there are no dimensions, and you need to create and map them to your rule.

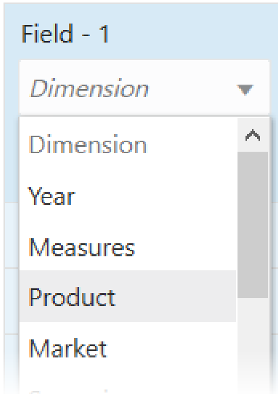
   Skip to the next step unless you are working with a new cube with no dimensions, instead of with Sample Basic. Or, use this step if you want to add a new dimension.

   a. To create and map a dimension, click **Dimensions**.

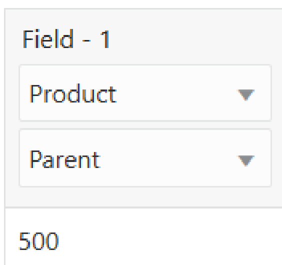
   b. Type a dimension name: **Products**. Click **Add**.

7. Click *Dimension* in Field 1, and select **Product** to associate the dimension with Field 1.



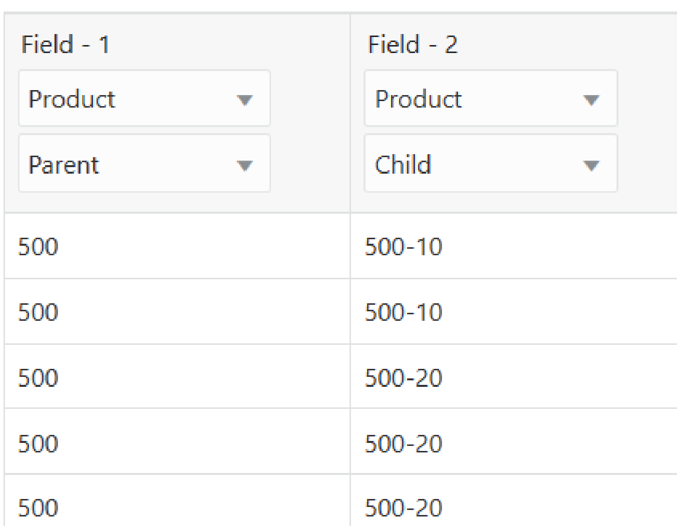
Product populates the Dimension selector for all fields in the rule. This is acceptable, as in this example exercise, you are designing the rule to build upon just one dimension in the Essbase cube.

8. Select field types for each field.
   a. For Field 1, select Parent as the type.



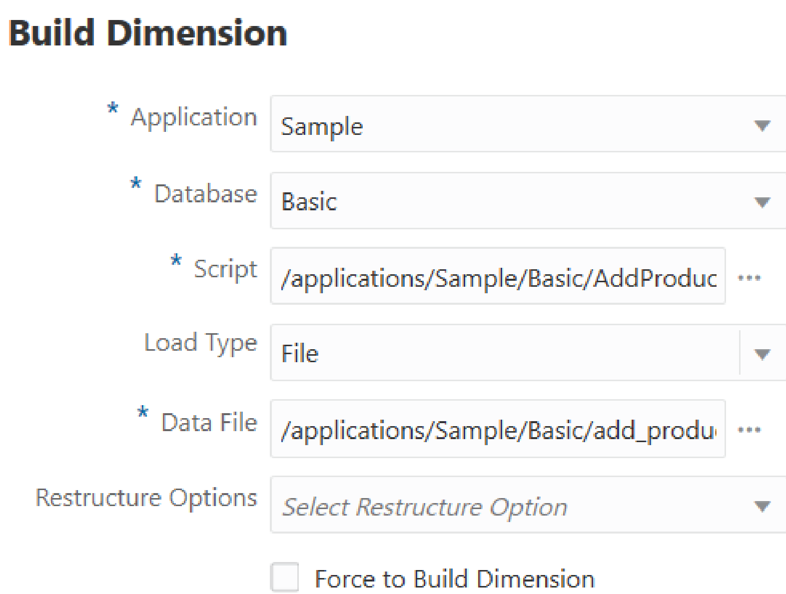
   b. For Field 2, select Child as the type.

   The rule should look like the following now:



9. Verify the rule, then save and close.
10. Close the cube inspector, and click **Jobs** to test your dimension build rule.

11. Click **New Job** and **Build Dimension**.
    - For Application, select Sample.
    - For Database, select Basic.
    - For Script, navigate to the rule file you created in the Sample Basic directory.
    - For Load Type, specify File.
    - For Data File, navigate to the text file you uploaded in the prerequisite step.



12. Click the refresh icon until the job status is Completed.
13. On the **Applications** page, expand the application (Sample).
14. From the **Actions** menu, to the right of the cube name (Basic), click **Outline**.
15. Expand the Products dimension and note that the new product, 500, was added with its children.

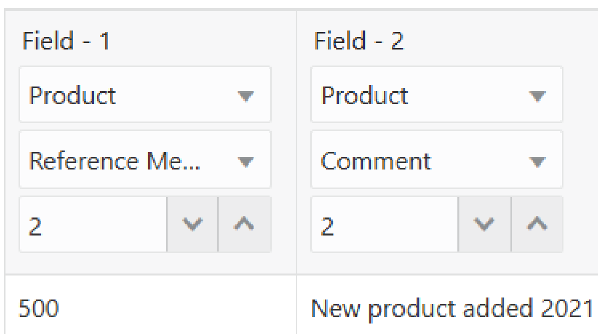
Bonus Exercise: Add a Comment

In the preceding steps, you used a regular dimension build rule to add a product member to the Sample Basic cube. Now, create an index-based rule to modify a property of the new member.

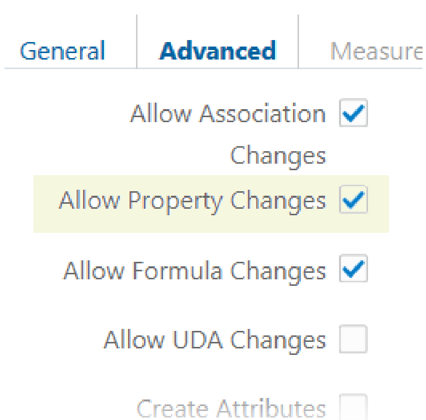
1. Upload a file named `addcomment.txt` to the Sample Basic directory, with the following contents:

   ```
   500,"New product added 2021"
   ```

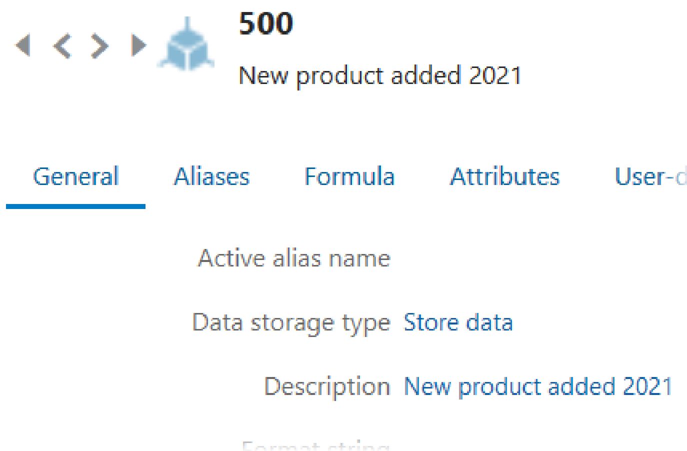
2. Create an Index-based rule named AddComment, and preview it with the text file you uploaded.
3. For Field 1, set the *Dimension* to Product, set the *Type* to **Reference Member**, and increment *Generation* to 2.
4. For Field 2, set the *Type* to **Comment**, and increment *Generation* to 2.



5. Click **Dimensions**, click Product, and click **Advanced**.
6. Check **Allow Property Changes** and click OK.



7. Verify the rule, then save and close.
8. Run the dimension build job.
9. View the outline, and inspect the properties of Product member 500 to confirm that the comment was added.



Create New Data Load Rule

Create and test a data load rule using the rule editor in the Essbase web interface. Load data into an Essbase cube using data from a flat file or from a variety of external sources. This example uses a flat file.

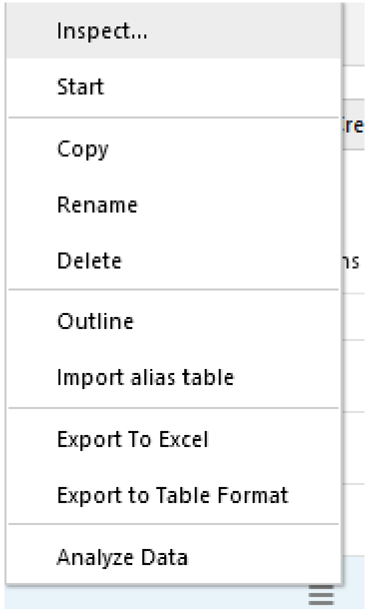
It is very efficient to modify existing rules, such as those that are created for you when you deploy from application workbooks. However, if you are creating a new rule in the Essbase web interface, this topic illustrates a sample flow for creating it, with the goal of loading data for new dimension members added to an Essbase cube.

Prerequisites:

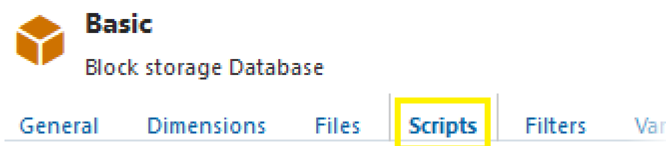
- Complete the steps in the topic Create New Dimension Build Rule. This exercise assumes you have already built the Sample Basic cube and have added new "500" members to the Product dimension.
- Create a comma-delimited data file like the following, name it `load_sales_cogs_newprod.txt`, and upload it to the Sample Basic cube directory

```
"Product","Market","Year","Scenario","Sales","COGS"
"500-10","New York","Jan","Actual","678","271"
"500-10","New York","Jan","Budget","640","260"
"500-10","New York","Feb","Actual","645","258"
"500-10","New York","Feb","Budget","610","240"
"500-10","New York","Mar","Actual","675","270"
"500-10","New York","Mar","Budget","640","250"
"500-10","New York","Apr","Budget","670","270"
"500-10","New York","May","Actual","756","302"
"500-10","New York","May","Budget","710","280"
"500-10","New York","Jun","Actual","890","356"
"500-10","New York","Jun","Budget","840","340"
```

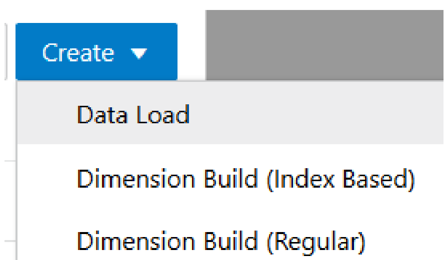
1. On the **Applications** page, expand the application (Sample).
2. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



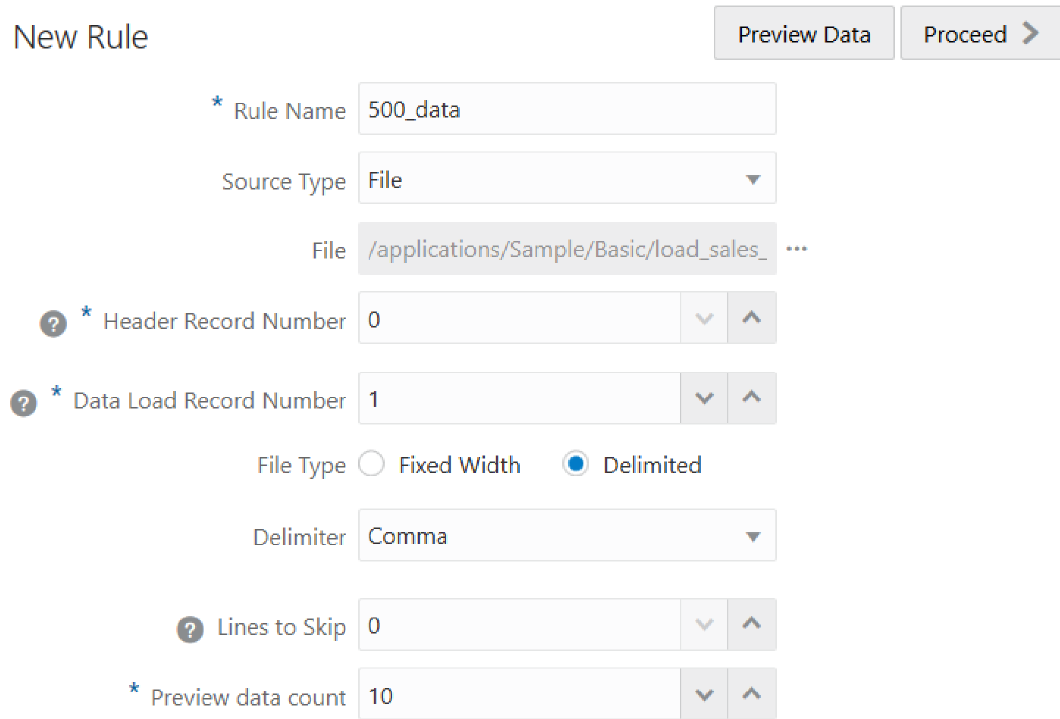
3. Select the **Scripts** tab, and then click **Rules**.



4. Click **Create** and choose **Data Load**.

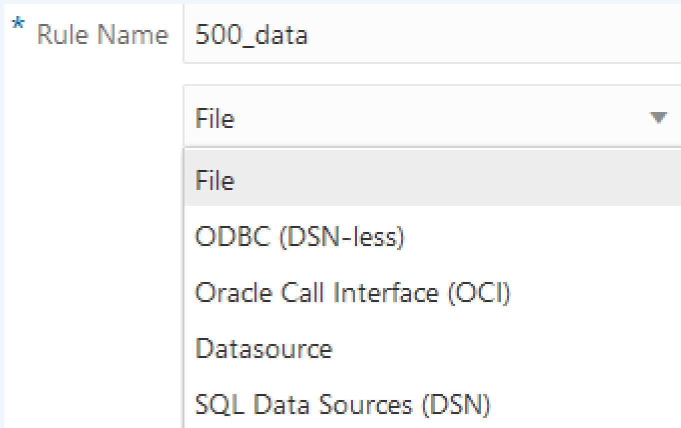


5. In the New Rule dialog,



a. Enter a rule name: 500_data.

b. For **Source Type** select **File**, as this example workflow is based on the assumption of using a flat data file. Select the data file you uploaded. For example, click **Catalog** and navigate to `load_sales_cogs_newprod.txt`.

> **Note:**
>
> Additional options besides **File** are available to use as the **Source Type**. Use these when your source data is an external source rather than a flat file.
>
> 
>
>
> When you are using an external source of data, you must complete certain prerequisites, depending on which type of connectivity you have to the source of data.
>
> - If your connectivity has been predefined using a connection and Datasource established in Essbase by an application manager or service administrator, choose **Datasource** as the Source Type for the load rule. To ensure your connectivity is ready, see Access External Data Using a Connection and Datasource to understand the prerequisite steps.
> - If your connectivity depends on ODBC drivers configured on the Essbase server, choose **ODBC (DSN-less)** or **SQL Data Sources (DSN)** as the Source Type for the load rule. To ensure your connectivity is ready, see Access Data Using ODBC Connectivity to understand the prerequisite steps.
> - If your connectivity depends on a connection string that includes `OCI` syntax, choose **Oracle Call Interface (OCI)** as the Source Type for the load rule. To ensure your connectivity is ready, see Access Oracle Database Using Oracle Call Interface to understand the prerequisite steps.

c. Increment the Data Load Record Number field to 1, because record 0 contains header information you won't load (though it will be used to populate the load rule fields).

d. As the sample data file is comma delimited, leave the Delimiter value as Comma.

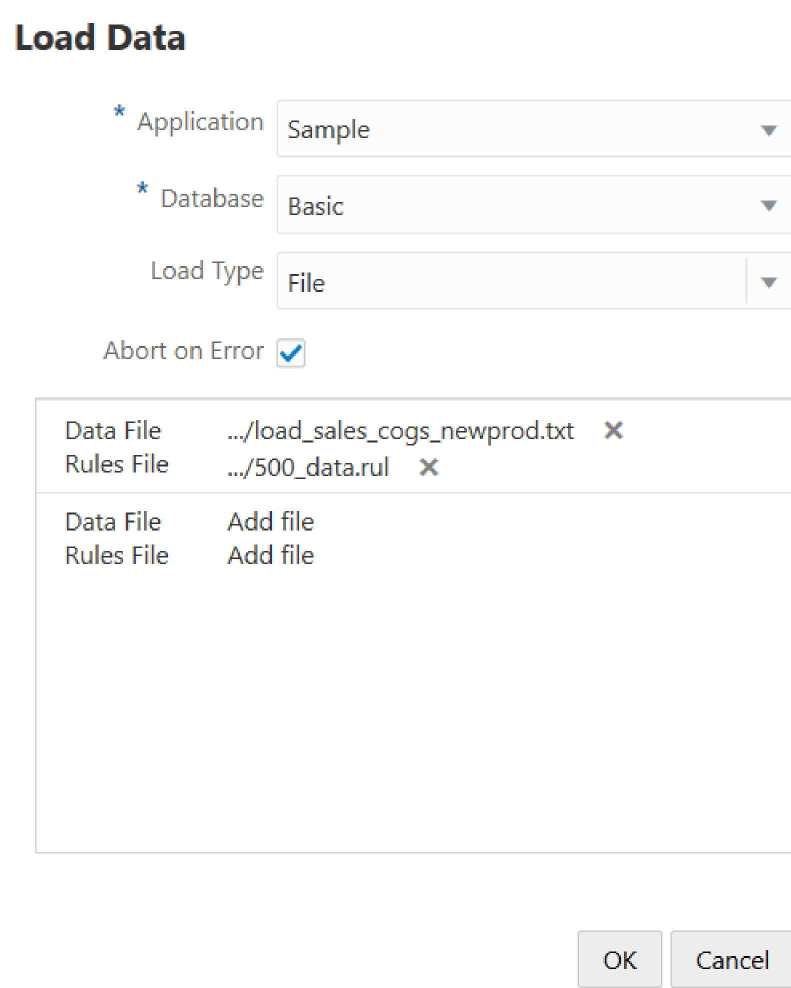
e. Click **Proceed**.

f. The data load rule opens fields mapped to dimensions that were listed in the header record. Preview data (from the text file) populates the grid below the fields.

Field - 1 Product	Field - 2 Market	Field - 3 Year	Field - 4 Scenario	Field - 5 Sales	Field - 6 COGS
500-10	New York	Jan	Actual	678	271
500-10	New York	Jan	Budget	640	260
500-10	New York	Feb	Actual	645	258
500-10	New York	Feb	Budget	610	240
500-10	New York	Mar	Actual	675	270
500-10	New York	Mar	Budget	640	250
500-10	New York	Apr	Budget	670	270
500-10	New York	May	Actual	756	302
500-10	New York	May	Budget	710	280
500-10	New York	Jun	Actual	890	356

Each row in the source data is a *record*, and corresponds with the rows you see in the rule preview data. Fields 1-4 contain metadata (member) information, and Fields 5-6 contain data to load.

6. Verify the rule, then save and close.
7. Close the cube inspector, and click **Jobs** to test your data load rule.
8. Click **New Job** and **Load Data**.
   - For Application, select Sample.
   - For Database, select Basic.
   - For Load Type, specify File.
   - Click **Abort on Error**.
   - Click **Select files from catalog**



- Locate the rule file, `500_data.rul`. Click it, hold the Shift key, and then find and click the data file name (`load_sales_cogs_newprod.txt`). Click Select.
- Click OK.

9. Click the Refresh icon until the job status is Completed. You have loaded data into the new product.

Set File Delimiters

One of your first tasks while defining load rules is to ensure the delimiter setting is appropriate. A file delimiter is the character used to separate fields in a file source of data. Essbase attempts to detect the delimiter when you create the preview in the rule editor. You can set the file delimiter as a comma, tab, whitespace, or custom value. Valid custom values are any characters in the standard ASCII character set numbered from 0 through 127.

> **Note:**
>
> You do not need to set file delimiters for SQL data.

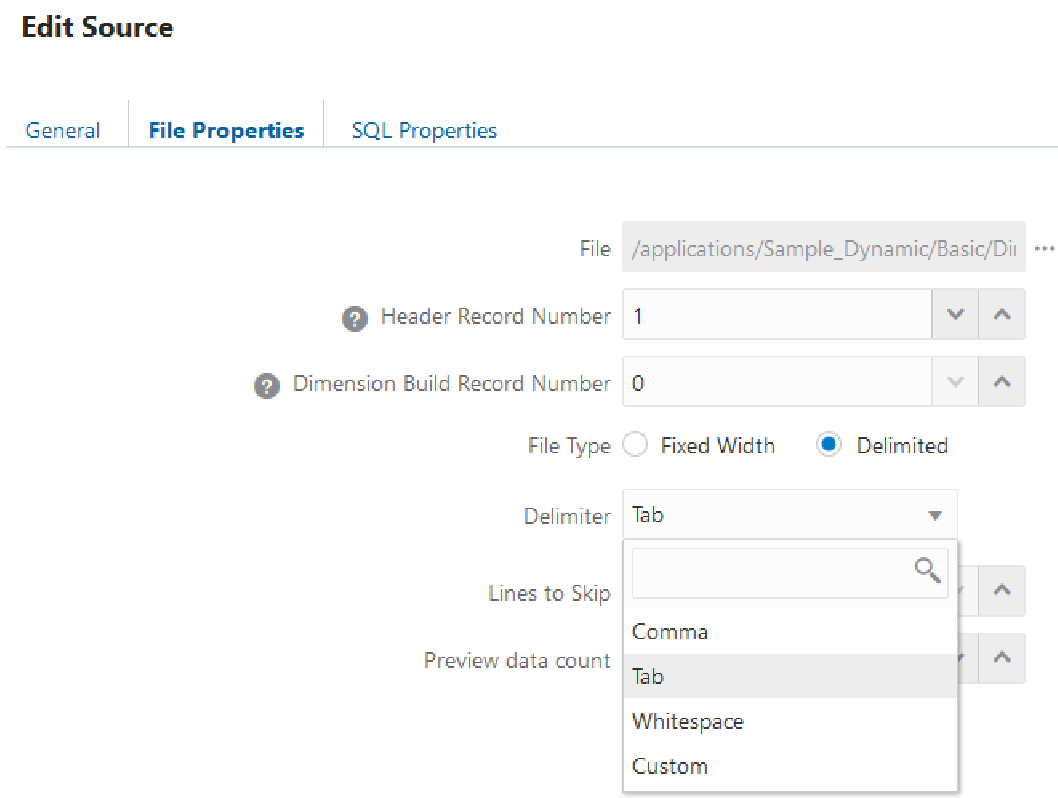
The following is an example of a comma-delimited source:

```
","Year","X","","年","年","Год","Jahr"
"Year","Qtr1","X","Quarter1","第一季","第一四半期","Квартал 1","1. Quartal"
"Qtr1","Jan","","January","一月","１月","Январь","Januar"
"Qtr1","Feb","","February","二月","２月","Февраль","Februar"
"Qtr1","Mar","","March","三月","３月","Март","März"
"Year","Qtr2","X","Quarter2","第二季","第二四半期","Квартал 2","2. Quartal"
"Qtr2","Apr","","April","四月","４月","Апрель","April"
"Qtr2","May","","May","五月","５月","Май","Mai"
"Qtr2","Jun","","Jun","六月","６月","Июнь","Juni"
"Year","Qtr3","X","Quarter3","第三季","第三四半期","Квартал 3","3. Quartal"
"Qtr3","Jul","","July","七月","７月","Июль","Juli"
"Qtr3","Aug","","August","八月","８月","Август","August"
"Qtr3","Sep","","September","九月","９月","Сентябрь","September"
"Year","Qtr4","X","Quarter4","第四季","第四四半期","Квартал 4","4. Quartal"
"Qtr4","Oct","","October","十月","１０月","Октябрь","Oktober"
"Qtr4","Nov","","November","十一月","１１月","Ноябрь","November"
"Qtr4","Dec","","December","十二月","１２月","Декабрь","Dezember"
```

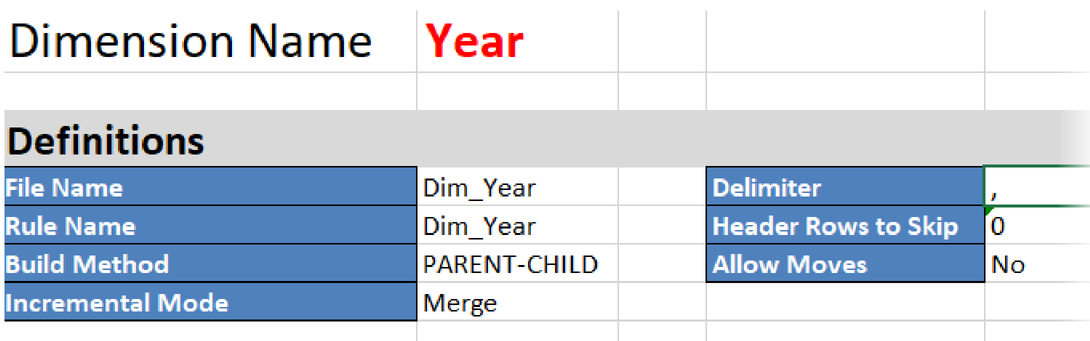
The following is an example of a tab-delimited source:

```
ProductSKU    California    Oregon    Washington    Utah    Nevada
100-20    -1587    338    231    398    86
200-20    2685    1086    579    496    167
300-30    1328    288    1217    413    362
```

To set the delimiter while designing or editing a load rule in the Essbase web interface, go to **Source Properties** and select the **File Properties** tab.



To set the delimiter in an application workbook prior to cube deployment, you can accept or change the default in the definitions of any dimension worksheet.



Set Dimension Build Field Type Information

If you are not performing a dimension build, skip this topic.

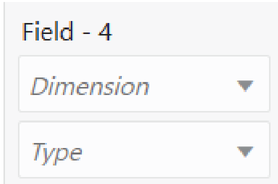
About Dimension Build Field Types

Fields in source data records are descriptive of members you add or change in Essbase. Fields can contain information about member names, member properties, and attribute associations. To enable Essbase to process field information it encounters in source data, you specify field types in the dimension build rule.

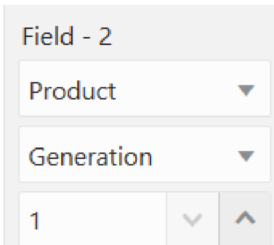
When you set field types in the rule, you specify at least the following information:

- The dimension to which the members of that column belong.

- The type of field to expect in that column, such as a parent, child, generation, property, or alias. The field type depends on the data source and the build method used.



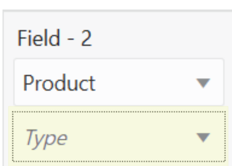
- Additional information depending on the field type. For example, for a generation field type, you also specify the generation number of the members of that column.



Field Types and Valid Build Methods

When designing a dimension build rule, you assign types to the source data fields to help Essbase build the dimension correctly.

In the Essbase web interface, you assign the type below the dimension name:



The following table lists field types and valid build methods.

Table 13-1   Field Types and Valid Build Methods

Field Type[1]	What the Field Contains	Valid Build Methods
Alias	An alias The alias value will not be assigned to the new member if **Member update dimension build** is set to **Remove unspecified** and the data source for a new member contains the alias value of a removed member. It will be assigned if you specify **Reset Dimension**. In a Regular rule, there must be a Generation or Level field with the same value, or, a preceding field must be a parent or child. In an Index-based rule, there must be a Generation or Level field with the same value, or, a preceding field must be a parent or child.	Generation, level, and parent-child references

Table 13-1 (Cont.) Field Types and Valid Build Methods

Field Type[1]	What the Field Contains	Valid Build Methods
Property	A member property. In the source data, use the following codes to specify the property. **Consolidation Operator** • `+` - Add • `-` - Subtract • `*` - Multiply • `/` - Divide • `%` - Percent • `~` - Ignore • `^` - Never consolidate **Data storage type** • `S` - Stored • `N` - Never share • `O` - Label only • `X` - Dynamic Calc **Time Balance** • `K` - None • `F` - First • `L` - Last • `A` - Average **Two Pass Calculation** • `T` - True • `G` - False **Skip Option** • `P` - None • `M` - Missing • `Z` - Zeroes • `B` - Missing and zeroes **Expense Reporting** • `E` - True • `J` - False • `Q` - Date **Hierarchy Type** • `C` - Stored • `D` - Dynamic • `H` - Multiple hierarchies enabled	Generation, level, and parent-child references
Formula	A formula	
Currency name	(Block storage outlines only) A currency name	
Currency category	(Block storage outlines only) A currency category	
UDA	A UDA	
Attribute Dimension	An attribute dimension. Available in Regular and Index-based rules.	

Table 13-1 (Cont.) Field Types and Valid Build Methods

Field Type[1]	What the Field Contains	Valid Build Methods
Attribute parent	In an attribute dimension, the name of the parent member of the attribute member in the subsequent field. Only needed if the outline is duplicate member enabled, and there are two attribute members of the same name. Only available for Regular rules.	
The name of a specific attribute dimension	A member of the specified attribute dimension. This member is associated with a specified generation or level of the selected base dimension.	
Generation	The name of a member in the specified generation	Generation references
Duplicate generation	The name of a member with a shared member as a child. Only available for Regular rules.	
Duplicate generation alias	The alias for the shared member. Only available for Regular rules.	
Level	The name of a member in a level. Only available for Regular rules.	Level references
Duplicate level	The name of a member with a shared member as a child. Only available for Regular rules.	
Duplicate level alias	The alias for the shared member. Only available for Regular rules.	
Parent	The name of a parent.	Parent-child reference
Child	The name of a child. There must be a parent field somewhere preceding it.	
Solve order	Priority in which the member is calculated (0 to 127)	
Member ID	Unique identifier for a member, separate from its name. Only available for Index-based rules. When you build dimensions, all members (including shared members) have unique member IDs. If you migrate an existing outline using one of the Essbase migration tools, then member IDs are automatically assigned during migration. If you patch Essbase without migrating using one of the Essbase migration tools, then member IDs are assigned on the first successful edit of the outline after you apply the patch.	

Table 13-1 (Cont.) Field Types and Valid Build Methods

Field Type[1]	What the Field Contains	Valid Build Methods
Reference Member	Supported only in Index-based rules, this type offers flexibility in modifying existing members. Use of this type requires the field to refer to an already-existing member. For a use case, see "Bonus Exercise" in Create New Dimension Build Rule	Generation References
Aggregate Level Usage	Options for influencing default and query-based view selection on aggregate storage cubes. • 11 - Default • 12 - Consider all levels • 13 - Do not aggregate • 14 - Consider bottom level only • 15 - Consider top level only • 16 - Never aggregate to intermediate levels	
Format String	A format string, if your cube is enabled for typed measures. The format string lets you format the values of numeric typed measures so that they appear in queries as text, dates, or other predefined types. For example, using a text-based formatted value, you can display data cells as “High,” “Medium,” and “Low.”	
Prototype	The member ID (or unique name) of the prototype member for this shared member. This field is required for shared members in duplicate member outlines. If this value is not set, the members will be created as duplicate members instead of shared members.	

1 Field types whose names begin with duplicate (such as duplicate generation and duplicate level alias), are not related to duplicate member names described in Creating and Working With Duplicate Member Outlines.

Parent-Child Build Method Example

The following is an example of a dimension build rule that uses the Parent-child build method. Field type of Parent must come first, followed by children.

Field - 1	Field - 2	Field - 3	Field - 4	Field - 5
Year	Year	Year	Year	Year
Parent	Child	Property	Alias	Alias
		Child	Long Names	ChineseNames
			Child	Child
	Year	X		年
Year	Qtr1	X	Quarter1	第一季
Qtr1	Jan		January	一月
Qtr1	Feb		February	二月

Generation Build Method Example

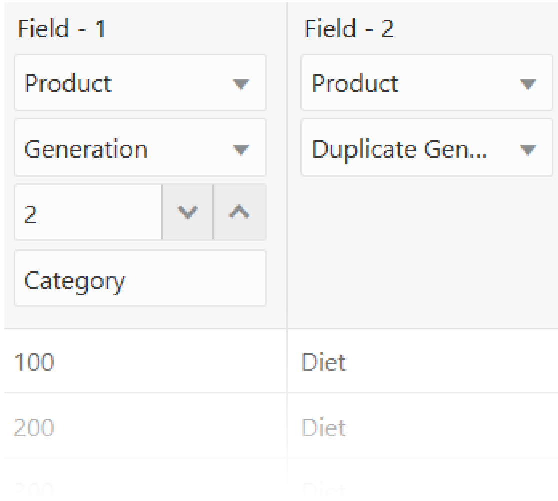
The following is an example of a dimension build rule that uses the Generation build method. Generation 1 is the dimension itself, so the build rule starts with Generation 2. Generations must be in ascending order in the rule, as this is a regular (non-Index-based) rule.

Field - 1	Field - 2	Field - 3
Product	Product	Product
Generation	Generation	Generation
2	3	4
Category	Product SKU	Promotions
500	500-10	500-10-10
500	500-10	500-10-20
500	500-20	500-20-12
500	500-20	500-20-15
500	500-20	500-20-20

Rules for Assigning Field Types

Each build method has rules for assigning field types.

- Generation build method rules for Regular (non-Index) dimension builds:
  - If generation numbers do not start at 2, the first member of the specified generation must exist in the outline.
  - Generation numbers must form a contiguous range. For example, if generation 3 and 5 exist, you must also define generation 4.
  - Put Duplicate Generation fields immediately after generation fields (these are applicable if you use shared members in a generation build method rule). Likewise, put Duplicate Generation Alias fields immidately after Duplicate Generation fields.



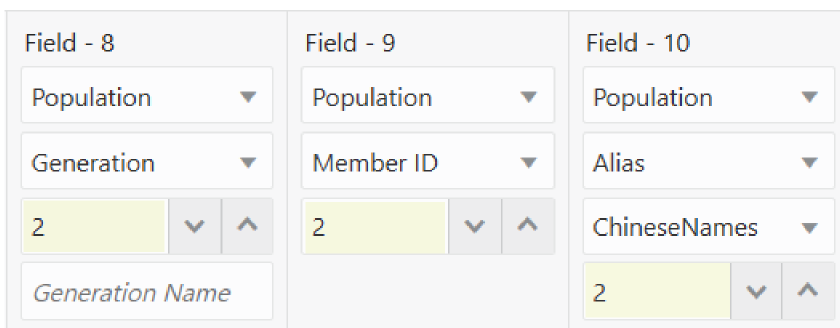
– Group generation fields sequentially within a dimension. For example:

  Product Generation 2, Product Generation 3, Product Generation 4.

– Put attribute association fields after the base field with which they are associated, and specify the generation number of the associated base dimension member. For example:

  Product Generation 2, Product Generation 3, Product Ounces 3

  The generation number must correspond to the generation of the member in the outline for which the field provides values. For example, *Product Generation 3* means that the values in the field are third-generation members of the Product dimension. The `2` in *Population Member ID 2*, and *Population Alias 2* shows that the values in the field are associated with the second-generation member of the Population dimension.



> **Note:**
>
> When using the generation build method to create a duplicate member dimension, the maximum number of generations is 20.

- Level build method rules:
  – Put Duplicate Level fields immediately after level fields (these are applicable if you use shared members in a level build method rule). Likewise, put Duplicate Level Alias fields immediately after Duplicate Level fields.

- Each record must contain a level 0 member. If a level 0 member is repeated on a new record with a different parent, Essbase rejects the record unless you select the Allow Moves member property.
- Group level fields sequentially within a dimension.
- Put the fields for each roll-up in sequential order.
- Use a single record to describe the primary and secondary roll-ups.
- Put attribute association fields after the base field with which they are associated, and specify the level number of the associated base dimension member. For example:

  Product Level 3, Product Ounces 3, Product Level 2
- The level number must correspond to the level of the member in the outline for which the field provides values. For example, the 3 in *Product Level 3* shows that the values in the field are level 3 members of the Product dimension. The `2` in *Population Alias 2* shows that the values in the field are associated with the second level of the Population dimension.

- Parent-child build method rules:

  If field type is parent or child, enter 0 (zero) in the Number text box.
- Attribute dimension name build method rules:

  The generation or level number must correspond to the generation or level of the associated base member in the outline. For example, the `3` in *Product Ounces 3* shows that the values in the field are the members of the Ounces attribute dimension that are associated with the third-generation member of the Product dimension in the same source data record.

If necessary, move the fields to the required locations.

Select a Build Method

When you modify dimension properties impacting the Essbase outline during a dimension build, one essential property you need is a selection of the appropriate build method that matches your source data.

If you are not performing a dimension build, skip this section.

If you are building a new dimension or adding members to an existing dimension, you must specify a build method for each dimension that you are creating or modifying. See Dimension Build Methods.

In application workbooks, you set the build method for a dimension on the dimension worksheet (Dim.*dimname*). See Understanding Dimension Worksheets.

Set Dimension Build Operational Instructions

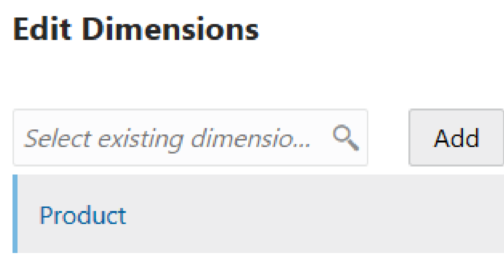
Modify dimensional properties to make adjustments to the Essbase outline, during a dimension build, that are based on elements of the source data.

Select the appropriate build method to match source data, enable shared/alternate hierarchies or duplicate members, influence ASO view selection, set solve order, control adds/sorts, allow/disallow updates to member properties, UDAs, formulas, and attributes, and change the behavior of accounts or attributes dimensions.

If you are not performing a dimension build, skip this section.

If you're performing a dimension build, you can define operations to be performed after the data source has been read. These operational specifications are always available in the dimension properties for the rule. Some changes affect all members of the selected dimension, some affect only the selected dimension, and some affect all dimensions in the rule file.

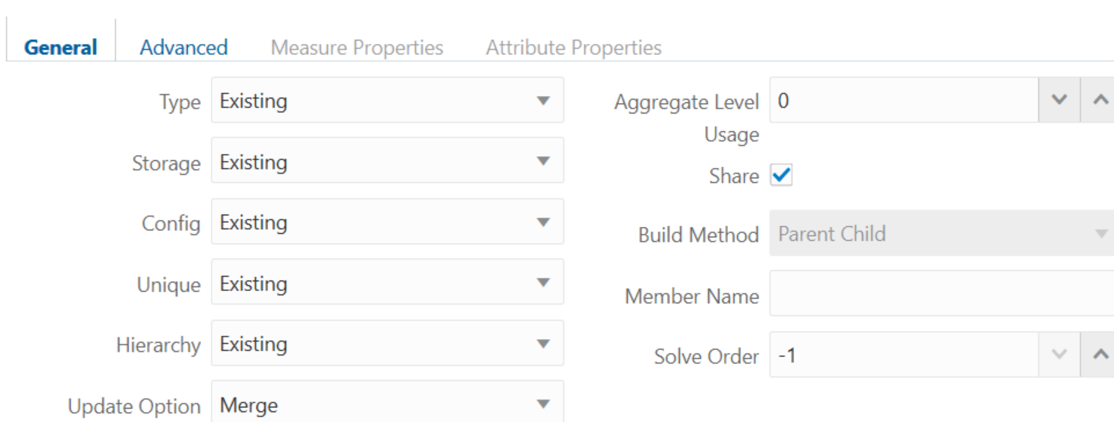
1. Select a field.
2. Click **Dimensions**.
3. Select an existing dimension, or type and add one, and then select it. For example, click Product.



4. The properties are available to edit. All dimensions have General and Advanced properties you can modify. Only accounts dimensions have Measure properties, and only attribute dimensions have Attribute properties. **Existing** means the dimension build will not affect this property.

General Dimension Properties

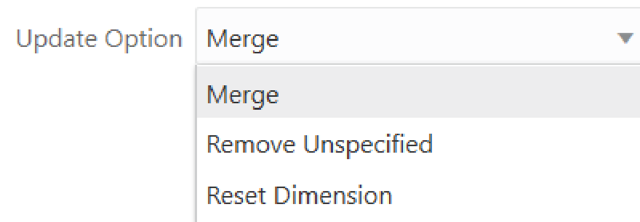
The following properties help you build or update dimensions and members, select a build method, enable shared/alternate hierarchies or duplicate members, influence ASO view selection, define solve order, and more.



Property	Description
Type	Information type of the dimension.
Storage	Data storage property for members in the dimension.
Config	Dense or sparse configuration of the dimension.
Unique	The uniqueness setting for all member names in the cube.

<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>Hierarchy</td><td>The type of aggregate storage outline hierarchy for the generation 1 member of this dimension (the dimension member itself).</td></tr>
<tr><td>Update Option</td><td>Incremental dimension build update option.

Incremental dimension building is the process of updating existing dimensions by adding, moving, deleting, and reordering members.

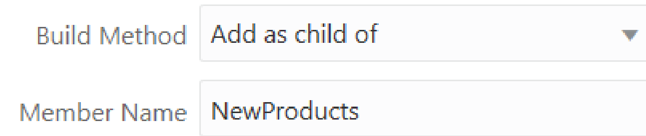


Merge is the default. This option adds the new members to the dimension while retaining the existing members.

Remove Unspecified removes members that are not specified in the source file.

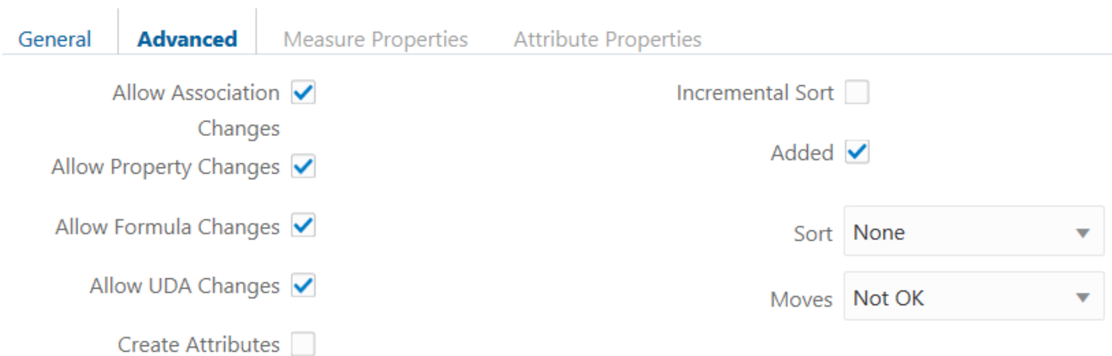
Removing unspecified members is available only with the generation reference, level reference, and parent-child reference build methods. Outlines are invalid if removing members results in level 0 Dynamic Calc members without formulas.

Reset Dimension (available only for Index-based rules) enables you to perform the following operations during the dimension build as you make incremental updates, and the placement you selected for the members in the hierarchy will persist (instead of members being moved to the end).
reorder members
insert new members
move or delete a member, while retaining shared members
move a parent member and have the children move with it
This option rebuilds the entire dimension. To use Reset Dimension, leave Moves set to Not OK on the Advanced tab (otherwise, you cannot build shared member hierarchies).</td></tr>
<tr><td>Aggregate Level Usage</td><td>View selection properties, applicable only to aggregate storage cubes.</td></tr>
<tr><td>Share</td><td>Whether the dimension includes shared members.</td></tr>
<tr><td>Build Method</td><td>The build method of this dimension (editable only for new dimensions).</td></tr>
</table>

Property	Description
Member Name	Parent member name under which to add children. Used with **Add as child of** build method. 
Solve Order	The solve order of the dimension.

Advanced Dimension Properties

The following dimension build properties control how members can be added, sorted, and moved, as well as whether the dimension build is allowed to update member properties, user defined attributes, outline formulas, and attribute members or their associations.

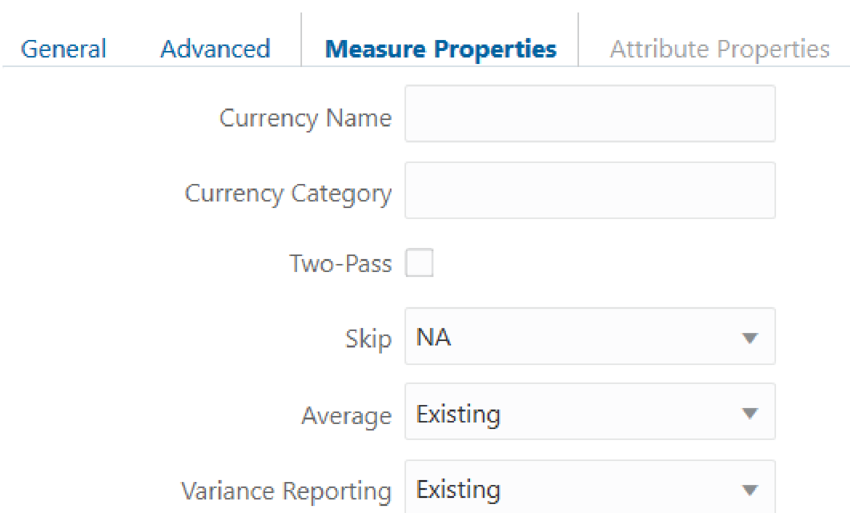


Property	Description
Allow Association Changes	Changes member attribute associations to the associations specified in the source data (For example, if the source data associates the 100-10 member with the 8 Ounce attribute and the outline associates the 100-10 member with the 12 Ounce attribute, Essbase associates the 100-10 member with the 8 Ounce attribute.)
Allow Property Changes	Changes member properties to the properties specified in the source data; makes the Allow UDA Changes option available.
Allow Formula Changes	Changes member formulas to the formulas specified in the source data.
Allow UDA Changes	Changes UDA values to the values specified in the source data (If the option is not selected, source-data UDAs are added to existing UDAs).
Create Attributes	If unchecked, prevents Essbase from creating attribute members.
Incremental Sort	Whether to sort members after Essbase has processed and added all members from the source data.

Property	Description
Added	Select if this is a new dimension that does not yet exist in the outline. If left unchecked, the dimension build will fail for a new dimension. If checked, the new dimension will be created during the dimension build.
Sort	Whether and how to sort children in hierarchies: None (do not sort), ascending, or descending.
Moves	Whether to allow moving of members and their children to new parents. Recognizes primary members and matches them with the source data. Not available for duplicate member outlines. Leave as Not OK if using the Reset Dimension update option.

Measure Dimension Properties

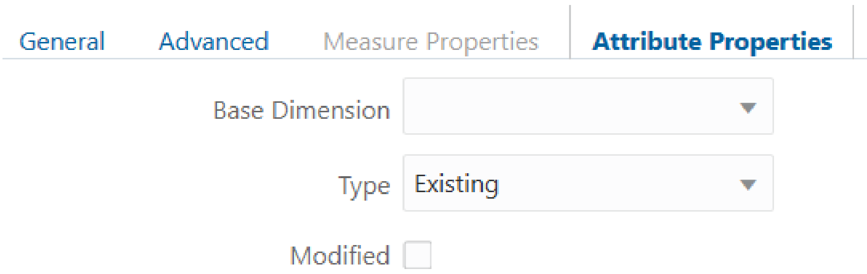
The following dimension build properties enable you to make changes to the behavior of the Accounts dimension as you build it.



Property	Description
Currency Name	If the dimension is associated with a currency conversion application, the currency name.
Currency Category	If the dimension is associated with a currency conversion application, the currency category.
Two-Pass	Whether the dimension is tagged for two-pass calculation.
Skip	If you set the Average value (for time balance) as first, last, or average, then set the Skip property to indicate what to do when missing values or values of 0 are encountered. See Setting Skip Properties.
Average	The time balance property, if used.
Variance Reporting	Whether to treat accounts members as expense items

Attribute Dimension Properties

The following dimension build properties enable you to modify the behavior of any attributes dimensions being built.



Property	Description
Base Dimension	Name of the associated base dimension.
Type	Attribute type. Attribute dimensions have a text, numeric, Boolean, or date type that enables different functions for grouping, selecting, or calculating data. Although assigned at the dimension level, the attribute type applies only to level 0 members of the attribute dimension.
Modified	Select if this dimension build requires modifications to a dimension's properties, such as solve order or aggregate level usage.

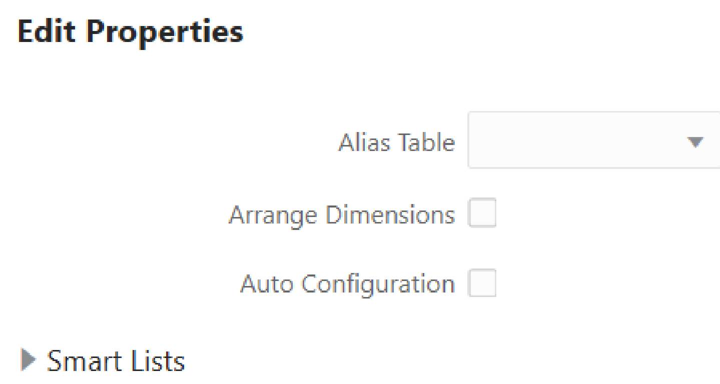
Some of the dimenson properties correspond to fields and values described in Understand Dimension Worksheets.

Set Dimension Build Global Properties and General Source Properties

Use the *global properties* of an Essbase dimension build rule to set properties affecting all dimensions in the rule, including alias table, dimension arrangement, and auto configuration. Use *source properties* of a dimension build or data load rule to specify the handling of headers and tokens in the source data.

Dimension Build Global Properties

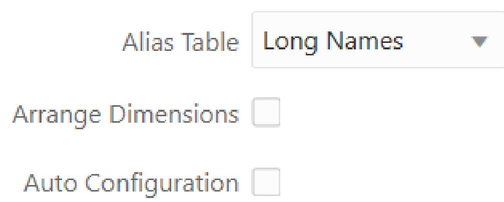
You can change the following global properties, affecting all dimensions included in a dimension build rule.



Select Alias Table

You can select which alias table to update with new aliases from the source data. If you do not specify an alias table, the dimension build updates the default alias table.

For example, the following selection indicates to Essbase that any alias updates found in the source data should affect only the Long Names alias table, when you run the dimension build.



To specify which alias table to update,

1. Open or create a dimension build rule.
2. Click **Global Properties**.
3. From the **Alias Table** selector, choose an alias table, and click OK.
4. Verify, save, and close the rule.

Arrange Dimensions for Calculation Performance

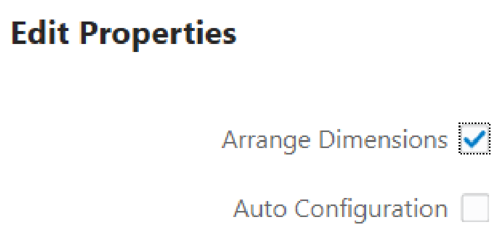
Dimension placement (within outlines) affects calculation time.

This setting is applicable to block storage cubes only. In global properties for a dimension build rule, you can optimize calculation performance by placing dimensions in the following order (called an hourglass arrangement):

1. Densest dimensions (accounts and time)
2. Remaining dense dimensions (largest to smallest)
3. Sparse dimensions (smallest to largest)
4. Attribute dimensions (no particular order)

To arrange the dimensions this way (applicable only for block storage outlines),

1. Open or create a dimension build rule.
2. Click **Global Properties**.
3. Select **Arrange Dimensions**, and click OK.



4. Verify, save, and close the rule.

Auto Configure Density and Sparsity

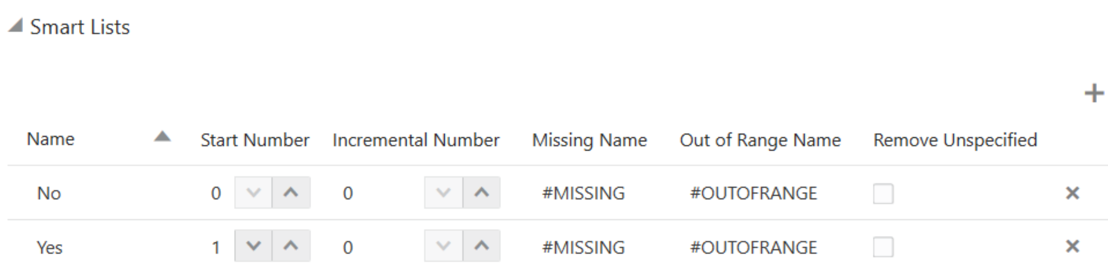
This setting is applicable to block storage cubes only.

By default, dense and sparse settings for dimensions are kept as either the existing setting or the setting specified in the dimension build rule. To override this default and let Essbase automatically assign dimensions to be dense or sparse,

1. Open or create a dimension build rule.
2. Click **Global Properties**.
3. Select **Auto Configuration**, and click OK.
4. Verify, save, and close the rule.

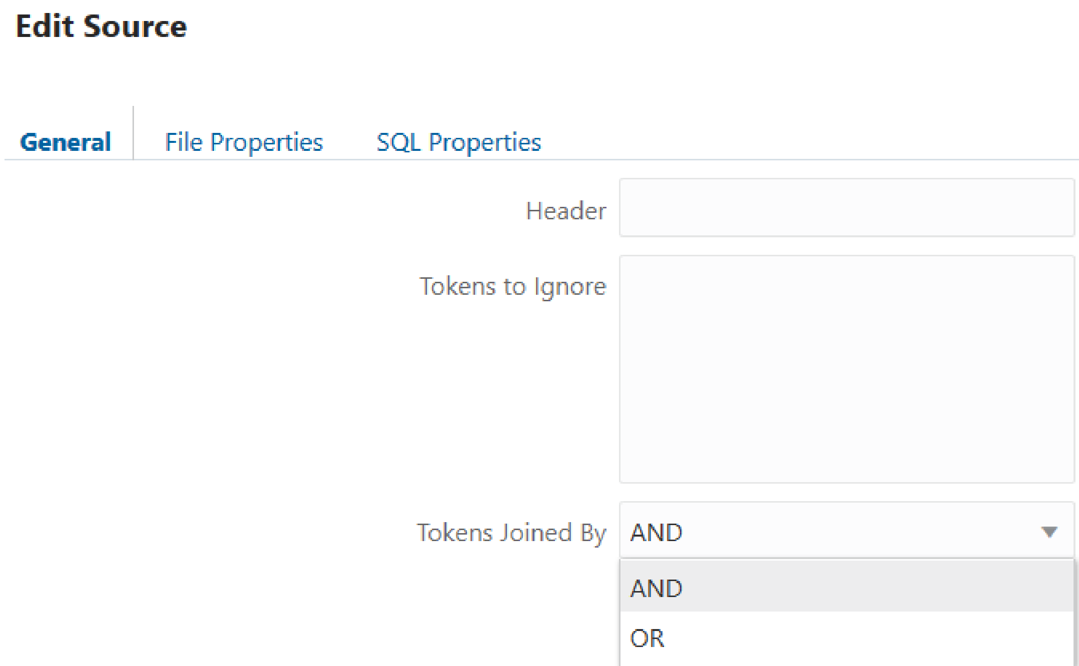
Add or Update Smart Lists (Text List Objects)

Expand the Smart Lists section to add or update a text list object, also known as a Smart List. Text list objects are a way to store metrics as textual values, when your accounts dimension is designed to work with text measures. You can implement text measures when your outline is enabled to work with typed measures. To see a sample cube that uses a textual measure dimension, import the sample application Facility Rating, available in the gallery section of the Files catalog.



General Source Properties (for Dimension Build and Data Load Rules)

Using the general Source Properties of a dimension build or data load rule, you can broadly set how the rule behaves with respect to the source data.



To specify general source data options in the rule,

1. Open or create a dimension build or data load rule.
2. Click **Source Properties**.
3. On the **General** tab,
   - Enter a Header string used in the source data, if known. This will be skipped during the dimension build or data load.
   - If needed, enter source data records to ignore, based on the presence of a given token, during the dimension build or data load. A token is one delimited string in the source data.
   - If you listed multiple tokens to ignore, specify AND if Essbase should ignore only records that contain all of the tokens. Specify OR if Essbase should ignore records that contain any of the tokens.
4. Click OK, then verify, save, and close the rule.

Use the Source Data to Work with Member Properties

You can apply codes for member properties directly into the source data if you want a dimension build to apply or reset properties of new or existing members in the Essbase outline. Member properties include consolidation, formulas, UDA, expense, time balance, label only, never share, stored, dynamic, and more.

You can modify the properties of new and existing members during a dimension build by:

- Including member properties in a field in the source data
- Leaving the source data field empty to reset the property to the default value, or to remove the formula or UDA

In the source data, put the properties in the field directly following the field containing the members that the properties modify. For example, to specify that member Margin% not roll up into its parent and not be shared:

1. Position the ~ property (which indicates that the member should not roll up into its parent) and the N property (which indicates that the member should not be shared) after the Margin% field. For example:

```
Margin%  Margin%  ~ N Sales
```

> **Note:**
>
> Margin % being repeated because of Alias.

2. Set the field type for the properties fields to Property.

Removing a formula, UDA, or attribute, or resetting a property to its default value, includes the following additional step: Leave the field NULL or empty in the source data.

The table below lists all member codes used in the source data to assign properties to block storage outline members.

Table 13-2 Member Property Codes for Block Storage Outlines

Code	Description
%	Express as a percentage of the current total in a consolidation
*	Multiply by the current total in a consolidation
+	Add to the current total in a consolidation
-	Subtract from the current total in a consolidation
/	Divide by the current total in a consolidation
~	Exclude from the consolidation
^	Exclude from all consolidations in all dimensions
A	Treat as an time balance average item (applies to accounts dimensions only)
B	Exclude data values of zero or #MISSING in the time balance (applies to accounts dimensions only)
E	Treat as an expense item (applies to accounts dimensions only)
F	Treat as a time balance first item (applies to accounts dimensions only)
G	Reset two-pass calculation to false (applies to accounts dimensions only). Also see property code T in this table.
J	Reset expense item to false (applies to accounts dimensions only). Also see property code E in this table.
K	Reset the time balance property to NONE (applies to accounts dimensions only). Also see property codes A, F, and L in this table.
L	Treat as a time balance last item (applies to accounts dimensions only)
M	Exclude data values of #MISSING from the time balance (applies to accounts dimensions only)

Table 13-2 (Cont.) Member Property Codes for Block Storage Outlines

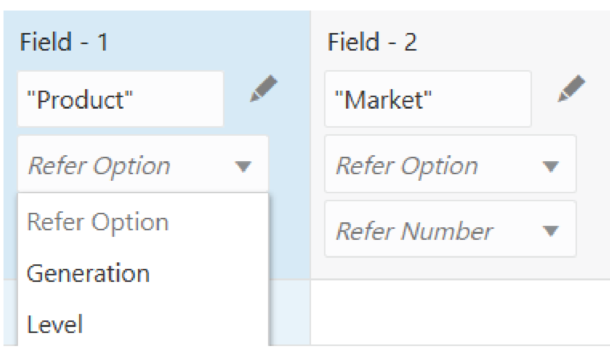
Code	Description
N	Never allow data sharing
O	Tag as label only (store no data)
P	Reset the time balance skip option to NONE (applies to accounts dimensions only). Also see property codes B, M and Z in this table.
S	Set member as stored member (non-Dynamic Calc and not label only)
T	Require a two-pass calculation (applies to accounts dimensions only)
X	Create as Dynamic Calc
Z	Exclude data values of zero from the time balance (applies to accounts dimensions only)

Define Data Load Field Properties

In a data load rule, each field must map to a corresponding member in the Essbase outline, or be marked as a data field or an ignored field.

For duplicate member outlines that require a load rule, if the source data is not a direct export from a duplicate member cube, you may also need to specify the dimension and the *Refer Option* that indicates the build method (level reference or generation reference) that Essbase uses to map the field.

You must map each rule field to the corresponding outline member, or mark it as a data field or ignored field. Other field characteristics may also apply.



- Use the level reference method when fields within a dimension are organized bottom-up in the source data.
- Use the generation reference method when fields within a dimension are organized top-down in the source data. For example:

```
State,"New York","100-10",Jan,Sales,Actual,42
City,"New York","100-20",Jan,Sales Actual,82
State,Texas,"100-10",Jan,Sales,Actual,37
```

Perform Operations on Records

Using fields in the Essbase data load or dimension build rule, you can define selection or rejection criteria for records in the source. You can also customize the rule to indicate how Essbase should handle header records for a data load.

- Select and Reject Records
- Set the Records Displayed
- Handle Header Records

Select and Reject Records

You can build filters into Essbase data load and dimension build rule fields to define precise selection and rejection criteria for records. For example, you can reject certain records before they are loaded into the cube.

Select Records

You can specify which records Essbase loads into the database or uses to build dimensions by setting selection criteria. *Selection criteria* are string and number conditions that must be met by one or more fields within a record for Essbase to load the record. If a field or fields in the record do not meet the selection criteria, Essbase does not load the record. You can define one or more selection criteria. For example, to load only 2019 Budget data from a data source, create a selection criterion to load only records in which the first field is Budget and the second field is 2019. If you define selection criteria on multiple fields, you can specify how Essbase combines the criteria.

Reject Records

You can specify which records Essbase ignores by setting rejection criteria. *Rejection criteria* are string and number conditions that, when met by one or more fields within a record, cause Essbase to reject the record. You can define one or more rejection criteria. If no field in the record meets the rejection criteria, Essbase loads the record. For example, to reject Actual data from a data source and load only Budget data, create a rejection criterion to reject records in which the first field is Actual.

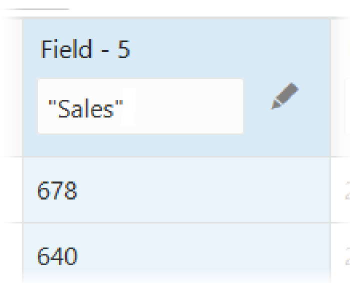
Combine Multiple Select and Reject Criteria

When you define select and reject criteria on multiple fields, you can specify how Essbase combines the rules across fields: whether the criteria are connected logically with AND or with OR. If you select AND from the Boolean group, the fields must match all of the criteria. If you select OR, the fields must match only one of the criteria. The global Boolean setting applies to all select or reject operations in the rule, for data load and dimension build fields.

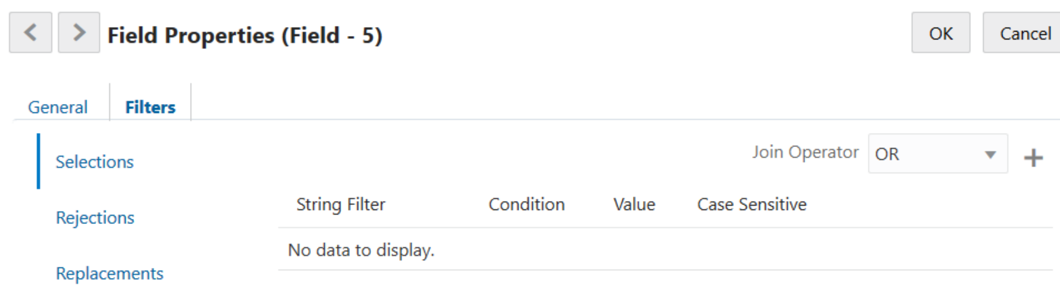
> **Note:**
>
> If selection and rejection criteria apply to the same record (you define select and reject criteria on the same record), the record is rejected.

To filter the records,

1. Select the field to which to apply the criteria.



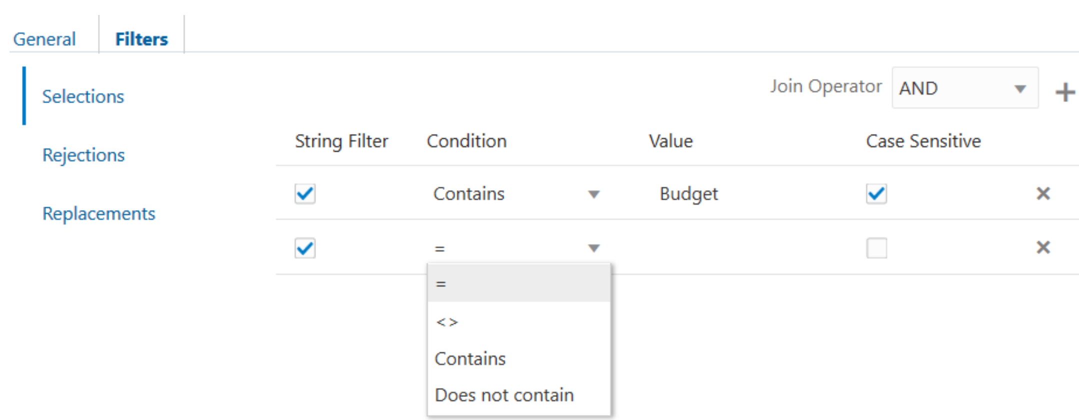
2. Click **Field Properties**, and click **Filters**.
3. Click the **+** icon to add a filter.



4. Add as many filter critera as needed. These instructions apply for Selection as well as Rejection criteria. Selection/Rejection criteria applied to a field let you define filters to be met for approval or rejection of a data load or dimension build record when loading to the Essbase cube. Filtration options include string or numeric matching of specific values, logical join options, and case sensitivity.

   For example, for Condition, if you select String Filter, the condition applies to string matching. If you do not select String Filter, the condition applies to numeric values.

   For string matching, click String Filter and choose conditions:



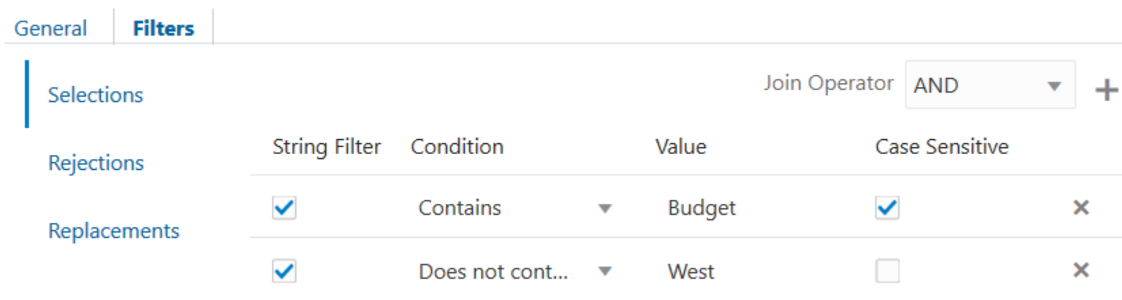
   a. = means the value you specify must be an exact string match
   b. <> means the value you specify must not be not a string match
   c. `Contains` means a specified string match must be found

d. `Does not contain` means a specified string match must not be found

e. Case sensitivity can optionally be applied

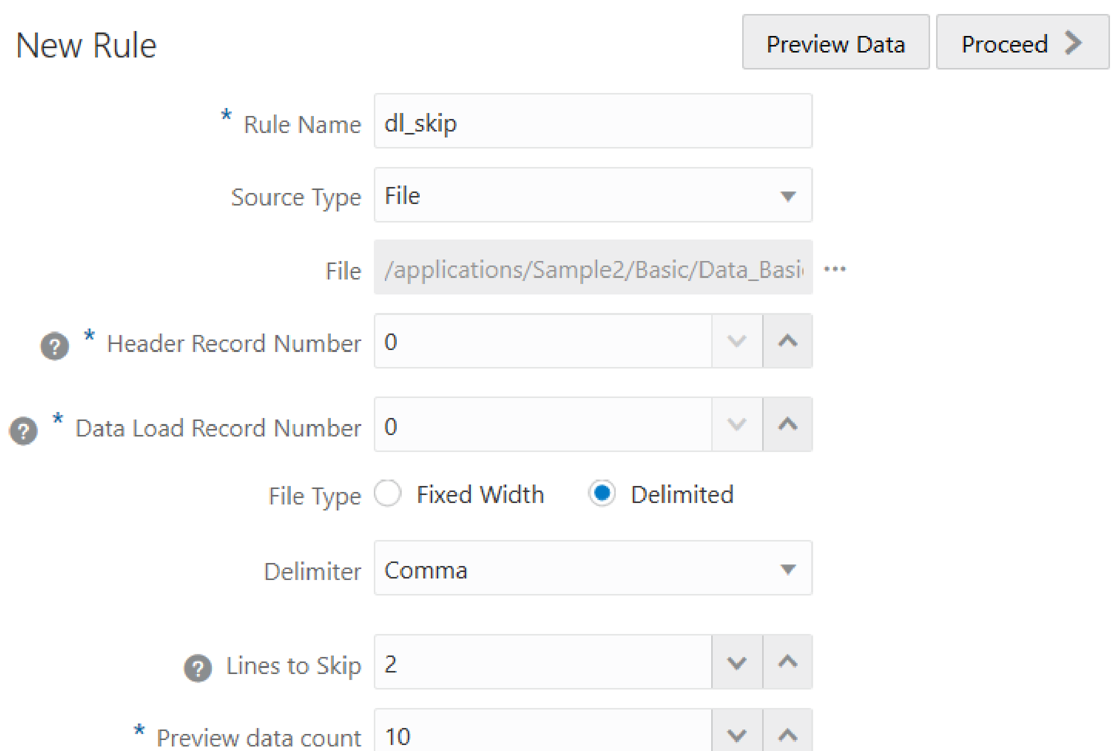
For numeric data value matching, use the mathematical Condition operands as needed (`=`, `<>`, `>`, `<`, `>=`, `<=`).

5. If you define more than one filter, choose a logical join operator, AND or OR. AND means that all the defined selection or rejection criteria must apply (if even one criterion is not met for any given record, then the filter does not apply to that record). OR means the opposite (if even one criterion is met for a given record, the filter applies).



Set the Records Displayed

You can specify the number of records to display in the rule preview, and you can also tell Essbase to skip a certain number of records from the data load or dimension build. For example, in the following new data load rule definition, the first `2` records will be skipped from the data load (but will not be omitted from the preview). No more than 10 records will display in the preview (this limit does not affect the data load operation).



> **Note:**
>
> Essbase treats header records the same as data records when counting the records to skip.

Handle Header Records

Header records can be used or skipped in Essbase data load rules, and should be skipped in dimension build rules. Using fields in the rule, you can specify how Essbase should handle header records, if they exist in the source data.

Sources of data can contain:

- *Data records*, which contain member fields and data fields
- *Header records*, which describe the contents of the source data

> **Note:**
>
> When you are loading SQL data, the data load rule cannot include header records.

The following is an example of a data file with a header record:

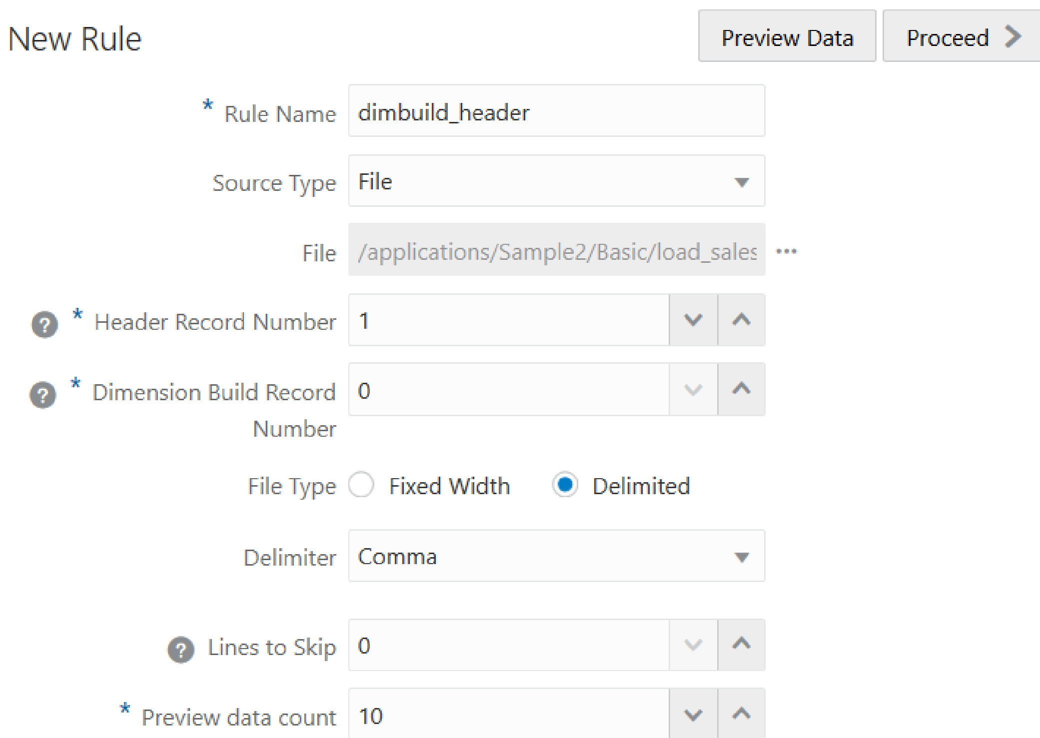
```
"Product","Market","Year","Scenario","Sales","COGS"
"100-10","New York","Jan","Actual","678","271"
"100-10","New York","Jan","Budget","640","260"
"100-10","New York","Feb","Actual","645","258"
"100-10","New York","Feb","Budget","610","240"
"100-10","New York","Mar","Actual","675","270"
"100-10","New York","Mar","Budget","640","250"
"100-10","New York","Apr","Budget","670","270"
"100-10","New York","May","Actual","756","302"
"100-10","New York","May","Budget","710","280"
"100-10","New York","Jun","Actual","890","356"
"100-10","New York","Jun","Budget","840","340"
```

Header Records and Dimension Build Rules

Header records are not useful for dimension build rules. To design a dimension build rule that skips the header record,

1. Upload a text file like the one above to the Sample Basic cube directory.
2. On the **Applications** page, expand the application (Sample).
3. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.
4. Select the **Scripts** tab, and then click **Rules**.
5. Click **Create** and choose **Dimension Build (Regular)**.
6. Choose any name for the rule.

7. For **Source Type**, select File, and for **File**, navigate to the source data file.
8. Increment **Header Record Number** to 1.



9. Preview the data and click **Proceed**.

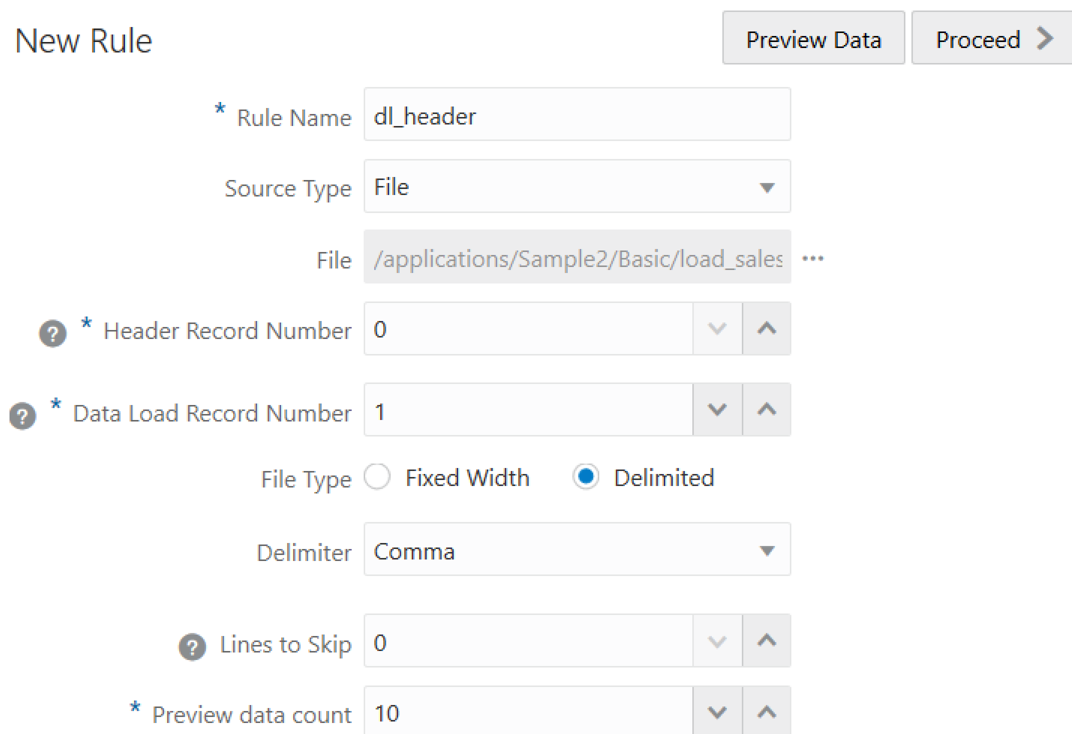
   The header record is omitted from the preview.

Field - 1	Field - 2	Field - 3	Field - 4	Field - 5	Field - 6
Dimension	*Dimension*	*Dimension*	*Dimension*	*Dimension*	*Dimension*
Type	*Type*	*Type*	*Type*	*Type*	*Type*
100-10	New York	Jan	Actual	678	271
100-10	New York	Jan	Budget	640	260
100-10	New York	Feb	Actual	645	258
100-10	New York	Feb	Budget	610	240
100-10	New York	Mar	Actual	675	270
100-10	New York	Mar	Budget	640	250
100-10	New York	Apr	Budget	670	270
100-10	New York	May	Actual	756	302
100-10	New York	May	Budget	710	280
100-10	New York	Jun	Actual	890	356

Header Records and Data Load Rules

Header records can be used or skipped in data load rules. To design a data load rule that incorporates the header record,

1. Complete steps 1-4 in the previous procedure.
2. Click **Create** and choose **Data Load**.
3. Choose any name for the rule.
4. For **Source Type**, select File, and for **File**, navigate to the source data file.
5. Increment **Data Load Record Number** to 1.



6. Preview the data and click **Proceed**.

   The header information is incorporated into the field properties, in your preview.

Field - 1 Product	Field - 2 Market	Field - 3 Year	Field - 4 Scenario	Field - 5 Sales	Field - 6 COGS
100-10	New York	Jan	Actual	678	271
100-10	New York	Jan	Budget	640	260
100-10	New York	Feb	Actual	645	258
100-10	New York	Feb	Budget	610	240
100-10	New York	Mar	Actual	675	270
100-10	New York	Mar	Budget	640	250
100-10	New York	Apr	Budget	670	270
100-10	New York	May	Actual	756	302
100-10	New York	May	Budget	710	280
100-10	New York	Jun	Actual	890	356

Perform Operations on Fields

To customize how Essbase builds dimensions or loads data, design the rule to perform operations on fields in the source data. You can use the rule to add, join, split, copy, rename, and move fields, ignore fields or strings, and map data fields to member names.

See:

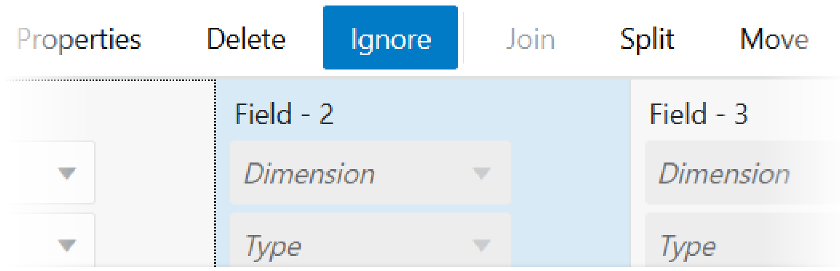
- Ignoring Fields
- Ignoring Strings
- Moving Fields
- Adding Fields
- Joining Fields
- Creating a Field by Joining Fields
- Splitting Fields
- Copying Fields
- Mapping Fields
- Changing Field Names

Ignoring Fields

To have Essbase skip one or more fields of the source data when building dimensions or loading data, use the rule to ignore those fields.

Ignored fields from the source will not be loaded into the Essbase cube. For example, the Sample.Basic cube has five standard dimensions: Year, Product, Market, Measures, and Scenario. If the source has an extra field that is not a member of any dimension, such as Salesperson, you can tell Essbase to ignore the Salesperson field.

To ignore a column, select it and click **Ignore**.



Ignoring Strings

To have Essbase ignore certain source data fields during a dimension build or data load, such as those with certain characters, use the source properties of the rule to specify the matching string/token criteria.

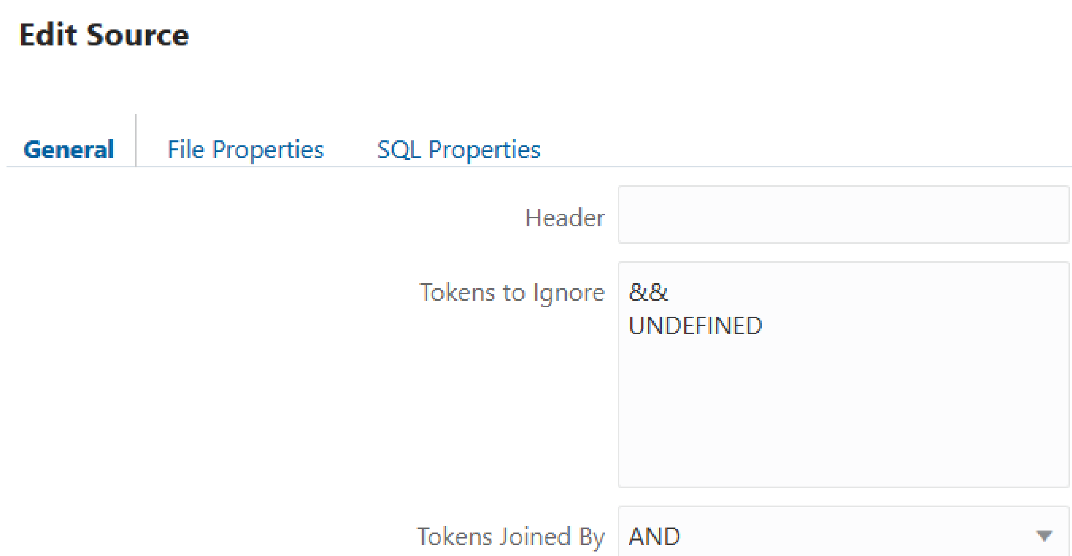
For your data load and dimension build rules, you can ignore any field in the source that matches a string (called a *token*). When you ignore fields based on string values, the fields are ignored everywhere they appear in the source data, not just in a particular column. For example, in a source that is a computer-generated report in text format, special ASCII characters might be used to create horizontal lines between pages or boxes around headings. These special characters can be defined as tokens to be ignored.

To ignore strings/tokens in the source data,

1. In the rule editor, click **Source Properties**.

2. Enter the string(s) to ignore in **Tokens to Ignore**.

3. If you entered more than one string, select AND or OR in **Tokens Joined By** to control whether the string matching is done logically with AND or with OR. If you select AND, both

strings must be present in the source to be ignored. If you select OR, either one or the other must be present.

4. Click OK. Verify and save the rule.

Moving Fields

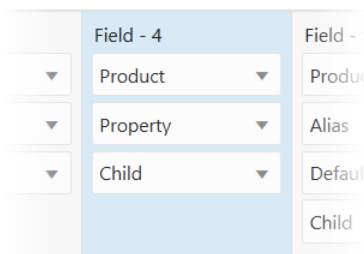
Use rules to have Essbase move one or more fields of the source data when building dimensions or loading data. Moving fields in the rule can help when source records need to be arranged to match the outline.

You can move fields to a different location using a rules file. For example, you can specify the first field in the source data to be the third field during the data load or dimension build.

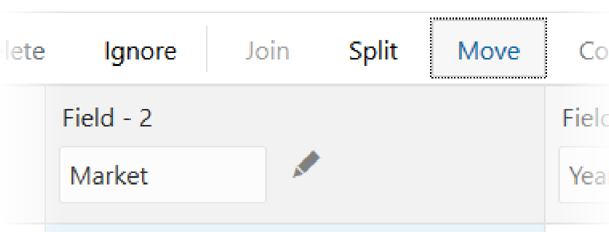
Move operation is not available in Index-based dimension build rules, where fields can be in any order.

To move a field,

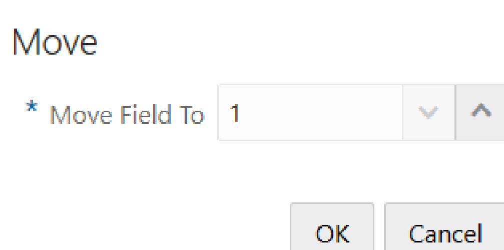
1. Select the field.



2. Click **Move**.



3. Indicate the position to move to, and click OK.



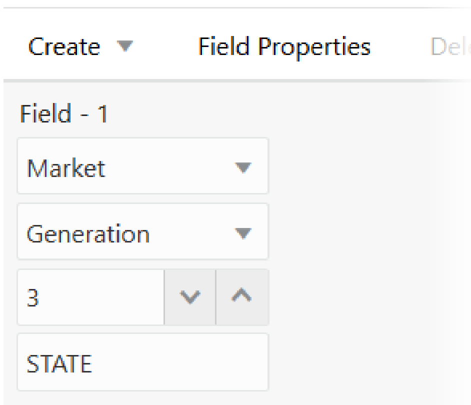
Adding Fields

Adding fields to a rule can be useful for adding new members or data to the Essbase outline, to prepare for joins, to build qualified member names, or to perform other operations required for an Essbase dimension build or data load.

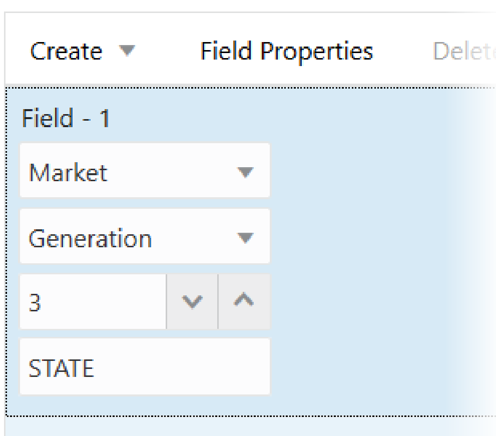
When adding a new field to a load rule,

- If you select an existing field and then add a new field, the new column is added to the left of the existing column.
- If you select no fields, the new field is added to the end (to the right of all existing fields).

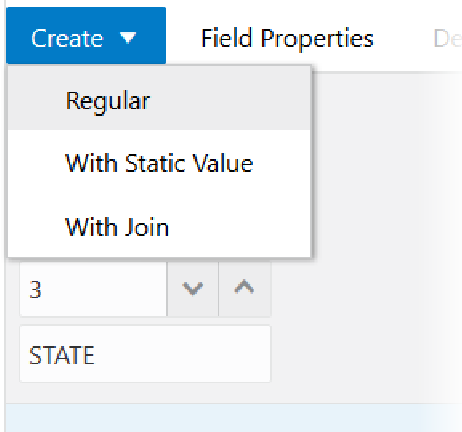
In the following dimension build rule example, assume you want to add a Generation 2 field prior to the existing Generation 3 field.



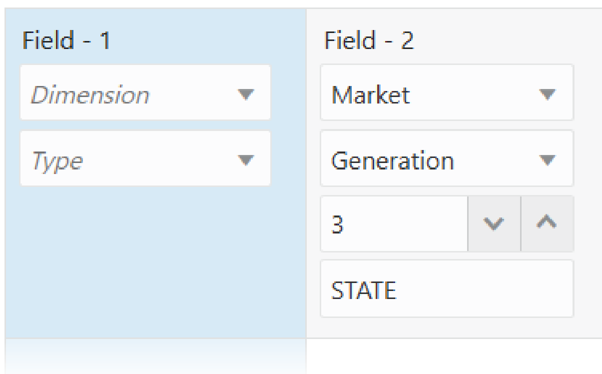
1. Select the field so that it is highlighted.



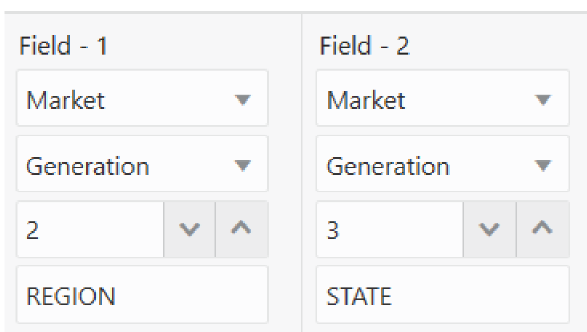
2. Click **Create > Regular**.



Field 1 moves to become Field 2, and a new Field 1 is added.



3. Enter the required information for the field you just added.



Joining Fields

You can join multiple fields into one field in an Essbase dimension build or data load rule. The new field is given the name of the first field in the join.

For example, if the source data has separate fields for product number (100) and product family (-10), you must use the rule to join the fields (100-10) before loading them into the Sample.Basic cube.

Consider the following dimension build rule and preview:

Field - 1	Field - 2	Field - 3
Product	Product	Product
Generation	Generation	Generation
2	3	4
Category	Product SKU	Promotions
500	500-10	500-10-10
500	500-10	500-10-20
500	500-20	500-20-12
500	500-20	500-20-15
500	500-20	500-20-20

Assume your goal is to join fields to create qualified member names of the following format:

```
[500].[500-10].[500-10-10]
```

To get started,

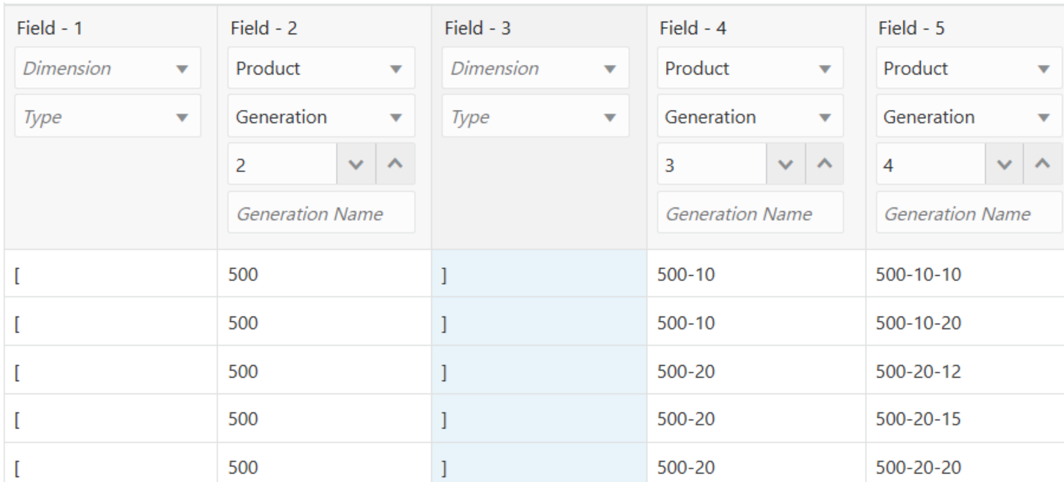
1. Select Field 1 and click **Create > With Static Value**.
2. Enter **[**.

   Field 1 is replaced with a new field containing the character you entered, and the other fields are shifted right.

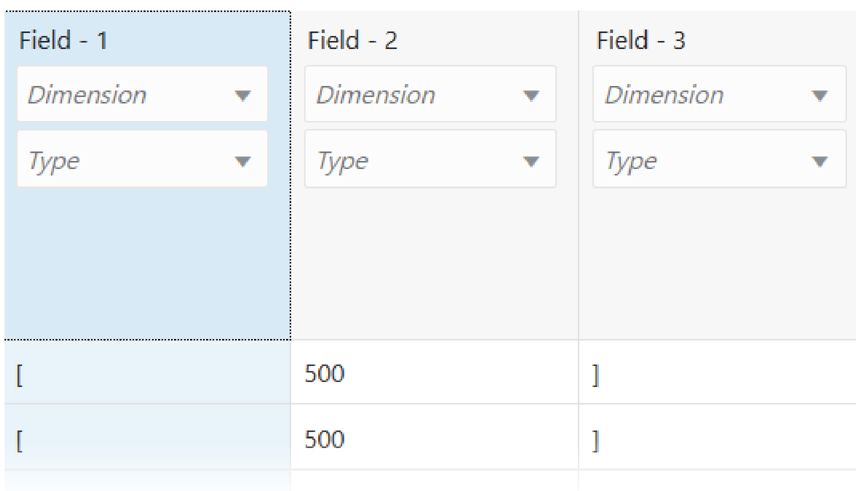
Field - 1	Field - 2	Field - 3	Field - 4
Dimension	Product	Product	Product
Type	Generation	Generation	Generation
	2	3	4
	Generation Name	*Generation Name*	*Generation Name*
[	500	500-10	500-10-10
[	500	500-10	500-10-20
[	500	500-20	500-20-12
[	500	500-20	500-20-15
[	500	500-20	500-20-20

3. Select Field 3 and click **Create > With Static Value**.
4. Enter **]**.

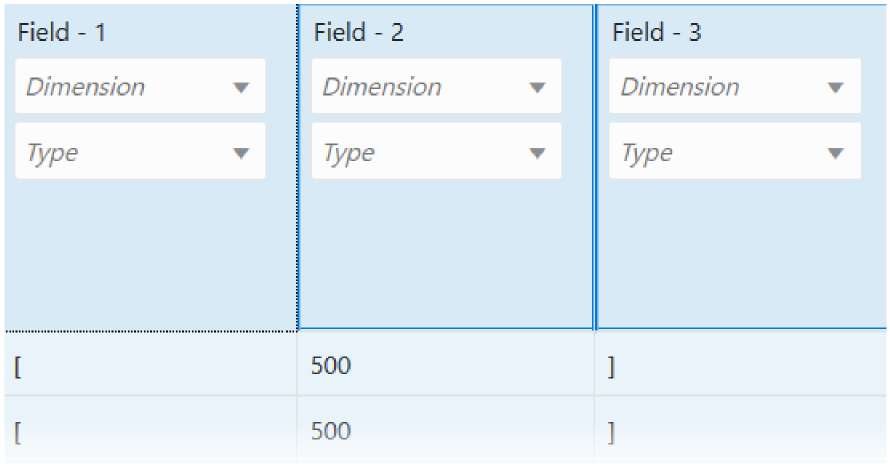
   Field 3 is replaced with a new field containing the character you entered, and fields 4 and 5 are shifted right.



5. Select Field 1.

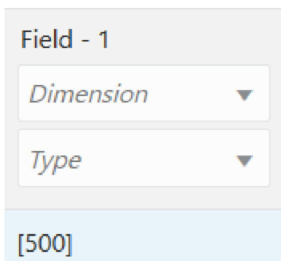


6. Hold Shift while clicking fields 2 and 3, to add fields 2 and 3 to the current selection.



7. Click **Join**.

   Fields 1, 2, and 3 are joined into a new Field 1 containing `[500]`.

   If your joined Field contains the brackets going the wrong directions, such as `]500[`, then you probably selected in order of Fields 3, 2, 1 instead of Fields 1, 2, and 3. The order of your selections impacts how the Join behaves. To fix this, select the incorrectly joined field and click **Split** (using position 1, and then position 3), and try again, reversing your selection order before clicking **Join**.

8. Add more static values for the periods and brackets, to practice more joins until you have built a field containing a qualified member name of the desired format:

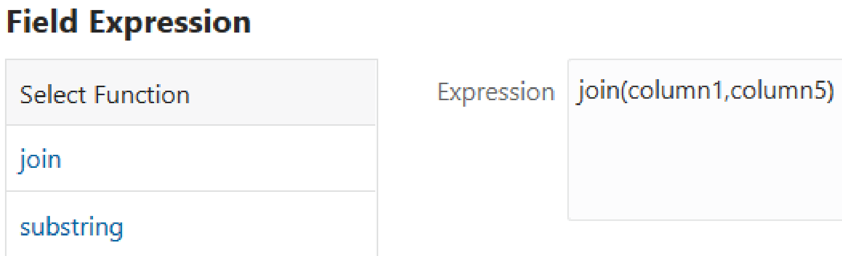
   ```
   [500].[500-10].[500-10-10]
   ```

9. If you need to review your history of column operations including joins, splits, moves, and static value creation, click **Column Operations** to see an ordered list of recent operations.

Field Expressions (Available for Index-Based Dimension Build Rules)

Although Index-based rules do not have split, join, and move column operations, you can join fields and perform other operations using a field expression.

1. Select a field and click **Expression** to add an operational expression to a field.
2. To join two columns, click a function and provide the column indices as arguments. For example,



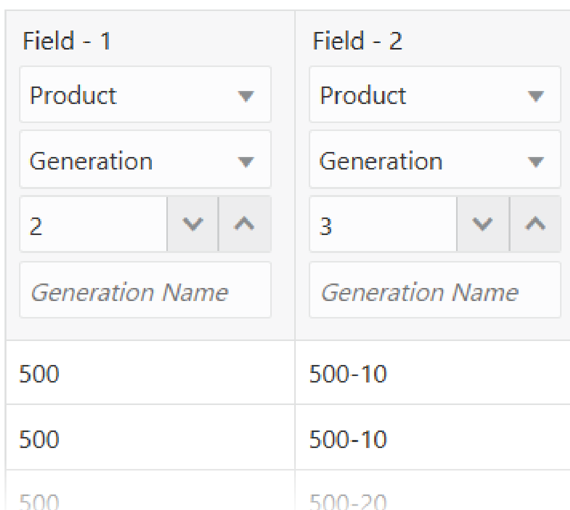
Creating a Field by Joining Fields

In the previous topic, Joining Fields, the join replaced existing fields in the Essbase dimension build rule.

You can also join fields by placing the joined fields into a new field, leaving the original fields intact. Creating a field in a rule is useful if you need to concatenate fields of the source data to create a member in Essbase outline.

For example, if a source of data has separate fields for product number (Example: `100`) and product family (Example: `-10`), you may need to join the fields (Example: `100-10`) before you load them into the Sample.Basic cube. If, however, you want to preserve the two existing fields in the source, you can create a new field (`100-10`) using a join, while preserving the original fields (`100` and `-10`).

Consider the following dimension build rule and preview:

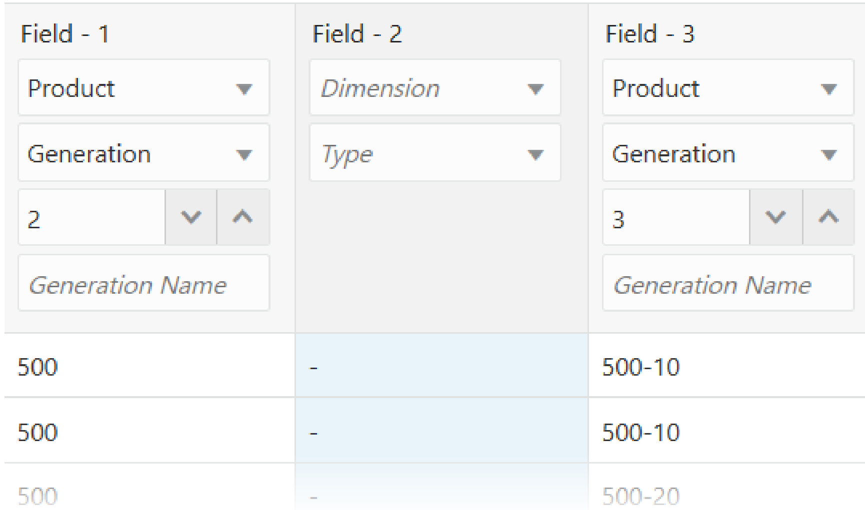


Assume your goal is to add a generation 4 field of the following format:

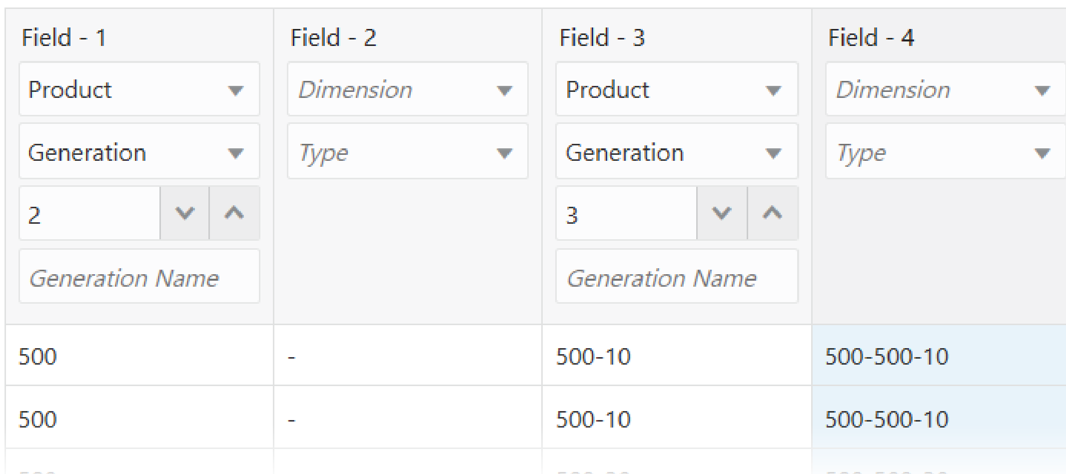
```
500-500-10
```

To get started,

1. Select Field 2 and click **Create > With Static Value**.
2. Enter a hyphen **-** and click OK.



3. Select Fields 1, 2, and 3 (in that order), and click Create > With Join, using join position 4.

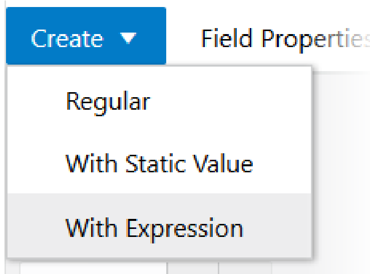


4. If you need to review your history of column operations including joins, splits, moves, and static value creation, click **Column Operations** to see an ordered list of recent operations.
5. You can move Field 2 out of the way to reuse later. Select it and click **Ignore** if you do not want it to affect the rule functionality.

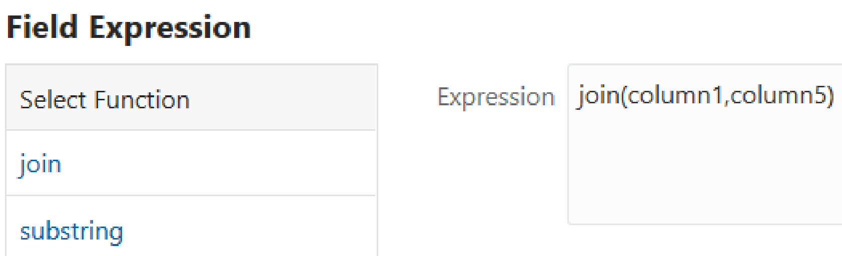
Field Expressions (Available for Index-Based Dimension Build Rules)

Although Index-based rules do not have split, join, and move column operations, you can join fields and perform other operations using a field expression.

1. Click **Create > With Expression** to add a field that you will define using an expression.



2. To join two columns, click a function and provide the column indices as arguments. For example,



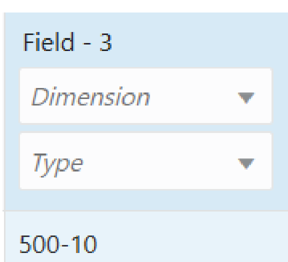
Splitting Fields

You can split a field into two fields in an Essbase dimension build or data load rule.

This operation is the opposite of a join, and is available only in Regular rules. For example, if a data source for the Sample.Basic database has a field containing UPC100-10-1, you can split "UPC" out of the field and ignore it. Then, only 100-10-1, the product number, is loaded.

To split a field,

1. Select the field.



2. Click **Split**.
3. Indicate the split position in terms of characters from left to right. For example, to separate `10` away from `500-`, indicate split position 4.

4. Click OK. Verify and save the rule.

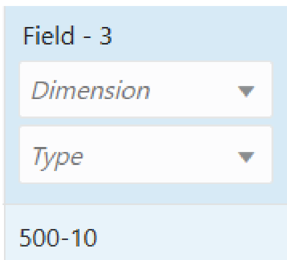
Copying Fields

You can create a copy of a field in an Essbase dimension build or data load rule, leaving the original field intact.

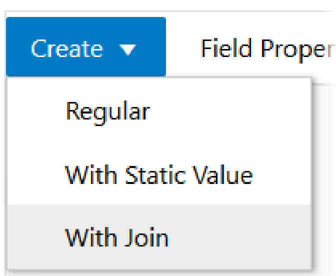
For example, if, during a single dimension build, you want to define a multilevel attribute dimension and associate attributes with members of a base dimension, you must copy some of the fields.

To copy a field

1. Select the field.



2. Click **Create** > **With Join**.



3. Enter the number of the field you want to duplicate.
4. Click OK. Verify and save the rule.

Mapping Fields

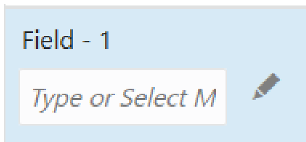
You can map source data fields to Essbase member names in a data load rule.

This section applies to data load only. If you are performing a dimension build, skip this section.

Map fields in the source directly to fields in the Essbase cube during a data load by specifying which field in the source maps to which member or member combination in the Essbase cube. The source data is not changed.

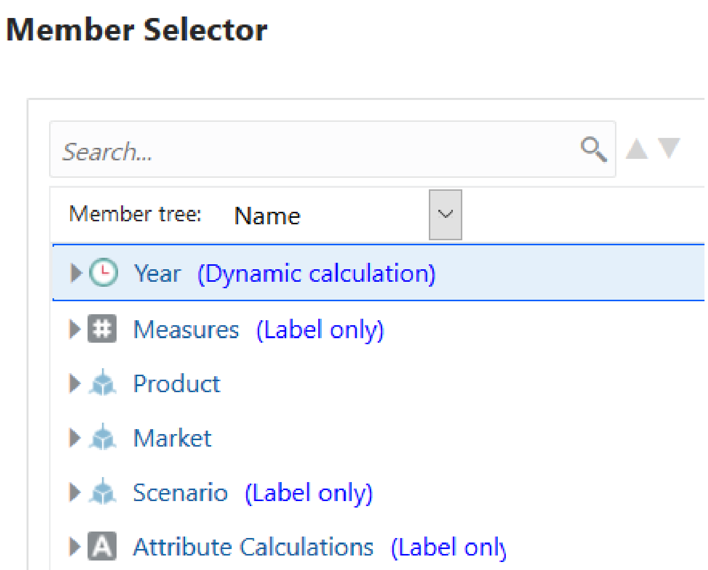
> **Note:**
>
> When you open a SQL data source, the fields default to the SQL data source column names. If the SQL column names and the Essbase dimension names are the same, you need not map the column names.

1. Select the unmapped data load rule field.

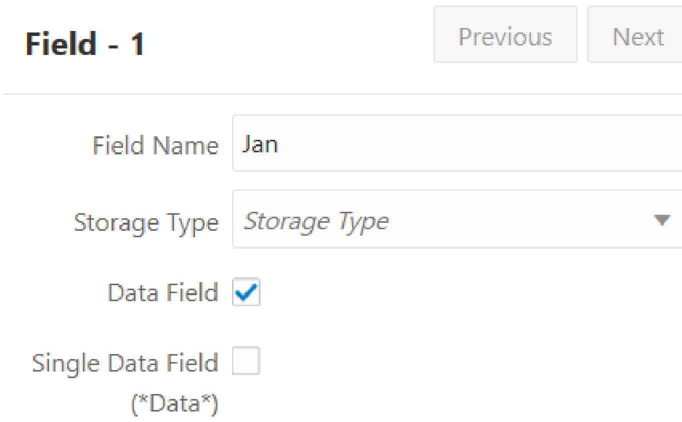
   

2. Either type a member name, or click the edit field/member selection tool next to the field.
   

3. If you are using the member selection tool, find or navigate to the member you want to associate with this field.

4. Double click the member so that it is added to the **Field Name**.



5. Click OK. Verify and save the rule.

Changing Field Names

You can name the fields in a rule to match the Essbase outline's dimension names during a data load.

To load a source of data, you must specify how the fields of the source map to the dimensions and members of the cube. Rules can translate fields of the source so that the fields match member names each time the source is loaded. This process does not change the source.

The rules file:

- Maps member fields of the source file to dimensions and members of the cube
- Maps data fields of the source file to member names or member combinations (such as Jan, Actual) of the cube

Naming Fields

To name a field,

1. Select the field.

2. Click **Field Properties**.

3. Enter a name in the **Name** field. If the name has a space in it, enclose it in quotation marks.

4. Click OK. Verify and save the rule.

Replacing Text Strings

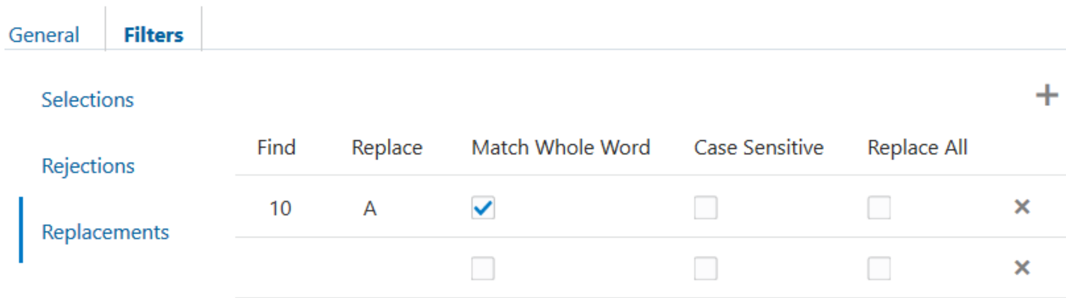
You can replace text strings so that the fields map to Essbase member names during a data load or dimension build. The data source is not changed. For example, if the data source abbreviates New York to NY, you can have the rules file replace each NY with New York.

To replace strings for the field,

1. Select the field.

2. Click **Field Properties**.

3. Click the **Filters** tab.
4. Click **Replacements**.
5. Click the **+** icon to add string replacement criteria.
6. Complete the **Find** and **Replace** values. For example, replace `CA` with `California`.

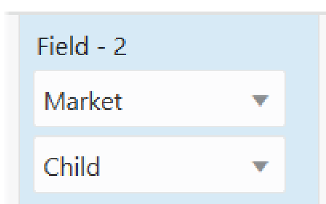


7. Specify if the replacement operation should:
   - Be case-sensitive
   - Make the replacement only if the whole word matches. For example, to replace the `10` in the string `100 10 1` with an A, checking **Match Whole Word** changes the string to `100 A 1`. Leaving **Match Whole Word** unchecked changes the string to `A0 10 1`.
   - Replace all occurrences of the string. For example, if you check **Replace All** to replace all occurrences of `10` in the string `100 10 1` with an A, the string changes to `A0 A 1`.
8. Click OK. Verify and save the rule.

Changing the Case of Fields

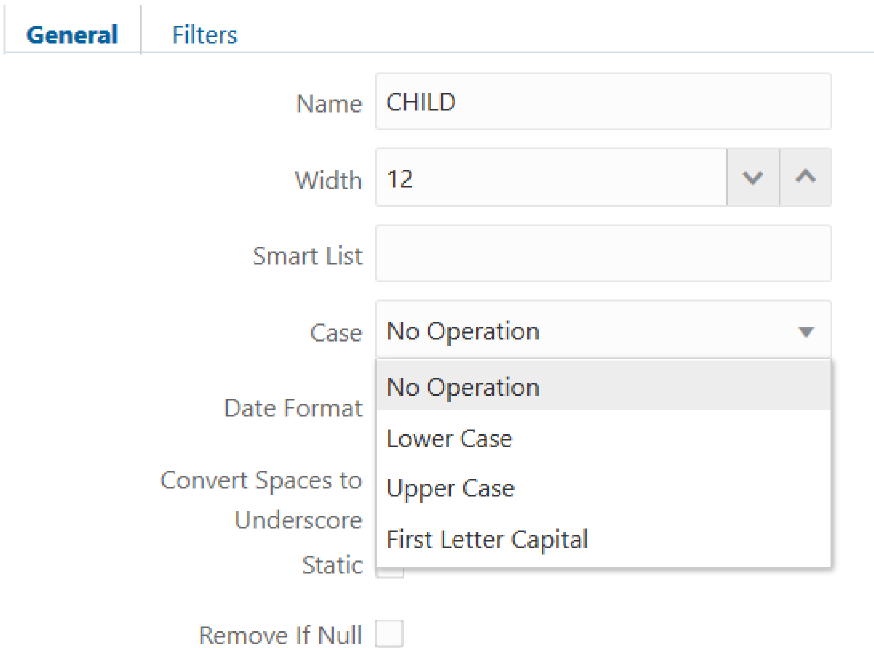
You can change the case of a field so that the field maps to Essbase member names during a data load or dimension build. The data source is not changed. For example, if the data source capitalizes a field (for example, JAN) that is in lowercase in the database (jan), you can have the rules file change the field to lowercase.

1. Select the field.



2. Click **Field Properties**.

3. For **Case**, select an operation to perform on incoming data.

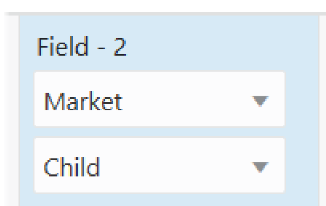


4. Click OK. Verify and save the rule.

Dropping Leading and Trailing Spaces

You can drop leading and trailing spaces from around fields of the source data. A field value containing leading or trailing spaces does not map to a member name, even if the name within the spaces is an exact match.

1. Select the field.



2. Click **Field Properties**.

3. Click **Trim**.
4. Click OK. Verify and save the rule.

By default, Essbase drops leading and trailing spaces.

Converting Spaces to Underscores

You can convert spaces in fields of the data source to underscores to make the field values match the member names of the database.

1. Select the field.

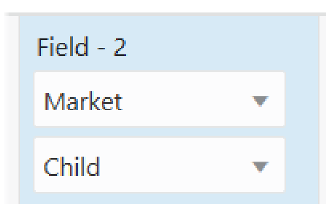
2. Click **Field Properties**.

3. Click **Convert Spaces to Underscore**.
4. Click OK. Verify and save the rule.

Adding Prefixes or Suffixes to Field Values

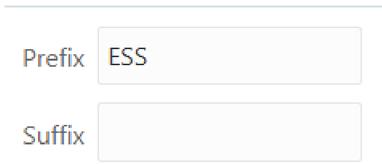
You can add prefixes and suffixes to each field value of the data source. For example, you can add 2017 as the prefix to all member names in the Year dimension.

1. Select the field.



2. Click **Field Properties**.

3. Enter the prefix or suffix to add.



4. Click OK. Verify and save the rule.

Perform Operations on Data

Use Essbase data load rule options to define a field as a data field, add to or substract from existing values, extract summarized data from the source, clear or replace data, scale data values, or flip signs.

This section applies to data load only. If you are performing a dimension build, skip this section.

You can perform operations on the data in a field; for example, moving a field to a new position in the record.

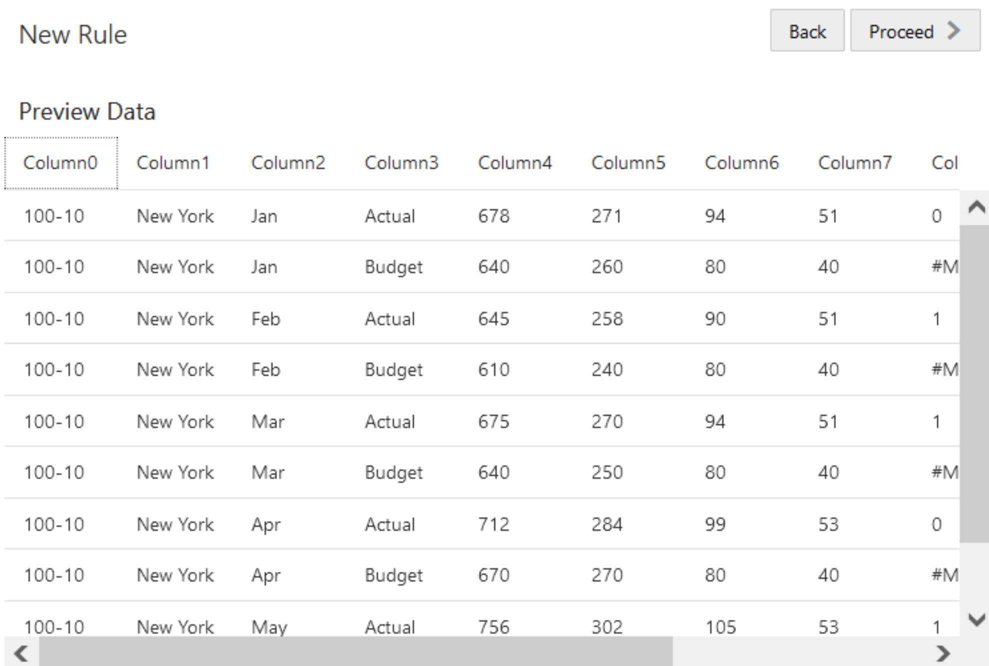
- Defining Columns as Data Fields
- Adding to and Subtracting from Existing Values
- Extracting Source Data Using Column Store Options
- Clearing Existing Data Values
- Replacing All Data
- Scaling Data Values
- Flipping Field Signs

Defining Columns as Data Fields

When records in the source data contain columns for dimensions followed by one or more data columns, use field properties in the Essbase data load rule to indicate the data fields.

This section applies to data load only. If you are performing a dimension build, skip this section.

If each record in the source data contains a column for every dimension followed by one or more data columns, as in the following example, use the Field Properties to define the data columns as data fields.



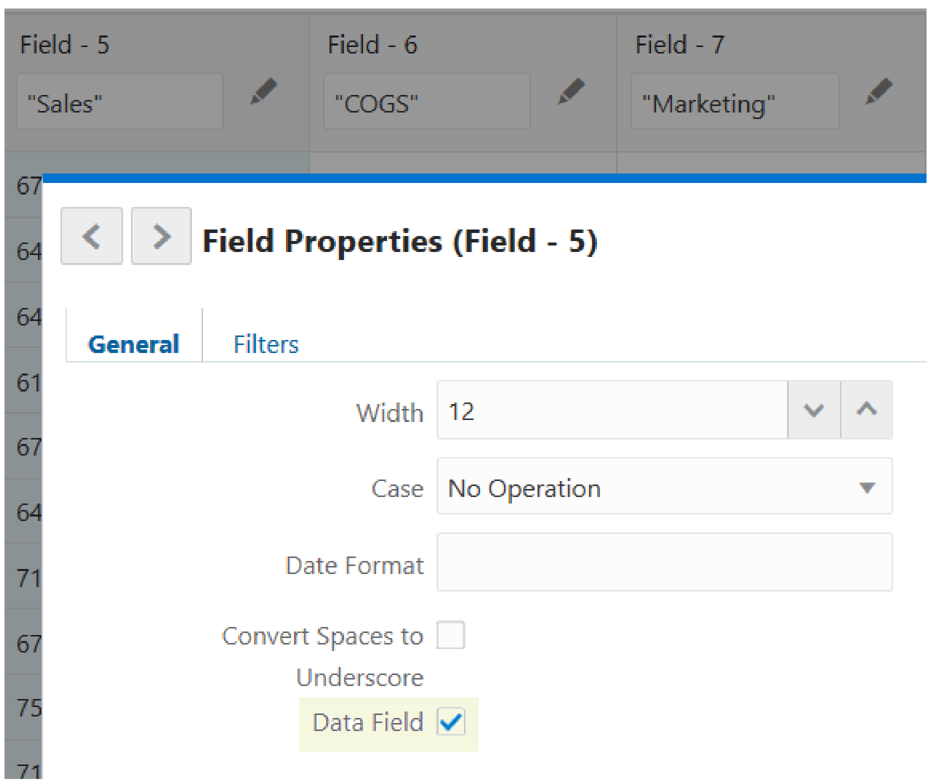
New Rule

Back Proceed

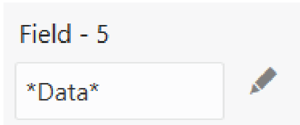
Preview Data

Column0	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Col
100-10	New York	Jan	Actual	678	271	94	51	0
100-10	New York	Jan	Budget	640	260	80	40	#M
100-10	New York	Feb	Actual	645	258	90	51	1
100-10	New York	Feb	Budget	610	240	80	40	#M
100-10	New York	Mar	Actual	675	270	94	51	1
100-10	New York	Mar	Budget	640	250	80	40	#M
100-10	New York	Apr	Actual	712	284	99	53	0
100-10	New York	Apr	Budget	670	270	80	40	#M
100-10	New York	May	Actual	756	302	105	53	1

The following example shows how to indicate that a field is a data field.

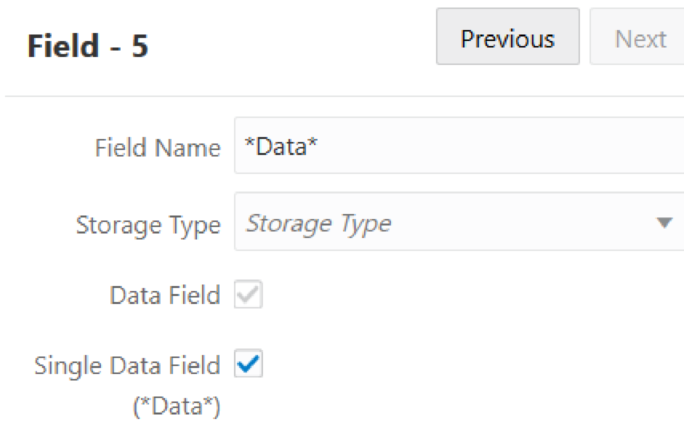


If all fields are dimensions followed by only a single data field, you must mark the data field as a single data field. The following field is how a single data field should look in the rule editor:



To set the field as a single data field,

1. Click the edit field/member selection tool next to the field. 
2. Select **Single Data Field (*Data*)**



3. Click OK. Verify and save the rule.

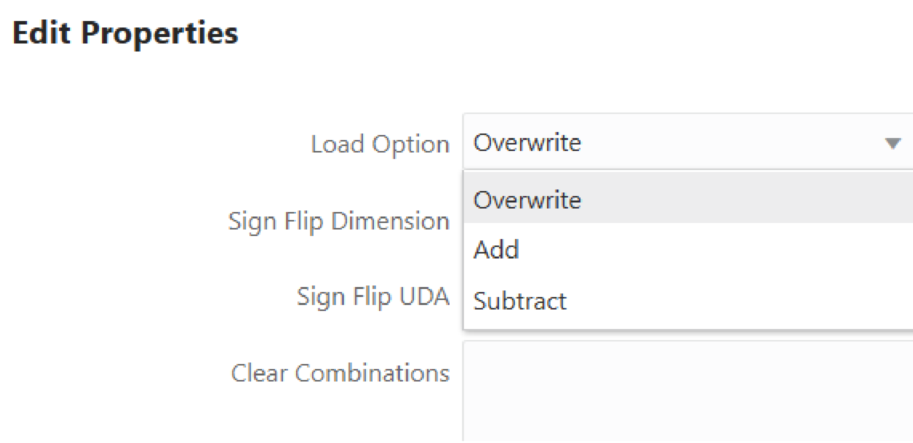
Adding to and Subtracting from Existing Values

If you want the Essbase data load to add to or subtract from existing cube values instead of overwriting them, use global properties in the Essbase data load rule to specify how newly loaded data values should affect existing values. For example, if you load weekly values, you can add them to create monthly values in the cube.

This section is for data load only. If you are performing a dimension build, skip this section.

By default, Essbase overwrites the existing values of the cube with the values of the source data, but you can set up the load rule to instead use incoming data values to add to or subtract from existing cube values.

1. In your data load rule, click **Global Properties**.
2. Select a load option: Overwrite (the default), Add, or Subtract.



3. Click OK. Verify and save the rule.

You can use these load options to perform Add, Subtract, or Overwrite operations in data-load rules when you want to make broad changes to all the values you are loading into the Essbase cube. You can only select one of these load options per data load, and the values are treated the same way for all data fields (all are either added, subtracted or overwritten).

Using this option makes recovery more difficult if the a problem occurs during data load, although Essbase lists the number of the last row committed in the application log.

For block storage cubes, in the settings, set the Commit Rows transaction option to 0 to prevent difficult recoveries. This setting causes Essbase to view the entire load as a single transaction and to commit the data only when the data load is complete.

Using the **import data (aggregate storage)** MaxL statement, you can only add to and substract from existing values in aggregate storage cubes only.

Extracting Source Data Using Column Store Options

You can load data to the Essbase cube using column-level options in the data load rule, to extract the source data the way you want it.

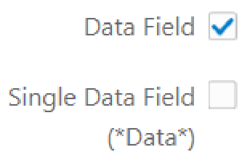
As described in Adding to and Subtracting from Existing Values, you can use load options to perform Add, Subtract, or Overwrite operations in data-load rules when you want to make broad changes to all the values you are loading into the cube. You can only select one of these load options per data load, and the values are treated the same way for all data fields (all are either added, subtracted or overwritten).

If you are loading to a large Essbase cube with many upper level members, you may need to perform more specific rule operations at the level of data columns, using SUM/MIN/MAX/COUNT. These operations help you drill through to the source data (from the load rule) to extract the source data in a more specific way.

As an example use for these operations, consider a retail business with credit card transactions stored in a relational system. You don’t want to load all transactional data into Essbase, but you want to load the sum of all transactions at the end of each day.

To select a column level option in a data load rule,

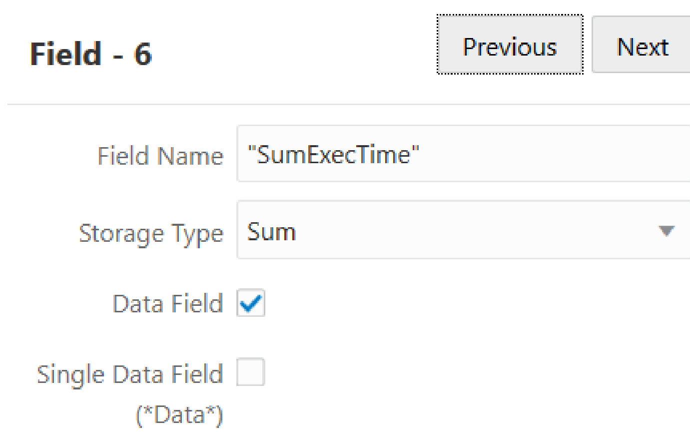
1. Click the edit field/member selection tool next to the rule field for the data column you want to customize. 
2. Ensure the field is marked as a data field.



3. Select a **Storage Type** option.

   Available options:

   - Minimum - stores the minimum value of the incoming data, including a comparison with existing cube data.
   - Maximum - stores the maximum value of the incoming data, including a comparison with existing cube data.
   - Sum - behaves the same as the Add global option. Adds the incoming data to existing cube data.
   - Count - stores the count of values present in the incoming data.



4. Click OK. Verify and save the rule.

Clearing Existing Data Values

When records in the source data will be added to or subtracted from existing Essbase data values, you may need to use to clear some regions of data first to ensure accuracy. You can use the global properties of the data load rule clear selected values.

This section is for data load only. If you are performing a dimension build, skip this section.

You can clear selected existing data values from the cube before you load new values. By default, Essbase overwrites the existing values of the cube with the new values of the source data. If you are adding and subtracting data values, however, Essbase adds or subtracts the new data values to and from the existing values.

Before adding or subtracting new values, make sure that the existing values are correct. Before loading the first set of values into the cube, make sure that there is no existing value.

For example, assume that the Sales figures for January are calculated by adding the values for each week in January:

```
January Sales = Week 1 Sales + Week 2 Sales + Week 3 Sales + Week 4 Sales
```

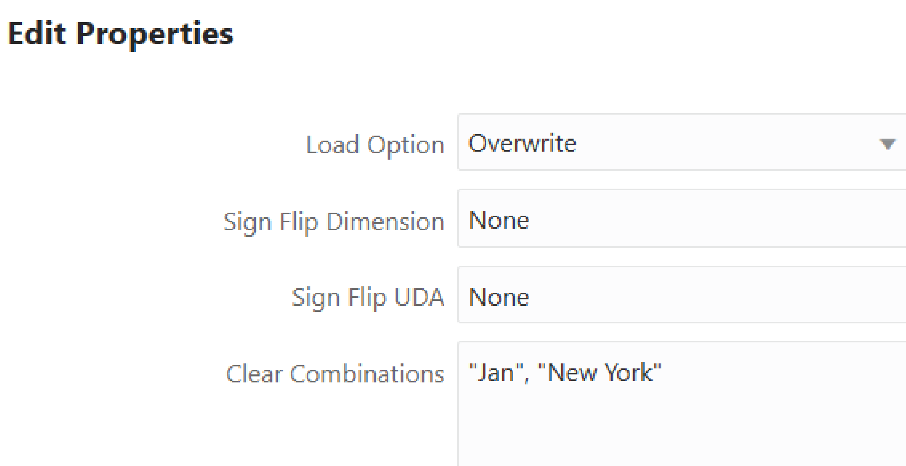
When you load Week 1 Sales, clear the cube value for January Monthly Sales. If there is an existing value, Essbase performs the following calculation:

```
January Sales = Existing Value + Week 1 Sales + Week 2 Sales + Week 3 Sales +
Week 4 Sales
```

You can also clear data from fields that are not part of the data load. For example, if a source contains data for January, February, and March, and you want to load only the March data, you can clear January and February data.

To clear before loading,

1. In your data load rule, click **Global Properties**.
2. In the **Clear Combinations** box, enter the member combinations to clear. For example, `"Jan","New York"`.



3. Click OK. Verify and save the rule.

Replacing All Data

For data slices in aggregate storage (ASO) cubes that can be reloaded in their entirety while maintaining low latency, Essbase can replace values using the contents of an incremental data load buffer.

This topic applies to loading data into an aggregate storage cube only. If you are loading data into a block storage cube or performing a dimension build, skip this topic.

In an aggregate storage cube, Essbase can remove all of the data in the cube or all of the data in each incremental data slice in a cube, and replace the data with the contents of a specified data load buffer. This functionality is useful when working with data sets that are small enough to reload completely, or when working with data that can be separated into large, static data sets that are never updated and small, volatile data sets in which you need to track changes.

To replace all data, see Replace Data Using Incremental Data Slice Contents.

Scaling Data Values

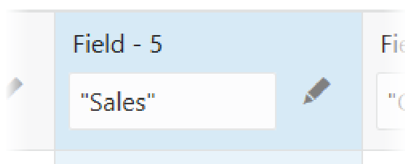
When data values in the source data are not of the same decimal scale as data values in the Essbase cube, use field properties in the data load rule to set a scaling factor for the incoming data values.

This section is for data load only. If you are performing a dimension build, skip this section.

For example, assume the real value of sales is $5,460. If the Sales source data tracks the values in hundreds, the value is 54.6. If the Essbase cube tracks the real value, you must multiply the value coming in from the Sales source data (54.6) by 100 to have the value display correctly in the Essbase cube (as 5460).

To scale the values,

1. Select a data field.



2. Click **Field Properties**.
3. Click **Scale**, and select a scaling factor.

4. Click OK. Verify and save the rule.

Flipping Field Signs

When records in source data contain negative values that need to be positive in the Essbase cube, or positive values that need to be negative, you can invert the signs. One case where sign flip can be useful is when loading data from double-entry accounting systems.

This section is for data load only. If you are performing a dimension build, skip this section.

You can reverse, or flip, the value of a data field by flipping its sign. Sign flips are based on the UDAs of the outline. When loading data into the accounts dimension, for example, you can specify that any record whose accounts member has a UDA of Expense change from a plus sign to a minus sign.

To flip signs,

1. In your data load rule, click **Global Properties**.
2. In **Sign Flip Dimension**, enter the dimension for which you want to flip signs.
3. In **Sign Flip UDA**, enter the name of the user defined attribute.

4. Click OK. Verify and save the rule.

See Also

Creating UDAs

Validate Rules

Essbase validates dimension build rules and data load rules, checking whether all members and dimensions in the rule are mapped to the associated cube outline. Learn what is required to ensure load rules are valid.

Validation cannot ensure that the source data loads properly.

To validate, click **Verify** in the rule editor.

If the rule is correct, you can perform a data load or dimension build.

If the rule is not valid, see the appropriate topic for each rule type:

- Data load: Requirements for Valid Data Load Rule Files
- Dimension build: Requirements for Valid Dimension Build Rule Files

Requirements for Valid Data Load Rule Files

For a data load rule file to validate, all of the following questions must be answered “yes.”

- Is the rule file associated with the correct outline?
- Does each record in the data source contain only one member from each dimension?

  See Items in a Source of Data.
- Are all member and dimension names spelled correctly?
- Are all members enclosed in quotation marks if they contain spaces, numbers, or file delimiters? Are there extra delimiters in the source?

  See Valid Member Fields.
- Is the member that each data field maps to spelled correctly in the rules file?

  See Changing Field Names.
- Are the file delimiters correctly placed?

  See Valid Delimiters.
- Is the member in the field name a valid member?

  See Mapping Fields.
- Is the dimension name used in only one field (for example, not in a field name and the header)?

  You can map a single data value to only one set of members.

- Is only one field defined as a data field?

  See Defining Columns as Data Fields.
- Is the UDA used for sign flipping in the associated outline?

  See Flipping Field Signs.
- If a header is used in the rule, does it contain only valid members that are in the dimensions of the outline?
- Do field names contain references only to valid members that are in the dimensions of the outline?

Requirements for Valid Dimension Build Rule Files

For a dimension build rule file to validate, all of the following questions must be answered "yes."

- Is the rule file associated with the correct outline?
- Are all member and dimension names spelled correctly?
- Are all members enclosed in quotation marks if they contain spaces, numbers, or file delimiters? Are there extra delimiters in the source?

  See Valid Member Fields.
- Are the reference numbers sequential?

  See Set Dimension Build Field Type Information.
- Are there no repeated generations? Are all generations set to a value of 1 or higher (0 is not not allowed)? Generations cannot be skipped (if there is a generation 2 and 4, there must be a 3). In Regular rules, they must also be in sequential order.

  See Set Dimension Build Field Type Information.
- Is each field type valid for the build method?

  Set Dimension Build Field Type Information
- Are all the fields in correct order?

  See Set Dimension Build Field Type Information.
- Does each child field have a parent field?
- Do all dimension names exist in the outline or the rule file?
- Are any dimensions specified in both the header record in the rules file and the header record in the data source?

  Dimensions can be specified in either the header in the rule file or the header in the data source, but not in both.

  See Handle Header Records.

Define Rules that Query External Sources

Using SQL queries, you can pull or push data to Essbase from external sources of data. You need to configure Essbase to access the external sources.

- Pull and Push Methods of Loading External Data into Essbase
- Overview of SQL Properties in Rules

- Connection Strings for SQL-Based Connectivity
- Access External Data Using a Connection and Datasource
- Access Oracle Database Using Oracle Call Interface
- Access Data Using ODBC Connectivity
- Access Data Using JDBC Connectivity
- Stream from a Remote Database

Notes

1. The rule file must exist in the cube directory for the relevant Essbase cube.
2. The database query used to load data or build dimensions must have the same dimensionality as the columns in the rule. (In other words, the order of dimensions in the rule file must match the order of dimensions in the SQL query).
3. Only use SQL functions that are supported by JDBC. ODBC scalar functions are not supported in CLI.

Pull and Push Methods of Loading External Data into Essbase

When building dimensions and loading data to Essbase from SQL-based sources, you can use either a “pull” or a “push” data flow to move the source data into the Essbase cube.

These examples are relevant for external sources you would access using SQL queries, but you can also load from a file on the server or a path accessible to the server.

Pull Method

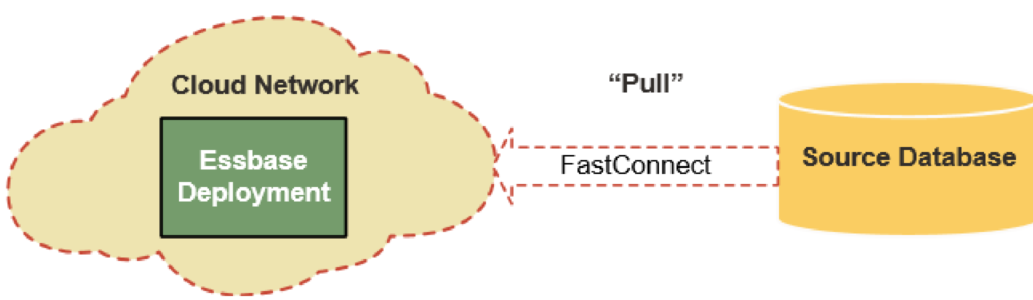
When an external source database is network-accessible to Essbase, then data loads and dimension builds can pull SQL query results directly from the source.

Supported pull methods include

- Connection/Datasource abstractions you define in Essbase
- SQL Properties you define in the load rule

Example Use Case

Essbase is deployed on a private network (VCN). A secure VPN->VCN connection is provided through a dynamic routing gateway (DRG), using a connection such as FastConnect to extend the private cloud network securely to the source database.



1. Define the connection and Datasource in Essbase web interface
2. Design a load rule over the connection and Datasource

3. Run the data load or dimension build job from Essbase

OR

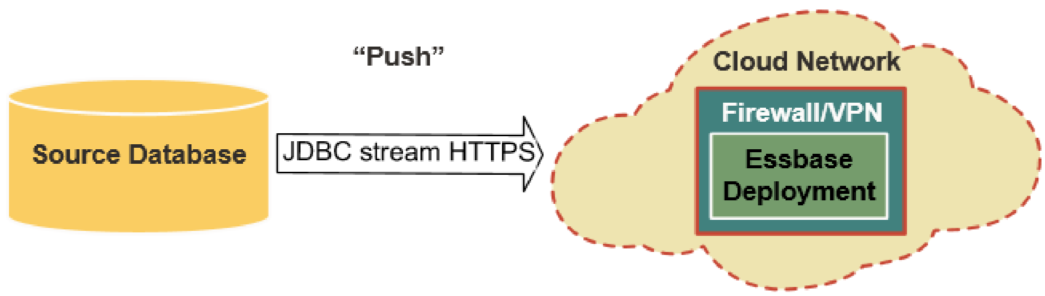
1. Design a load rule with direct connection details to the source database
2. Run the data load or dimension build job from Essbase

Push Method

An external source database can push SQL query results to Essbase. Pushing data into Essbase is helpful in cases when the database is not network accessible to Essbase. With this method, you use Essbase REST API or CLI to create a local JDBC connection, and also to initiate the data stream. This streaming connection is secured by the HTTPS protocol.

Example Use Case

Essbase is deployed within a firewall/VPN.



1. Use CLI createlocalconnection command to make a local JDBC connection to the source database
2. Use CLI dimbuild or dataload command, with `-stream` and `-conn` options, to initiate the data flow over the JDBC connection. For more information, see Build Dimensions and Load Data by Streaming from a Remote Database.

Overview of SQL Properties in Rules

Design Essbase load rules with SQL Properties to extract data from a relational database using a SQL query, when your source of data requires connectivity to an external system such as an RDBMS.

SQL-based loads extract data from a relational database using a SQL query, and load it into Essbase using your defined rule.

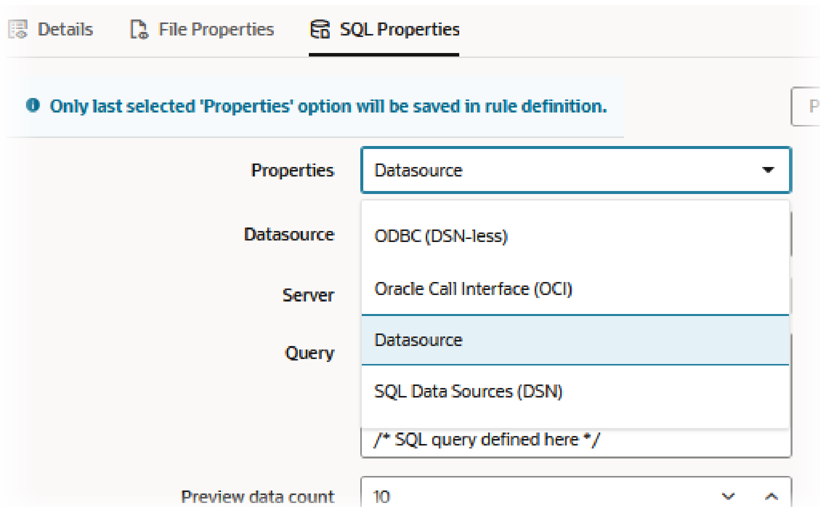
Before you start, identify whether your access to the external source is based on:

- a predefined connection and Datasource established in Essbase
- a connection string to your external source of data
- ODBC drivers configured on Linux or on Windows

Once you know the above information, log in to the Essbase web interface and start here to define SQL-based connectivity for your load rule.

1. Open an existing rule or create a new one (see Access and Edit Rules, Create New Dimension Build Rule, or Create New Data Load Rule).
2. Click **Source Properties**.

3. Click **SQL Properties**.
4. Identify which of the following Properties categories best applies to your source data.



If your connectivity has been predefined using a connection and Datasource established in Essbase, choose **Datasource**, and see Access External Data Using a Connection and Datasource.

If your connectivity depends on ODBC drivers configured on the Essbase server, choose **ODBC (DSN-less)** or **SQL Data Sources (DSN)**, and see Access Data Using ODBC Connectivity.

If your connectivity depends on a connection string that includes `OCI`, choose **Oracle Call Interface (OCI)**, and see Access Oracle Database Using Oracle Call Interface.

Connection Strings for SQL-Based Connectivity

To design Essbase load rules that access external sources of data using SQL, you sometimes need to have a connection string. Supported formats for different sources are listed here.

The available OCI and DSN-less connection string types are listed, with syntax and examples.

In the examples, `DBName` is the name of the external source database containing tables and data.

Oracle Call Interface (OCI)

Syntax: `DatabaseServerName:PortNumber/ServiceName`

Example: `OracleServer123.example.com:1521/orcl.example.com`

Oracle Database (DSN-less)

Syntax (SID): `oracle://HostName:PortNumber/SID`

Example (SID): `oracle://somedb99:1234/ORCL`

Syntax (ServiceName): `ORACLESERVICE:oracle://HostName:PortNumber/Servicename`

Example (ServiceName): `ORACLESERVICE:oracle://somedb99:1234/esscs.host1.oraclecloud.com`

Microsoft SQL Server (DSN-less)

Syntax: `sqlserver://HostName:1433:DBName`

Example: `sqlserver://myMSSQLHost:1433:myDbName`

DB2 (DSN-less)

Syntax: `db2://HostName:PortNumber:DBName`

Example: `db2://myDB2Host:1234:myDbName`

MySQL (DSN-less)

Syntax: `mysql://HostName:3306:DBName`

Example: `mysql://someHostName:3306:myDbName`

Teradata (DSN-less)

Syntax: `teradata://host:port/DBName`

Example: `teradata://192.0.2.110:1025/myDBName`

Access External Data Using a Connection and Datasource

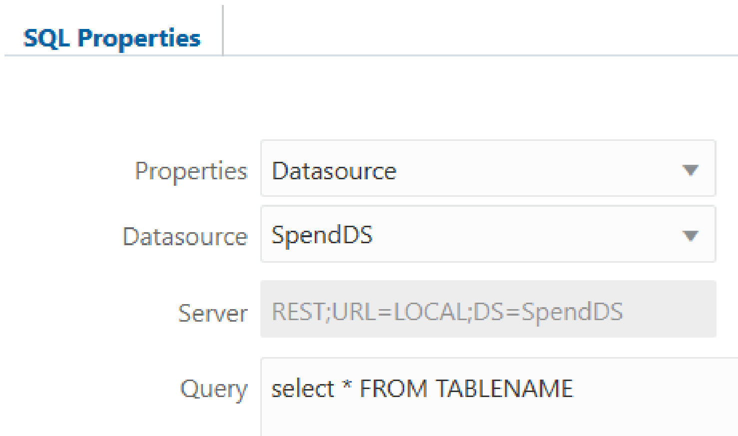
When you need to load data or dimensions to an Essbase cube using external source data, an efficient option is to use connections and Datasources, which you create and save as reusable abstractions in Essbase.

Connections and Datasources are effective when you have network connectivity between an external source of data and Essbase. Once you define them, they enable Essbase to easily "pull" data from the external source.

A **connection** stores information about an external server and the login credentials that are required to access it.

A **Datasource**, used in conjunction with the connection, helps you manage data flow into and out of Essbase. You define one Datasource to represent each external source of data, whether that is a relational system, a table, a file, or another cube.

Here is an example of the **SQL Properties** for a load rule that depends on a connection and Datasource pre-defined in Essbase.



See Work With Sources to get started.

Access Oracle Database Using Oracle Call Interface

If you use applications developed with Oracle Call Interface (OCI), you can connect Essbase load rules to Oracle Database using either an OCI connection string, or a net service name defined in a `tnsnames.ora` configuration file on the Essbase Server.

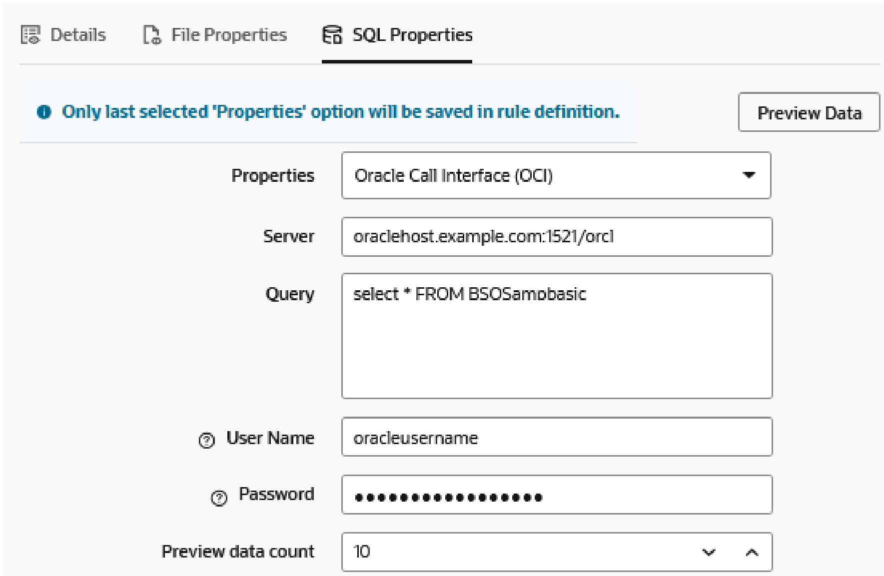
Rule Properties with Oracle Call Interface Connection String

To connect an Essbase load rule to Oracle Database using an OCI connection string,

1. In the Essbase web interface, open an existing rule or create a new one (see Access and Edit Rules, Create New Dimension Build Rule, or Create New Data Load Rule).
2. Click **Source Properties**.

3. Click **SQL Properties**.
4. From **Properties**, select **Oracle Call Interface (OCI)**.
5. In the **Server** field, enter the Oracle Database connection string, in the format *<host>*:*<port>*/*<service_name>*.
6. In the **Query** field, enter a SQL query that accesses the data you need from Oracle Database.
7. In the **User Name** and **Password** fields, provide the credentials required to access the Oracle Database schema. This information is used for previewing data, but will not be saved in the rule.



8. Click **Preview** to ensure that the rule can connect and retrieve data, then save and close the rule.

Load Rule with Oracle Call Interface Predefined in Configuration File

To connect an Essbase load rule to Oracle Database using a net_service_name defined in a configuration file on the Essbase Server,

1. On the Essbase Server, create a configuration file (for example, named `tnsnames.ora`) in `<Oracle Home>/network/admin`.

   If the configuration file already exists, ensure it has a valid entry for connecting to Oracle Database using Oracle Call Interface (OCI). Otherwise, configure it now. In the following example,

```
OCISQL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = oraclehost.example.com)(PORT =
1521))
    )
    (CONNECT_DATA =
      (service_name = orcl.example.com)
      (SERVER = DEDICATED)
    )
  )
```

   - The net_service_name is `OCISQL`. Take note of this name, which you will need for the next steps.
   - The host is `oraclehost.example.com`.
   - The port is `1521`.
   - The service name is `orcl.example.com`.

2. In the Essbase web interface, open an existing data load or dimension build rule, or create a new one (see Access and Edit Rules, Create New Dimension Build Rule, or Create New Data Load Rule).

3. Click **Source Properties**.

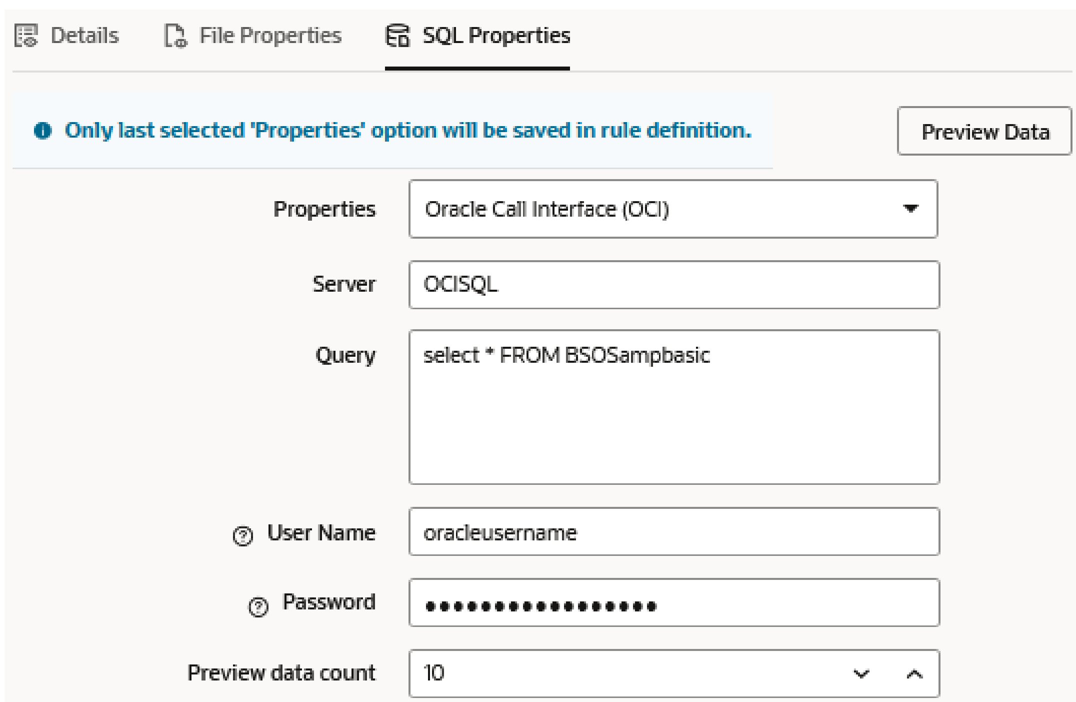
4. Click **SQL Properties**.

5. From **Properties**, select **Oracle Call Interface (OCI)**.

6. In the **Server** field, enter the net_service_name used in the `tnsnames.ora` configuration file. Based on the example given in step 1, enter `OCISQL`.

7. In the **Query** field, enter a SQL query that accesses the data you need from Oracle Database.

8. In the **User Name** and **Password** fields, provide the credentials required to access the Oracle Database schema. This information is used for previewing data, but will not be saved in the rule.



9. Click **Preview** to ensure that the rule can connect and retrieve data, then save and close the rule.

Troubleshooting Connectivity

On Windows, previewing data or running a dimension build or data load with a rule that uses an Oracle Call Interface (OCI) connection string may fail with the following error message:

```
Attempt to connect to OCI failed. [ORA-12638: Credential retrieval failed].
```

The cause of this issue is that the version 19c Oracle Database server and the 19c client (in this case the client is where Essbase Server is installed) has no setting, or has the following setting in configured in `tnsnames.ora` (or `sqlnet.ora`): SQLNET.AUTHENTICATION_SERVICES=(NTS)

The solution is to configure SQLNET.AUTHENTICATION_SERVICES=(NONE) on the Essbase Server machine.

1. In `<Oracle Home>/network/admin`, locate the configuration file.
2. Open the file and add `SQLNET.AUTHENTICATION_SERVICES=(NONE)`
3. Restart the Essbase application.

> **Note:**
>
> Refer to usage notes in the SQLNET.AUTHENTICATION_SERVICES documentation for Oracle Database 19c.

Access Data Using ODBC Connectivity

If your source data is network accessible and configured using ODBC, you can configure Essbase to work with the source data.

- Configure the ODBC Source of Data (Linux)
- Configure the ODBC Source of Data (Windows)
- Customize the Rule File for an ODBC Source of Data
- Special SQL Configuration Options

Configure the ODBC Source of Data (Linux)

To make ODBC sources of data work with Essbase on Linux, you primarily configure `odbc.ini` in the Essbase server domain.

- Prerequisites
- Configure the ODBC Source of Data
- Teradata Only - Set Environment Variables
- Teradata Only - Configure odbcinst.ini

Prerequisites

Before you begin, make note of the location of these files on the Essbase server domain:

- For Linux only, `odbc.ini` - In Essbase 21c, `odbc.ini` is located in:

  ```
  <Domain Root>/<Domain Name>/config/fmwconfig/essconfig/core
  ```

- ODBC driver libraries - located in

  ```
  <Essbase Product Home>/modules/
  ```

If you are configuring Teradata as a source, do the following:

1. Download the Teradata Tools and Utilities client from https://downloads.teradata.com/
2. Install the Teradata client onto your Essbase server machine.

Configure the ODBC Source of Data

To configure your ODBC source of data, you start by editing `odbc.ini` (or `.odbc.ini`, if hidden). This is the ODBC data source configuration information file located on the Essbase server domain.

1. Open `odbc.ini` in a text editor.
2. In the section labeled

   ```
   [ODBC Data Sources]
   ```

   add/identify your source using the following syntax:

   ```
   <DSN name>=<Driver name>
   ```

   where

   - DSN name = A name you choose to identify the ODBC connection to the source of data
   - Driver name = The name of the ODBC driver to use for this connection

   For example, to add a DB2 source, add:

   ```
   [ODBC Data Sources]
   TBC_DB2=Oracle 7.1 DB2 Wire Protocol
   ```

   In this release, Oracle Database uses driver version 8.0, and all other RDBMS use driver versions 7.1.

   The following example entries include DB2, SQL Server, MySQL, Oracle Database, and Teradata. For Teradata, see the last two lines (the syntax varies from the other sources).

   ```
   [ODBC Data Sources]
   TBC_DB2=Oracle 7.1 DB2 Wire Protocol
   TBC_SQL=Oracle 7.1 SQL Server Wire Protocol
   TBC_MYSQL=Oracle 7.1 MySQL Wire Protocol
   TBC_ORA=Oracle 8.0 Oracle Wire Protocol
   TBC_ORA_DBAAS=Oracle 8.0 Oracle Wire Protocol
   TBC_ORA_ADW=Oracle 8.0 Oracle Wire Protocol
   Teradata=tdataodbc_sb64.so
   $TELAPI$10.xx.xxx.211=tdataodbc_sb64.so
   ```

3. For each DSN name you added in `[ODBC Data Sources]` section, add a section for it specifying the driver path (required), description (optional), and host details (required). The following example includes sections for various sources.

   ```
   [TBC_DB2]
   #Example for DB2
   Driver=/scratch/username/oracle_home/essbase/modules/
   oracle.essbase.datadirect.odbc/7.1.6/lib/Ardb227.so
   Description=Oracle 7.1 DB2 Wire Protocol Driver
   ```

```
Database=TBC
IpAddress=myhost.example.com
TcpPort=50000

[TBC_SQL]
#Example for SQL Server
Driver=/scratch/username/oracle_home/essbase/modules/
oracle.essbase.datadirect.odbc/7.1.6/lib/ARsqls27.so
Description=Oracle 7.1 SQL Server Wire Protocol
HostName=myhost.example.com
PortNumber=1433
Database=TBC
EnableQuotedIdentifiers=1

[TBC_MYSQL]
#Example for MySQL
Driver=/scratch/username/oracle_home/essbase/modules/
oracle.essbase.datadirect.odbc/7.1.6/lib/ARmysql27.so
Description=Oracle 7.1 MySQL Wire Protocol
Database=tbc
HostName=myhost.example.com
PortNumber=3306

[TBC_ORA]
#Example for standalone Oracle Database
Driver=/scratch/username/oracle_home/essbase/modules/
oracle.essbase.datadirect.odbc/8.0.2/lib/ARora28.so
Description=Oracle 8.0 Oracle Wire Protocol
EnableNcharSupport=1
EncryptionLevel=1
DataIntegrityLevel=1
PortNumber=1521
SID=orcl
HostName=myhost.example.com

[TBC_ORA_DBAAS]
#Example for Oracle DBaaS
Driver=/scratch/username/oracle_home/essbase/modules/
oracle.essbase.datadirect.odbc/8.0.2/lib/ARora28.so
Description=Oracle 8.0 Oracle Wire Protocol
DataIntegrityLevel=1
EncryptionLevel=1
EncryptionTypes=AES256
DataIntegrityTypes=SHA1
HostName=192.0.2.20
PortNumber=1521
EnableNcharSupport=1
ServiceName=ORCL.esscs.mydb.myhost.example.com

[TBC_ORA_ADW]
#Example for Oracle Database with TLS/SSL security
Driver=/scratch/username/oracle_home/essbase/modules/
oracle.essbase.datadirect.odbc/8.0.2/lib/ARora28.so
Description=Oracle 8.0 Oracle Wire Protocol
EnableNcharSupport=1
EncryptionLevel=1
```

```
DataIntegrityLevel=1
EncryptionMethod=1
AuthenticationMethod=1
CryptoProtocolVersion=TLSv1.2,TLSv1.1,TLSv1
ValidateServerCertificate=1
TNSNamesFile=/scratch/username/oracle_home/network/admin/tbc_ora_adw/
tnsnames.ora
ServerName=TBC_ORA_ADW
Truststore=/scratch/username/oracle_home/network/admin/tbc_ora_adw/
cwallet.sso
Keystore=/scratch/username/oracle_home/network/admin/tbc_ora_adw/
cwallet.sso

[Teradata]
Driver=/opt/teradata/client/16.20/lib64/tdataodbc_sb64.so
Description=NCR 3600 running Teradata V2R6.2
DBCName=192.xx.xx.110
astUser=
Username=xxxxxx
Password=xxxxxx
Database=TBC
DefaultDatabase=TBC
NoScan=no
MechanismName=TD2

[$TELAPI$192.0.2.110]
Driver=/opt/teradata/client/16.20/lib64/tdataodbc_sb64.so
Description=NCR 3600 running Teradata V2R6.2
DBCName=192.0.2.110
astUser=
Username=xxxxxx
Password=xxxxxx
Database=TBC
DefaultDatabase=TBC
NoScan=no
MechanismName=TD2
```

Note:

The TBC_ORA_ADW example is configured for Autonomous Data Warehouse with TLS/SSL security enabled. For any TLS/SSL configurations (including Autonomous Data Warehouse), you must provide the path to the keystore and wallet as part of the configuration. The keystore and wallet (and TNS names configuration file) are located in `<Oracle Home>/network/admin`. If you use Autonomous Data Warehouse, you must download the wallet from Autonomous Data Warehouse to `<Oracle Home>/network/admin` on the Essbase server.

Note:

For Teradata, `DBCName` is not the same as `DBName` used in connection strings.

> **Note:**
>
> For SQL Server, be sure to enable quoted identifiers as shown in the example.

Teradata Only - Set Environment Variables

If you are configuring Teradata as a source, you additionally need to edit the Essbase server configuration file, `<Domain Root>/<Domain Name>/config/fmwconfig/essconfig/essbase/essbase.cfg`.

1. Navigate to `essbase.cfg` and open it in a text editor.
2. Add LD_LIBRARY_PATH variables to specify the loation of Teradata client libraries:

```
env:LD_LIBRARY_PATH /opt/teradata/client/16.20/bin
env:LD_LIBRARY_PATH /opt/teradata/client/16.20/lib64
env:LD_LIBRARY_PATH /opt/teradata/client/16.20/tbuild
env:LD_LIBRARY_PATH /opt/teradata/client/16.20/msg
env:LD_LIBRARY_PATH /opt/teradata/client/16.20/tbuild/tptapi
```

3. Add the following line to the end of `essbase.cfg`, to configure authentication:

```
env:LD_PRELOAD /usr/lib64/libgssapi_krb5.so:/usr/lib64/libkrb5.so:/usr/
lib64/libk5crypto.so:/usr/lib64/libkrb5support.so
```

4. Save `essbase.cfg`.
5. Stop and restart the Essbase server.
   a. Navigate to the domain tools directory, `<Domain Root>/<Domain Name>/esstools/bin`.
   b. Stop and restart the Essbase services by running `./stop.sh` and `./start.sh`, respectively.

Teradata Only - Configure odbcinst.ini

If you are configuring Teradata as a source, you additionally need to edit the `odbcinst.ini` file with information about the ODBC driver for Teradata.

1. Navigate to `<Oracle Home>/essbase/modules/oracle.essbase.datadirect.odbc/8.0.2/` and open `odbcinst.ini` in a text editor.
2. Add information about the Teradata ODBC driver:

```
Teradata=Installed
[Teradata]
Driver=/opt/teradata/client/16.20/odbc_64/lib/tdataodbc_sb64.so
APILevel=CORE
ConnectFunctions=YYY
DriverODBCVer=3.51
SQLLevel=1
```

3. Save `odbcinst.ini`.

Configure the ODBC Source of Data (Windows)

To make ODBC sources of data work with Essbase on Windows, you use the ODBC Data Source Administrator utility to configure external sources of data.

To configure Essbase on Windows to work with the ODBC drivers, use the following workflow.

- Prerequisites
- Configure the ODBC Source of Data (Oracle Database)
- Configure the ODBC Source of Data (Teradata)
- Configure the ODBC Source of Data (SQL Server)

Prerequisites

Before you begin,

- Make note of the location of the ODBC driver libraries on the Essbase server domain:
  - For Oracle Database:

    ```
    <Essbase Product Home>\modules\oracle.essbase.datadirect.odbc\8.0.2\drivers
    ```

  - For all other databases:

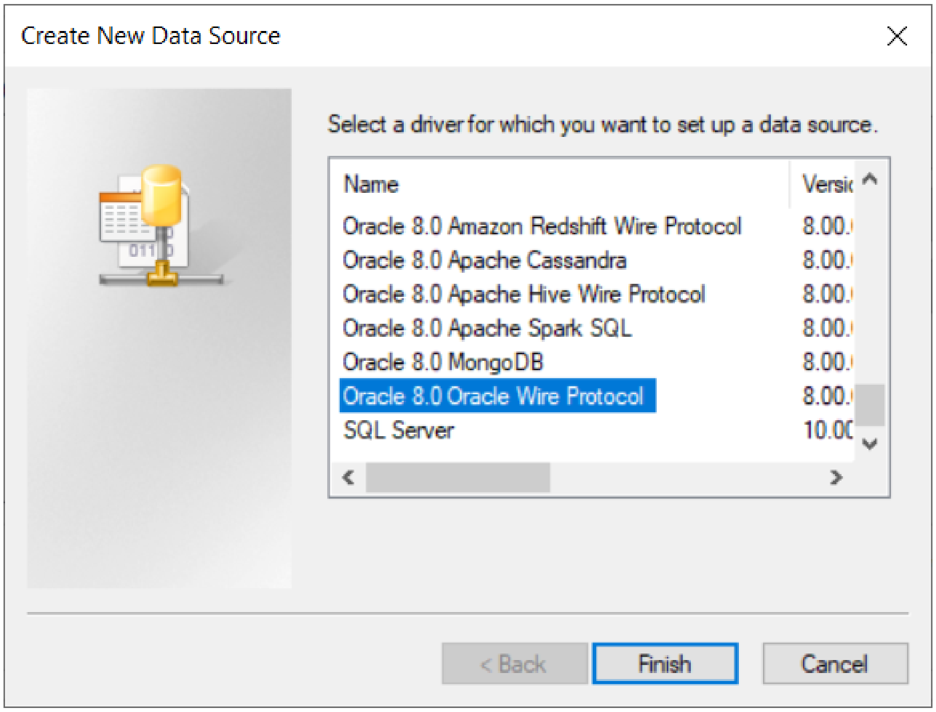
    ```
    <Essbase Product Home>\modules\oracle.essbase.datadirect.odbc\7.1.6\Drivers
    ```

- If you are configuring Teradata as a source, do the following:
  1. Download the Teradata Tools and Utilities client from https://downloads.teradata.com.
  2. Install the Teradata client onto your Essbase server machine.

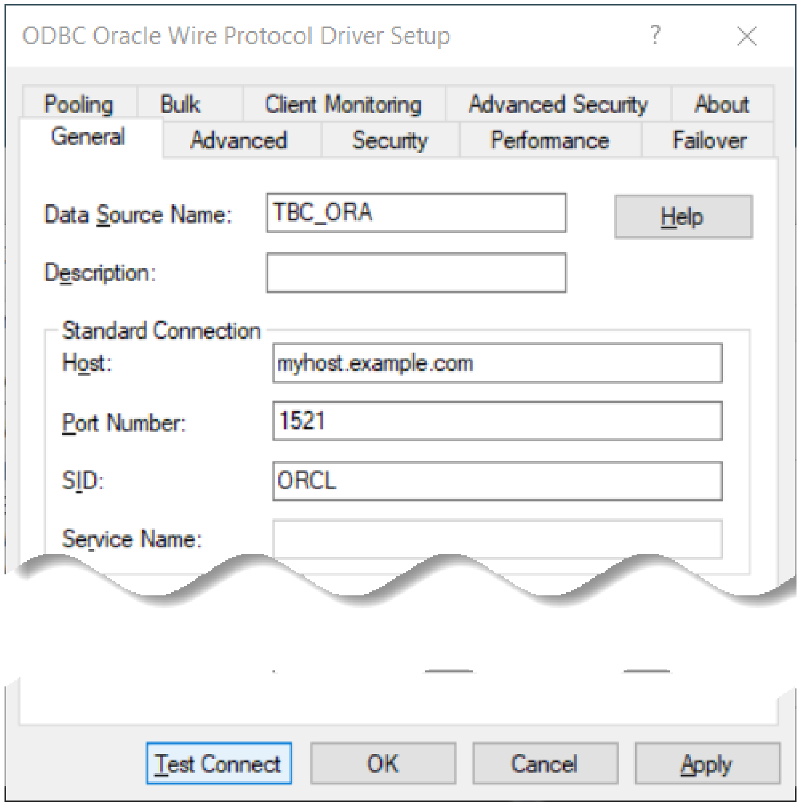
Configure the ODBC Source of Data (Oracle Database)

To configure Essbase on Windows to work with Oracle Database ODBC drivers, use the following workflow.

1. Click **Start**, click **Windows Administrative Tools**, and select **ODBC Data Sources (64-bit)**.
2. Click the **System DSN** tab if you will configure for all users of this instance of Essbase on Windows, or click **User DSN** to configure only for the current user.
3. Click **Add**, and select a driver to configure. For example, to add an ODBC driver for Oracle, select Oracle 8.0 Oracle Wire Protocol.

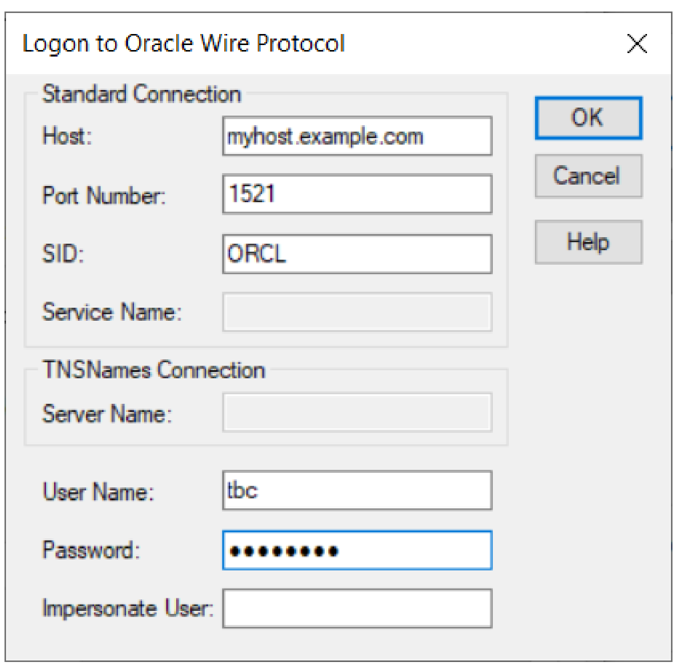


4. Click **Finish**.
5. In the driver setup dialog,
   a. Enter the Data Source Name (for example, TBC_ORA).
   b. Enter the name of the Oracle Database host machine (for example, myhost.example.com).
   c. Enter the port number of the Oracle Database (for example, 1521).
   d. Enter either one of the following:
      - the Oracle System ID (SID) that uniquely identifies the database (for example, ORCL)
      - the Oracle Service Name, if the Database is registered with a listener (for example, orcl.esscs.myhost.example.com)

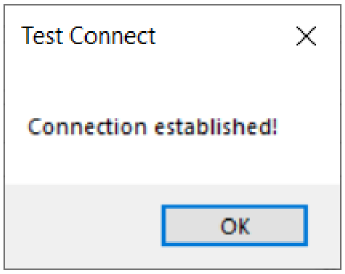


e. Click **Test Connect**.

6. In the Logon to <source> dialog, enter the user name and password of an authorized Oracle Database user, and click OK.

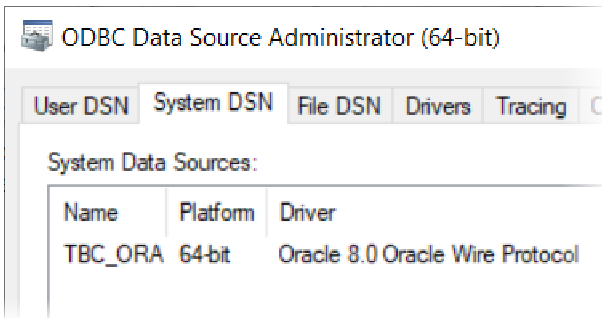


7. If the connection to Oracle Database was successful, a Connection established message displays.

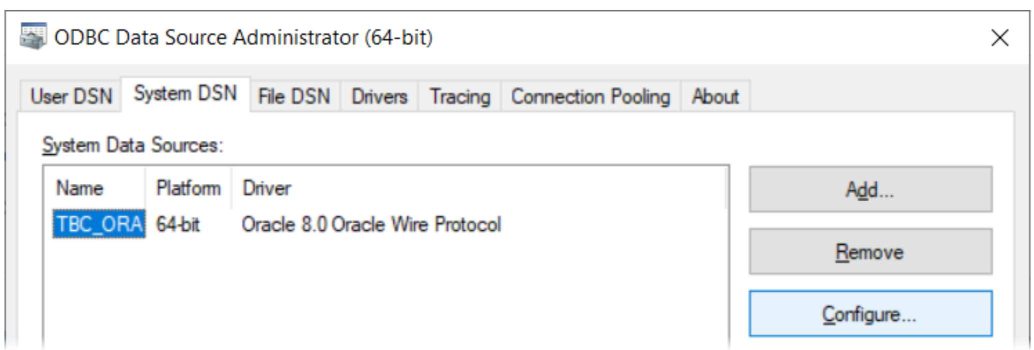


Click OK.

8. Click **Apply**, and click OK. The ODBC data source is configured and ready to use.



9. If the data source configuration needs to be modified,
   a. Select the data source name in System Data Sources.
   b. Click **Configure**.



   c. Make any needed changes, using only the General tab.

Configure the ODBC Source of Data (Teradata)

The following workflow is for configuring Teradata to work with Essbase as an ODBC data source on Windows.

1. Click **Start**, click **Windows Administrative Tools**, and select **ODBC Data Sources (64-bit)**.
2. Click the **System DSN** tab if you will configure for all users of this instance of Essbase on Windows, or click **User DSN** to configure only for the current user.

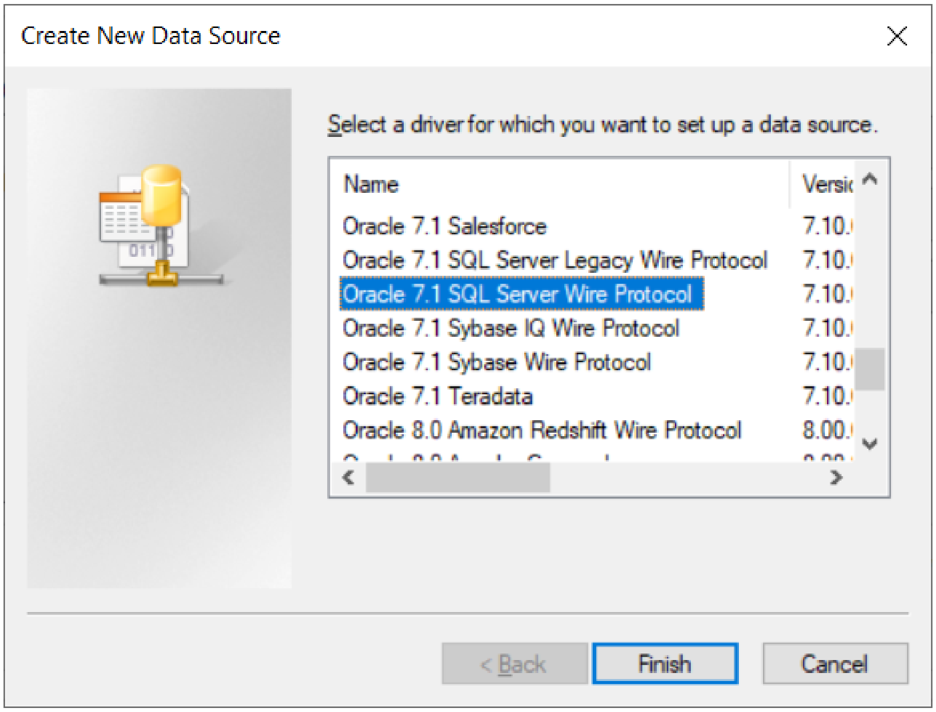
3. Click **Add**, and select a driver to configure. For example, to add an ODBC driver for Teradata, select Teradata Database ODBC Driver 16.20.
4. Click **Finish**.
5. In the driver setup dialog,
    a. Enter the Data Source Name (for example, TBC_TD).
    b. Enter the host name or IP address of the Teradata Database Server (for example, 10.xx.xx.211).
    c. Enter the user name and password of an authorized Teradata user.
    d. Enter the Default Database name (for example, TBC).
    e. Click **Test Connect**.
6. If the connection to Teradata was successful, a success message displays.

    Click OK.
7. Click **Apply**, and click OK. The ODBC data source is configured and ready to use.
8. If the data source configuration needs to be modified,
    a. Select the data source name in System Data Sources.
    b. Click **Configure**.
    c. Make any needed changes, using only the General tab.

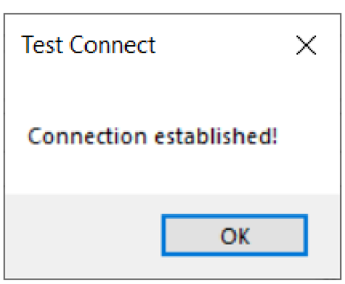
Configure the ODBC Source of Data (SQL Server)

To configure Essbase on Windows to work with SQL Server ODBC drivers, use the following workflow.

1. Click **Start**, click **Windows Administrative Tools**, and select **ODBC Data Sources (64-bit)**.
2. Click the **System DSN** tab if you will configure for all users of this instance of Essbase on Windows, or click **User DSN** to configure only for the current user.
3. Click **Add**, and select a driver to configure. For example, select Oracle 7.1 SQL Server Wire Protocol.



4. Click **Finish**.
5. In the driver setup dialog,
   a. Enter the Data Source Name (for example, TBC_SQL).
   b. Enter the name of the host machine (for example, myhost.example.com).
   c. Enter the port number of the SQL Server (for example, 1433).
   d. Enter the Database Name of the SQL Server (for example, TBC).
   e. Click the Advanced tab, and check the **Enable Quoted Identifiers** box.
   f. Click **Test Connect**.
6. If the connection to SQL Server was successful, a Connection established message displays.



   Click OK.
7. Click **Apply**, and click OK. The ODBC data source is configured and ready to use.
8. If the data source configuration needs to be modified,
   a. Select the data source name in System Data Sources.
   b. Click **Configure**.

c. Make any needed changes, using only the General tab.

Customize the Rule File for an ODBC Source of Data

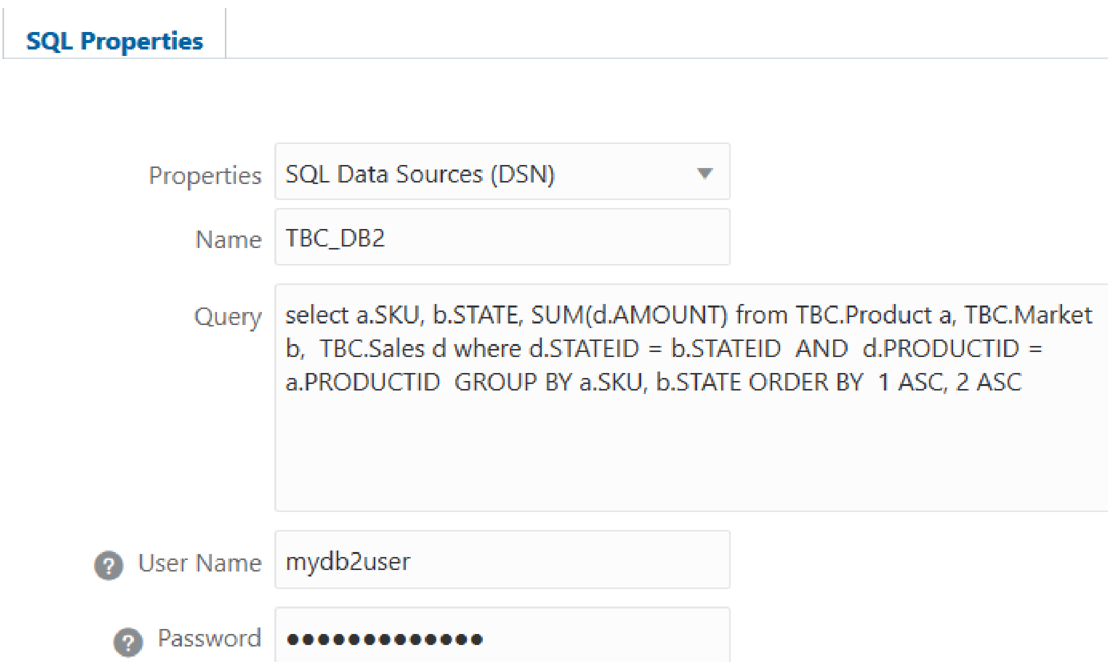
If your source data is network accessible and configured using ODBC, you can configure Essbase rules to access the source data using SQL query results.

- Example 1 - DB2
- Example 2 - Teradata

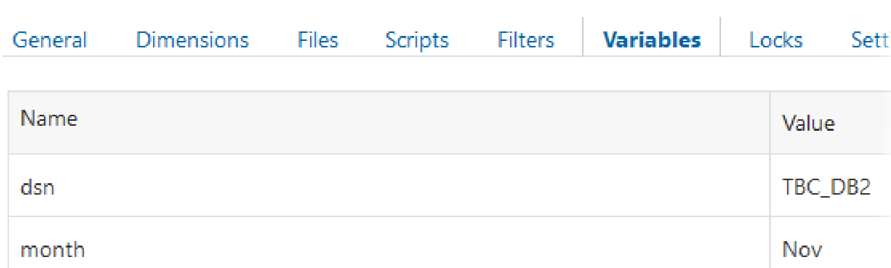
Example 1 - DB2

Here is an example of the **SQL Properties** for a load rule for accessing a relational source of data in DB2 that is *not* defined as Datasource in Essbase.

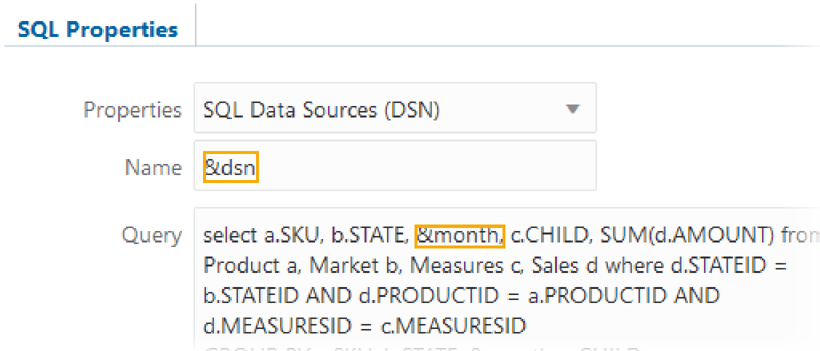
The **Name** field must match the DSN name defined in `odbc.ini` (Linux), or the Data Source Name configured in Windows. A SQL query you write pulls the data from DB2 to Essbase.



If you want to use one load rule for more than one ODBC source of data, you can use a substitution variable for the DSN name (and for elements in SQL clauses). For example, if the &dsn and &month variables are defined in the cube associated with the rule file,

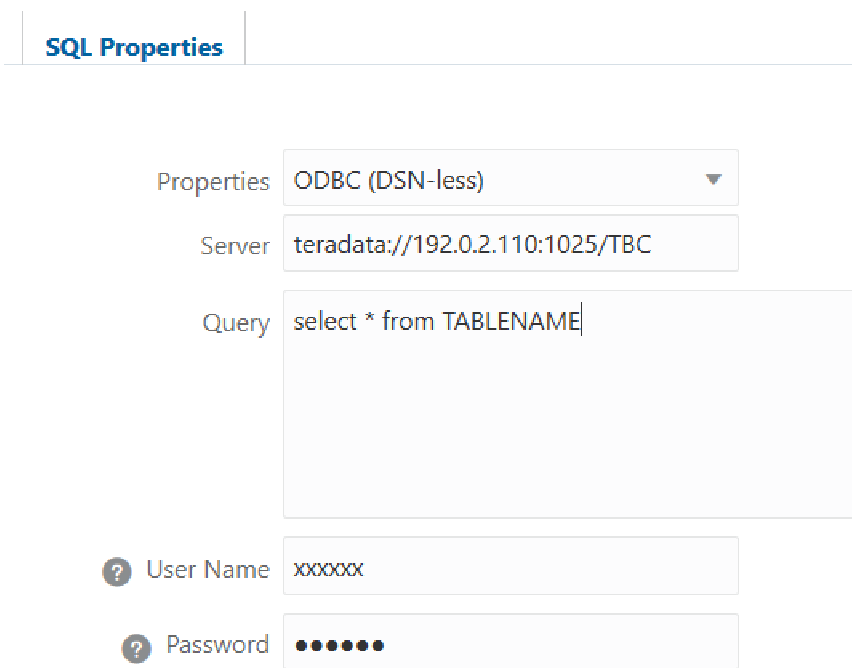


then the load rule can reference these variables, as shown in the following rule example:



Example 2 - Teradata

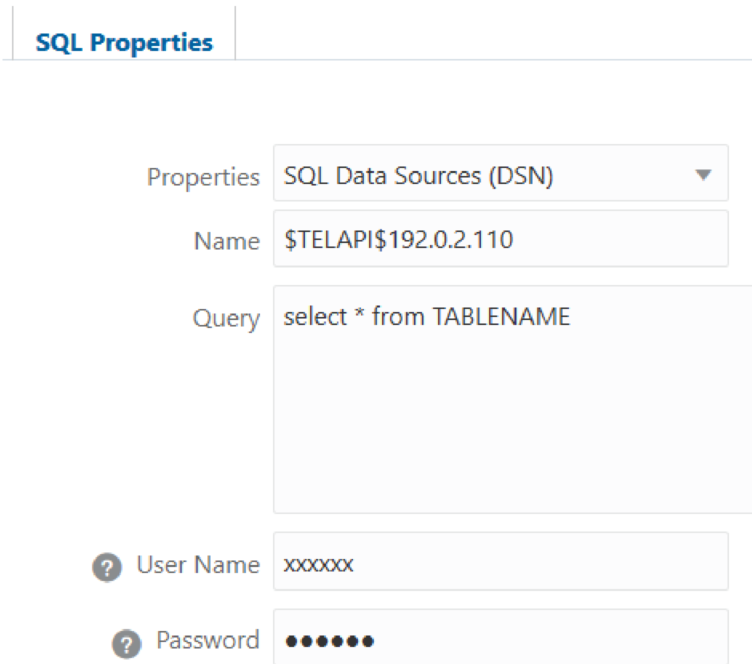
Here is an example of the **SQL Properties** for a load rule for accessing a relational source of data in Teradata.



The example above corresponds to this configuration in `odbc.ini` (on Linux):

```
[Teradata]
Driver=/opt/teradata/client/16.20/lib64/tdataodbc_sb64.so
Description=NCR 3600 running Teradata V2R6.2
DBCName=192.0.2.110
astUser=
Username=xxxxxx
Password=xxxxxx
Database=TBC
DefaultDatabase=TBC
NoScan=no
MechanismName=TD2
```

Here is another example of the **SQL Properties** for a load rule for accessing a relational source of data in Teradata.



The example above corresponds to this configuration in `odbc.ini` (on Linux):

```
[$TELAPI$192.0.2.110]
Driver=/opt/teradata/client/16.20/lib64/tdataodbc_sb64.so
Description=NCR 3600 running Teradata V2R6.2
DBCName=192.0.2.110
astUser=
Username=xxxxxx
Password=xxxxxx
Database=TBC
DefaultDatabase=TBC
NoScan=no
MechanismName=TD2
```

Special SQL Configuration Options

Configure `esssql.cfg` on the Essbase domain when you want to connect to a database using ODBC drivers not included by Oracle with Essbase, or when you want to change the default settings for any of the drivers that are distributed with Essbase.

- Use Case
- Change the SQL Configuration
- Sample esssql.cfg File
- esssql.cfg Options

Use Case

One use case requiring a change to the default settings is when you need to enable multi-threaded SQL data loads (which are not the same as parallel data loads) using any driver (whether JDBC, ODBC, included with Essbase, or not included with Essbase). For example, when loading SQL data into aggregate storage cubes, you can use up to eight rule files to load data in parallel. Each rule file must include the same authentication information (SQL user name and password).

Essbase initializes multiple temporary aggregate storage data load buffers (one for each rules file), where data values are sorted and accumulated. When the data is fully loaded into the data load buffers, Essbase commits the contents of all buffers into the cube in one operation, which is faster than committing buffers individually.

In the following MaxL example, SQL data is loaded from two rules files (rule1.rul and rule2.rul):

```
import database AsoSamp.Basic data
   connect as TBC identified by 'password'
   using multiple rules_file 'rule1' , 'rule2'
   to load_buffer_block starting with buffer_id 100
   on error write to "error.txt";
```

Change the SQL Configuration

To enable multithreaded data loads from external SQL sources as in the use case above, you must change the default settings for the drivers that are distributed with Essbase, by modifying `esssql.cfg`. Otherwise, concurrent users attempting data loads will encounter an error such as: `ERROR - 1021025 - SQL driver [`*`driverLibrary`*`] for [`*`DriverName`*`] is in use already and does not allow multiple connections. Please try later.`

You also need to add entries to `esssql.cfg` when you want to connect to a database using ODBC drivers not included by Oracle with Essbase.

To change the SQL configuration settings,

1. Locate the SQL configuration file on the Essbase domain:

   ```
   <DOMAIN HOME>/config/fmwconfig/essconfig/essbase/esssql.cfg
   ```

2. Save a copy of the file with a different name. For example, make a copy named `esssql_cfg.orig`.
3. Open `esssql.cfg` for editing.

   You configure this file using multiple blocks (one for each driver). Blocks are delineated by square brackets `[]`. You can copy and paste a block, and edit the settings as needed for each driver.
4. Make the necessary changes, depending on your requirements, and save the file. See esssql.cfg Options below. To enable multi threaded operations, for example, change the value of SingleConnection to 0.
5. Stop and restart the Essbase server, as described in Start, Stop, and Check Servers.

Sample esssql.cfg File

The following sample `esssql.cfg` has 6 configuration blocks. The first one is there by default, and the next five are added as examples of configuring different sources to enable multi-threaded SQL data loads.

```
[
Description "Oracle BI Server"
DriverName libnqsodbc
UserId 1
Password 1
Database 1
SingleConnection 0
IsQEDriver 0
ConvertUTF16toUTF8 1
]
[
Description "SQL Server"
DriverName SQLSRV
UserId 1
Password 1
Database 1
SingleConnection 0
IsQEDriver 0
]
[
Description "Oracle SQL Server Wire Protocol"
DriverName ARSQLS
UserId 1
Password 1
Database 1
SingleConnection 0
IsQEDriver 1
]
[
Description "Oracle DB2 Wire Protocol"
DriverName ARDB2
UserId 1
Password 1
Database 1
SingleConnection 0
IsQEDriver 1
]
[
Description "Oracle MySQL Wire Protocol"
DriverName ARMYSQL
UserId 1
Password 1
Database 1
SingleConnection 0
IsQEDriver 1
]
[
Description "Teradata"
DriverName TDATAODBC_SB64
```

```
UserId 1
Password 1
Database 1
SingleConnection 0
IsQEDriver 0
]
```

For the Teradata DriverName, you can use either TDATA or TDATAODBC_SB64.

esssql.cfg Options

The `esssql.cfg` configuration file must contain at least the driver file name (DriverName). Optionally, it can include a description (Description). The configuration file may contain additional keywords, the values for which are 0 or 1.

To find the DriverName on Linux, look in `odbcinst.ini`, located in

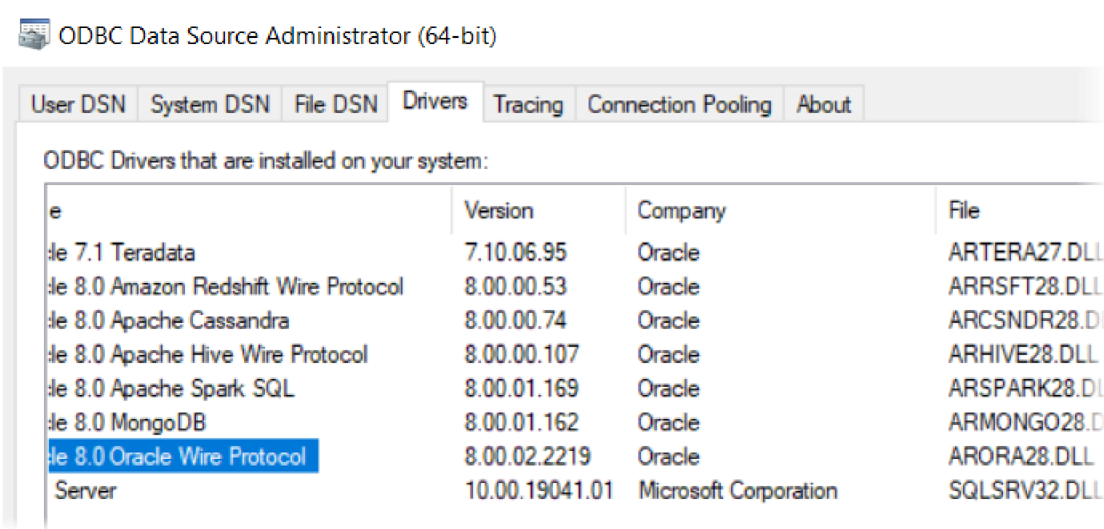
```
<Essbase Product Home>/modules/<driver class>/<driver version>
```

The driver name is at the end of the path given for the driver.

For example, from the following entry in `odbcinst.ini`, we know that the driver name is `ARora` (no need to include the version information, `28`):

```
Driver=/scratch/username/oracle_home/essbase/modules/
oracle.essbase.datadirect.odbc/8.0.2/lib/ARora28.so
```

To find the DriverName on Windows, look in the Windows 64-bit ODBC Data Source Administrator utility, on the Drivers tab.



Keyword	Value	Value = 0	Value = 1
Description	(Optional) A description of the driver, enclosed in double quotation marks. The default value is ""	N/A	N/A

Keyword	Value	Value = 0	Value = 1
DriverName	(Required) The driver file name; for example ARORA	N/A	N/A
UserId	0 or 1	User ID not required (default)	User ID required
Password	0 or 1	Password not required (default)	Password required
Database	0 or 1	Database name not required (default)	Database name required
Server	0 or 1	Server name not required (default)	Server name required
Application	0 or 1	Application name not required (default)	Application name required
Dictionary	0 or 1	Dictionary name not required (default)	Dictionary name required
Files	0 or 1	File name not required (default)	File name required
SingleConnection	0 or 1	Driver thread-safe -- Multiple active connections permitted	Driver not thread-safe -- One active connection permitted
IsQEDriver	0 or 1	Driver is not shipped with Essbase (default)	Driver is shipped with Essbase
ConvertUTF16toUTF8	0 or 1	No conversion of UTF16 data to UTF8 (this is the default).	Convert UTF16-encoded data from an Oracle BI data source to UTF8. This is required on UNIX for SQL data loads to Essbase from OBI.

Defaults apply to values that are not specified. The defaults applied within configuration files differ from the Essbase default values that apply if no `esssql.cfg` file exists.

Keywords and values must be separated by at least one space, and the set of keywords and values for each driver must be enclosed within brackets ([ ]).

Different drivers may require additional values. See the driver documentation for specific information.

Access Data Using JDBC Connectivity

If your source data is network accessible and configured using JDBC, you can configure Essbase to work with the source data.

See Create Connections and Datasources for Generic JDBC Drivers.

Stream from a Remote Database

If the data or dimensions you want to load to a cube are in a remote database, you can use the stream option in the Essbase Command Line Interface (CLI) utility to push the data or members to your cube using a rule file.

When you use the **stream** option for the CLI dataload or dimbuild commands, you must also reference a saved JDBC connection that reflects your driver and connection strings.

Workflow for Streaming

1. Log in to the Essbase web interface and download the CLI from the Console. See Download and Use the Command-Line Interface (CLI).
2. Log in to the CLI. See Login/Logout: CLI Authentication.
3. Create a saved JDBC connection string that reflects your data source's driver and connection strings, using the createlocalconnection command. Example (for Teradata):

   ```
   esscs createLocalConnection -name TeraConn -connectionString
   jdbc:teradata://192.0.2.110:1025/TBC -driver com.teradata.jdbc.TeraDriver -
   user TBC -p TBC
   ```

4. (Not required for Oracle database) Set an environment variable EXTERNAL_CLASSPATH to point to the .jar file for your database driver. See the *EXTERNAL_CLASSPATH Examples* section in this topic.
5. Run the CLI dataload or dimbuild commands with the streaming option, providing the saved connection name.

   You can optionally specify the database query in the dataload or dimbuild command, as in the Teradata example below. Otherwise, you can specify it in the load rule.

   ```
   esscs dataload -v -application Sample -db Basic -rule Data.rul -S -conn
   TeraConn -query "Select * from tbc.product"
   ```

EXTERNAL_CLASSPATH Examples

You must set the EXTERNAL_CLASSPATH environment variable before you can stream from any data source other than Oracle Database. Set the variable to point to the location of the relevant database driver .jar file.

DB2

Set the EXTERNAL_CLASSPATH variable to point to the location of the DB2 driver jar file.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/db2jcc.jar
```

Korn or bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/db2jcc.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\db2jcc.jar
```

MySQL

Set the EXTERNAL_CLASSPATH variable to point to the location of the MySQL driver jar file.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/mysql-connector-java-5.1.43-bin.jar
```

Korn or bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/mysql-connector-java-5.1.43-bin.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\mysql-connector-java-5.1.43-bin.jar
```

Microsoft SQL Server

Set the EXTERNAL_CLASSPATH variable to point to the location of the SQL Server driver jar file.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/sqljdbc4-3.0.jar
```

Korn or bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/sqljdbc4-3.0.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\sqljdbc4-3.0.jar
```

Teradata

Set the EXTERNAL_CLASSPATH variable to point to the location of both Teradata driver jar files.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/tdgssconfig.jar:/scratch/db/jars/
terajdbc4.jar
```

Korn or bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/tdgssconfig.jar:/scratch/db/jars/
terajdbc4.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\tdgssconfig.jar;C:\db\jars\terajdbc4.jar
```

Export Load Rules for Offline Editing

You can export an Essbase dimension build or data load rule to JSON format, to view and edit advanced options offline or using REST API.

The JSON is structured into dimensions and fields, and reveals build methods and all other defined load rule settings. You can edit the JSON and re-import it to the cube directory, to apply your changes to a dimension build or data load.

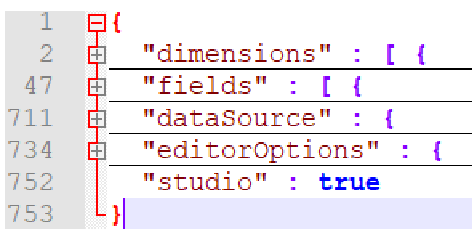
Learning Task

Export a rule file to JSON to view its format. Prerequisite: First, import the Sample.Basic cube from the gallery of application workbooks.

In this exercise, you'll view the JSON format of a dimension build rule for the Product dimension of Sample.Basic.

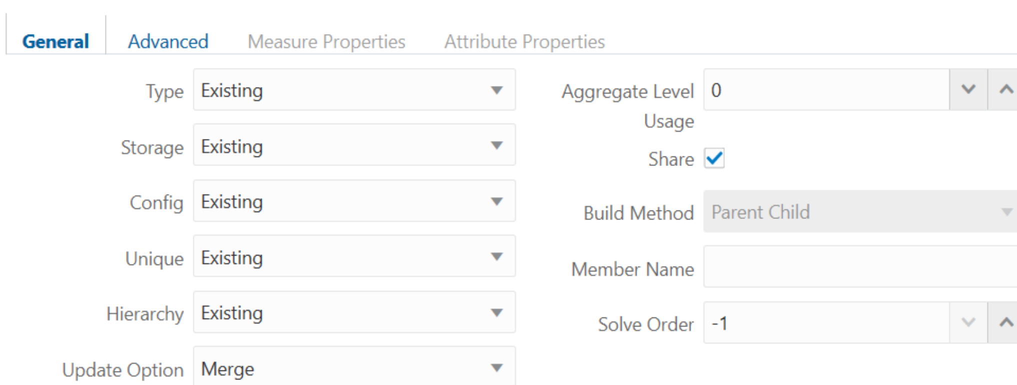
1. On the **Applications** page, expand the application (Sample).
2. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.
3. Select the **Scripts** tab, and then click **Rules**.
4. From the **Actions** menu, to the right of Dim_Product, select **Export**, and save `Dim_Product.json` to a local directory.
5. Use a text editor to examine the details of the Product dimension rule settings.

   If you have an advanced editor that recognizes JSON, you can collapse the main sections as shown, to observe the high-level structure of this JSON format rule file.



   - The **dimensions** section contains key/value pairs matching dimension settings that are defined in the rule.

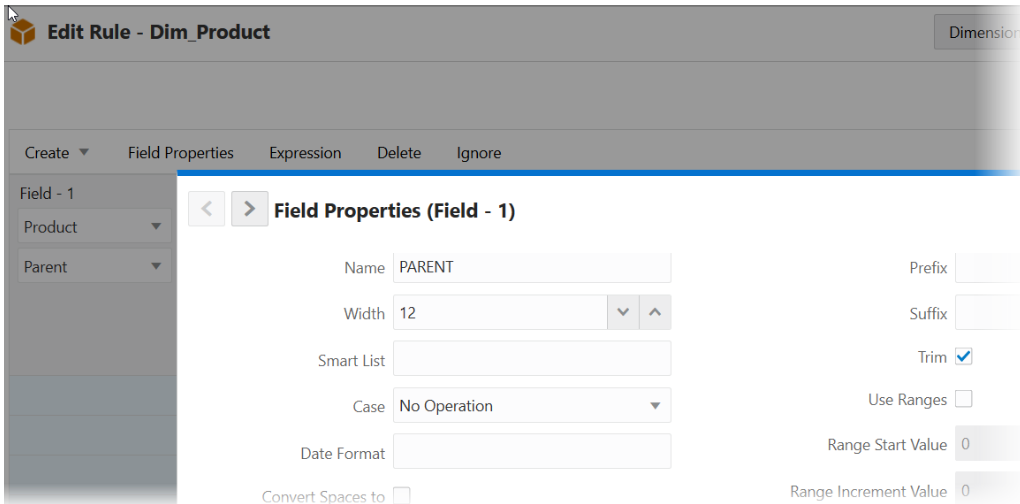
     For example, you can find the following general settings for the Product dimension represented in the JSON file.



> **Note:**
>
> "Existing" specified as the value for any given setting indicates that the designer of the rule has designated no change in preference for that setting (the default is retained).

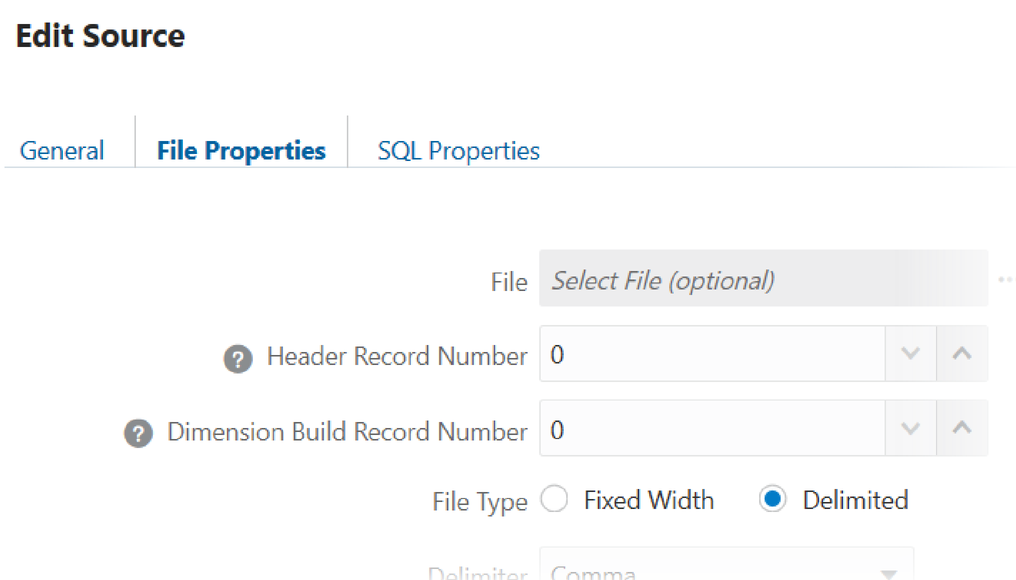
- The **fields** section is repeated for each field in the rule, and contains key/value pairs matching the settings defined in the rule for each field.

  For example, you can find the following field properties for Field 1 of the Product dimension represented in the JSON file.



- The **dataSource** section contains all the settings you can find in the **Source Properties** area of the rule definition.

  For example, the JSON file includes the General properties, and either the File or SQL properties, that are defined in the rule.



- The key/pair **"studio" : true** indicates that this is an Index-based rule rather than a Regular type. An index-based rule is sometimes called a BPM rule. It has different validation requirements. To learn more about this kind of dimension build rule, see Create New Dimension Build Rule.

14 Performing and Debugging Data Loads or Dimension Builds

You can load data or members from one or more external data sources to an Essbase database. You can load data without updating the outline, update the outline without loading data, or load data and build dimensions simultaneously.

Before you can load data or build dimensions, you must have an Essbase cube, source of data, and a load rule.

Performing Data Loads or Dimension Builds

You can load data or build dimensions when you build a cube from an application workbook, or afterwards using jobs in the Essbase web interface. You can also automate the job using CLI, MaxL, or REST API.

When you perform a data load or dimension build, the process runs in the background so that you can continue working during the load or build. You can then check the status of the background process to see when the load or build has completed.

If you are using multiple data sources in a dimension build, to reduce total processing time you can perform a deferred-restructure dimension build. See Incremental Dimension Builds.

> **Note:**
>
> If you are loading data into a transparent partition, follow the same steps as for loading data into a local database.

Essbase web interface

Load data or build dimensions using these Job types:

- For data loading: Load Data
- For dimension building: Build Dimension

Command Line Interface (CLI)

Load data or build dimensions using these CLI commands:

- For data loading: Dataload

- For dimension building: Dimbuild

MaxL

Load data or build dimensions using these MaxL statements:

- For data loading: Import Data
- For dimension building: Import Dimensions

REST API

Load data or build dimensions using REST API Job endpoints:

- For data loading: Execute Job > Load Data
- For dimension building: Execute Job > Build Dimensions

Build Dimensions and Load Data by Streaming from a Remote Database

If the data or dimensions you want to load to a cube are in a remote database, you can use the stream option in the Essbase Command Line Interface (CLI) to push the data or members to your cube.

When you use the **stream** option for the CLI dataload or dimbuild commands, you must also reference a saved JDBC connection that reflects your driver and connection strings.

Before you Begin

1. The load rule file must exist in the cube directory.
2. The database query used to load data or build dimensions must have the same dimensionality as the columns in the load rule (in other words, the order of dimensions in the rule must match the order of dimensions in the SQL query).

Limits

- Substitution variables are not supported in SQL statements used in load rules.
- Only use SQL functions that are supported by JDBC. ODBC scalar functions are not supported in CLI.

Workflow for Streaming Dimension Builds and Data

1. Create a saved JDBC connection string that reflects your data source's driver and connection strings, using the CLI createlocalconnection command.
2. (Not required for Oracle database) Set an environment variable EXTERNAL_CLASSPATH to point to the .jar file for your database driver. See the *Examples of EXTERNAL_CLASSPATH Environmental Variables* section in this topic.
3. Run the CLI dataload or dimbuild command with the streaming option, providing the saved connection name.

   You can optionally specify the database query in the dataload or dimbuild command. Otherwise, you can specify it in the load rule (in SQL Properties).

Examples of Setting EXTERNAL_CLASSPATH Environment Variable

You must set the EXTERNAL_CLASSPATH environment variable before you can stream from any data source other than Oracle Database. Set the variable to point to the location of the relevant database driver .jar file.

DB2

Set the EXTERNAL_CLASSPATH variable to point to the location of the DB2 driver jar file.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/db2jcc.jar
```

Korn or Bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/db2jcc.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\db2jcc.jar
```

MySQL

Set the EXTERNAL_CLASSPATH variable to point to the location of the MySQL driver jar file.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/mysql-connector-java-5.1.43-bin.jar
```

Korn or Bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/mysql-connector-java-5.1.43-bin.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\mysql-connector-java-5.1.43-bin.jar
```

Microsoft SQL Server

Set the EXTERNAL_CLASSPATH variable to point to the location of the SQL Server driver jar file.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/sqljdbc4-3.0.jar
```

Korn or Bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/sqljdbc4-3.0.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\sqljdbc4-3.0.jar
```

Teradata

Set the EXTERNAL_CLASSPATH variable to point to the location of both Teradata driver jar files.

C Shell Example

```
setenv EXTERNAL_CLASSPATH /scratch/db/jars/tdgssconfig.jar:/scratch/db/jars/
terajdbc4.jar
```

Korn or Bash Shell Example

```
export EXTERNAL_CLASSPATH=/scratch/db/jars/tdgssconfig.jar:/scratch/db/jars/
terajdbc4.jar
```

Windows Example

```
set EXTERNAL_CLASSPATH=C:\db\jars\tdgssconfig.jar;C:\db\jars\terajdbc4.jar
```

Stopping Data Loads or Dimension Builds

You can stop an Essbase data load or dimension build if necessary.

You should not stop a data load or dimension build before it completes unless it is necessary to do so. If a data load or dimension build process is terminated, Essbase displays the file name as partially loaded.

If you initiate a data load or dimension build from a client and terminate the data load or dimension build from the server, it could take time before the client responds to the termination request. Because Essbase reads the source file until all source data is read, the amount of time depends on the file size and the amount of source data that Essbase has processed. If the process is terminated from the computer that initiated it, termination is immediate.

To stop a data load or dimension build before it completes, use the Console in Essbase web interface (see View Job Status and Details).

Tips for Loading Data and Building Dimensions

To save time, you can perform incremental dimension builds to Essbase, and you can load data subsets. To prevent overwriting consolidated values, avoid loading data into parent members. To avoid data processing errors, ensure each record in the source has the same number of data value fields.

See these topics for more information to help you load data and build dimensions:

- Incremental Dimension Builds
- Where to Load Data
- Missing Fields in the Source Data
- Loading a Subset of Records from the Source

Incremental Dimension Builds

To enable Essbase to read multiple sources of data while building the cube outline, use incremental dimension builds. Incremental dimbuild delays outline restructuring until all sources of data have been processed.

Skip this topic if you are loading data only, or are using a single data source for a dimension build.

By default, each time you make changes to an outline, Essbase considers the type of change, and restructures the cube if needed. Restructuring rebuilds the database, which consumes time and disk space.

Incremental dimension builds, also known as deferred-restructure dimension builds, read multiple data sources for dimension builds and delay restructuring until all data sources have been processed.

If you add a dimension that is virtual (Dynamic Calc or label only), then any data existing in the cube is stored with the first level-0 stored member in the new dimension. There must be at least one stored member in the hierarchy, or an error will be returned during outline verification.

When you incrementally add a dimension to an existing cube using an application workbook, the data is automatically mapped to the new top member. There is not a way to choose a stored member to which to map the existing data. If the new dimension has a top member that is dynamic calc, the data is lost because dynamic members can't store data.

When using an application workbook to add a new dimension in which you want the top member to be dynamic calc, follow these steps:

1. Add the new dimension with the top member as stored.
2. Run a calc script to copy the data from the new top member into another stored member in that dimension.
3. Change the top member to dynamic calc.

MaxL enables you to include all of the data sources within one import statement. You can control whether outline validation is performed for each file. You must enable outline validation for the last file.

In all cases, the data sources are processed in the order in which they are listed.

> **Note:**
>
> MaxL enables you to enforce or suppress outline verification for each file. To ensure a valid outline, ensure that the last build verifies the outline.

See Also

Update Cubes Incrementally in Cube Designer

Optimizing Database Restructuring (block storage cubes)

Aggregate Storage Cube Restructuring (aggregate storage cubes)

Where to Load Data

Before performing an Essbase data load, take steps to prevent overwriting data unintentionally.

Skip this section if you are building dimensions or working with an aggregate storage database.

If you load data into a parent member, when you calculate the database, the consolidation of the children's data values can overwrite the parent data value. To prevent overwriting:

- If possible, do not load data directly into a parent.
- If you must load data into a parent member, ensure that Essbase knows not to consolidate #MISSING values from the children of the parent into the parent.

You can set the consolidation operator in an application workbook, the outline or calculation scripts:

- Application workbook: See Understanding Dimension Worksheets.
- Outline: See Setting Information Properties.
- Calculation script: Use the SET AGGMISSG calculation command.

You can set the consolidation using the **alter database** MaxL statement.

The methods in this table work only if the child values are empty (#MISSING). If the children have data values, the data values overwrite the data values of the parent. See #MISSING Values.

> **Note:**
>
> You cannot load data into Dynamic Calc or attribute members. For example, if Year is a Dynamic Calc member, you cannot load data into it. Instead, load data into Qtr1, Qtr2, Qtr3, and Qtr4, which are not Dynamic Calc members.

Missing Fields in the Source Data

When performing an Essbase data load, each record in the source data must have the same number of data value fields. If data values are missing, the data load processes incorrectly.

If a dimension field or member field is missing, Essbase uses the value that it used previously for that dimension or member field.

If a rule has extra blank fields, join the empty fields with the field next to them.

In aggregate storage cubes, values can be loaded only to level 0 cells. Specifying #MISSING or #MI as a value in the source removes the associated cell, if it is present in the cube.

For information about data load differences for aggregate storage cubes, refer to:

- Build Dimensions in Aggregate Storage Cubes
- Load Data into Aggregate Storage Cubes

Loading a Subset of Records from the Source

You can load a subset of records from the source data during an Essbase data load or dimension build.

To load a subset of records:

1. Using a text-editing tool, number the records in the data source.
2. Set the load rule to ignore the column containing the record number.
3. Define rejection criteria to reject all records except those that you want to load. For example, reject all records for which the ignored column is fewer than 250 or greater than 500.

   You cannot reject more records than the error log can hold. By default, the limit is 1000. You can change the limit by setting the DATAERRORLIMIT configuration.

Debugging Data Loads and Dimension Builds

Problems with Essbase data loads and dimension builds include incorrect loads or failed processes. The source may contain unknown members or missing / invalid fields, the load rule may be invalid, or there is no connection to the server or the source data. Error logs can help you debug.

If a data source does not correctly load into Essbase Server, ensure that you are connected to the appropriate application and database and that you are loading the correct data source.

If you still encounter problems, see these topics:

- Verifying that Essbase Server Is Available
- Verifying that the Data Source Is Available
- Checking Error Logs
- Resolving Problems with Data Loaded Incorrectly
- Creating Rejection Criteria for End of File Markers
- Understanding How Essbase Processes a Rules File
- Understanding How Essbase Processes Missing or Invalid Fields During a Data Load

After you correct the problems, you can reload the records that did not load by reloading the error log.

Verifying that Essbase Server Is Available

To help identify that the problem is with Essbase and not with the server or network, try to access the server without using Essbase. Check whether:

- The server is running.

  Try connecting to the server without using Essbase. If you cannot, check with your system administrator.
- Essbase Server is running.

  Check with your Essbase administrator.
- The client is connected to the server.

Try connecting to the server from the client without using Essbase.

Verifying that the Data Source Is Available

If Essbase cannot open the data source that you want to load, ensure that the following conditions are true:

- The data source is open (for example, is someone editing the data source?).

  Essbase can load only data sources that are not locked by another user or application.
- The data source has the correct file extension.

  Text files must have a `.txt` extension; rules files must have an `.rul` extension.
- The data source name and the path name are correct.

  Check for misspellings.
- The data source is in the specified location.

  Ensure that no one has moved or deleted the data source.
- If you are using a SQL data source:
  - The connection information (such as the user name, password, and database name) is correct.
  - You can connect to the SQL data source without using Essbase.

Checking Error Logs

If a data load or dimension build fails, the error log can be a valuable debugging tool.

If there is no error log, check whether the following conditions exist:

- The person running the data load set up an error log.

  By default, when using a rules file, Essbase creates an error log.
- The data source and Essbase Server are available.

  See Verifying that Essbase Server Is Available and Verifying that the Data Source Is Available
- Essbase Server crashed during the data load.

  If so, you probably received a timeout error on the client.
- The application log exists.

If the error log exists but is empty, Essbase does not think that an error occurred during loading. Check whether the following conditions exist:

- The rules file contains selection or rejection criteria that rejected every record in the data source.

  See Select and Reject Records.
- The rules file validates properly.

  See Requirements for Valid Data Load Rule Files and Requirements for Valid Dimension Build Rule Files.

Resolving Problems with Data Loaded Incorrectly

If the data source loads without error, but the data in the database is wrong, check whether the following conditions exist:

- You loaded the correct data source.

  If so, check the data source again to make sure that it contains the correct values.
- The data source is formatted correctly.
  - All ranges are set up properly.
  - The data is clean. For example, as it processes the data source, Essbase recognizes member names and knows the dimensions they belong to. If a data source record inadvertently includes a member from a dimension for which there is a member named in the header record, the new member name replaces the header record member for that dimension. In the following example data source, Essbase recognizes Florida as a member of the Market dimension. The values in the last four records are interpreted as Florida values instead of Texas values.

```
Jan  Actual  Texas  Sales
   "100-10"   51.7
   "100-20"   102.5
   "100-20"   335.0
   Florida    96.7
   "200-20"   276.0
   "200-20"   113.1
   "200-10"   167.0
```

- You added incoming data to existing data instead of replacing incoming data with existing data.

  See Adding to and Subtracting from Existing Values.
- You selected or rejected any records that you did not intend to select or reject.

  See Select and Reject Records.
- The sign is reversed (for example, a minus sign instead of a plus sign) and whether you performed sign flips on any UDAs.

  See Flipping Field Signs.
- You cleared data combinations that you did not intend to clear.

  See Clearing Existing Data Values.
- You scaled the incoming values incorrectly.

  See Scaling Data Values.
- All member and alias names are fewer than 79 characters long.

To check data in Smart View, see the *Working with Oracle Smart View for Office*.

Creating Rejection Criteria for End of File Markers

A SQL data source may have an end of file marker made up of special characters that cause a data load or dimension build to fail. To fix this problem, define a rejection criterion to reject the problem record.

To define rejection criteria:

1. Find the end of file marker in the SQL data source.
2. Determine how to search for the end of file marker using the Essbase search command.

   This task may be difficult, because the end of file marker may be composed of one or more special characters.
3. Define a rejection criterion that rejects the end of file marker.

Understanding How Essbase Processes a Rules File

Sometimes, you can track down problems with dimension builds by understanding how Essbase initializes the rules file and processes the data source.

Essbase performs the following steps to initialize a rules file:

1. Validates the rules file against the associated outline.
2. Validates the dimensions. This process includes ensuring that the build method and field types are compatible and that each dimension name is unique. Member names must be either unique or shared.
3. Adds new dimensions defined in the rules file to the outline.
4. Reads header records specified in the data source.

Then Essbase performs the following operations on each record of the data source during a data load or dimension build:

1. Sets the file delimiters for all records.
2. Applies field operations to the data in the order in which the operations are defined in the rules file.

   Field operations include joins, moves, splits, and creating fields using text and joins. To see the order in which field operations are defined in the rules file, see Perform Operations on Fields.
3. Essbase applies all properties for each field, applying all properties to field1 before proceeding to field2. Essbase applies field properties in the following order:
   a. Ignores fields set to be ignored during data load
   b. Ignores fields set to be ignored during dimension build
   c. Flags the data field
   d. Applies field names
   e. Applies field generations
   f. Performs all replaces in the order in which they are defined in the rules file
   g. Drops leading and trailing spaces
   h. Converts spaces to underscores
   i. Applies suffix and prefix operations
   j. Scales data values
   k. Converts text to lowercase
   l. Converts text to uppercase
4. Adds members, or member information, or both, to the outline

5. If you chose to skip lines, Essbase skips the number of lines that you specified; otherwise, Essbase proceeds to the first record.
6. Essbase performs selection or rejection criteria in the order in which the criteria are defined in the rules file. Essbase loads or rejects individual records of the data source based on the specified criteria.

Understanding How Essbase Processes Missing or Invalid Fields During a Data Load

See these topics to understand how Essbase processes invalid fields during a data load.

- Missing Dimension or Member Fields
- Unknown Member Fields
- Invalid Data Fields

Missing Dimension or Member Fields

If a dimension or member field is missing, Essbase uses the value that it used previously for that dimension or member field. If there is no previous value, Essbase aborts the data load.

For example, when you load the following file into the Sample.Basic database, Essbase maps the Ohio member field into the Market dimension for all records, including the records that have Root Beer and Diet Cola in the Product dimension.

```
Jan   Sales   Actual   Ohio
                       Cola           25
                       "Root Beer"    50
                       "Diet Cola"    19
```

Essbase stops the data load if no prior record contains a value for the missing member field. For example, if you try to load the following file into the Sample.Basic database, the data load stops, because the Market dimension (Ohio, in the previous example) is not specified.

```
Jan   Sales   Actual
               Cola           25
              "Root Beer"     50
              "Diet Cola"     19
```

Unknown Member Fields

If you are performing a data load and an unknown member name is encountered, the entire record is rejected. If there is a prior record with a member name for the missing member field, Essbase continues to the next record. If there is no prior record, the data load stops. For example, when you load the following file into the Sample.Basic database, the record containing Ginger Ale is rejected because it is not a valid member name. The records containing Cola, Root Beer, and Cream Soda are loaded. If Ginger Ale were in the first record, however, the data load would stop.

```
Jan, Sales, Actual
Ohio    Cola          2
        "Root Beer"   12
```

```
"Ginger Ale"  15
"Cream Soda"  11
```

> **Note:**
>
> If you are performing a dimension build, you can add the new member to the database. See Performing Data Loads or Dimension Builds.

Invalid Data Fields

If you are performing a data load, the data load stops if an invalid data field is encountered. All fields that are read before the invalid field are loaded into the cube, resulting in a partial data load. For example, in the following file, the data load stops when it encounters the 15- data value. The Jan and Feb Sales records are loaded but not the Mar and Apr Sales records.

```
East    Cola    Actual
Sales           Jan     $10
                Feb     $21
                Mar     $15-
                Apr     $16
```

15
Dimension Build Examples

The build method you should use to load dimensions to an Essbase outline depends on the type of data in the source system. Design your load rules with the build method in mind. Build methods determine the algorithm that is used to add, change, or remove dimensions, members, and aliases.

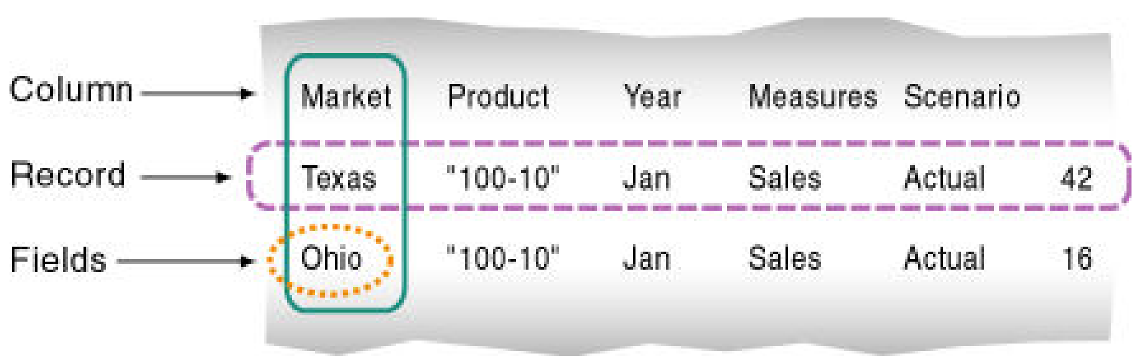
All examples in this chapter are based on the Sample.Basic database.

All examples in this chapter are based on the Sample.Basic database, which you can create using the Block Storage Sample (Stored) sample application workbook (Sample_Basic.xlsx). See About Application Workbooks.

Dimension Build Methods

The dimension build method determines the algorithm Essbase uses to build or modify dimensions, members, and aliases in the cube outline. Depending on the shape of your source of data, the ideal build method may reference generations, levels, parent-child, or strings.

When you build dimensions, each record (row) in the source of data defines one member of the dimension. Each record must follow the same dimensional order ("dimensionality") as the other records.



Build Method Guidelines

Use these guidelines to select the appropriate build method for the data source:

Type of Data in Each Record	Examples	Desired Operation	Build Method[1]	Field Type Information
Top-down data Each record specifies the parent's name, the child's name, the children of that child, and so on.	Year, Quarter, Month	Modify the properties of existing dimensions and members	Generation references	The generation number for each field.
Bottom-up data Each record specifies the name of the member, the name of its parent, the name of its parent's parent, and so forth.	Month, Quarter, Year	• Create shared members that roll up into different generations • Modify the properties of existing dimensions and members	Level references	The level number for each field.
Parent followed by its child Each record specifies the name of the parent and the name of the new child member, in that order, although they can specify other information as well.	Cola, Diet Cola	• Create shared members that roll up into different generations • Share non-level 0 members • Modify properties of existing dimensions and members	Parent-child references	
A list of new members Each data source lists new members; the data source does not specify where in the outline the members belong. Essbase provides algorithms that determine where to add these members.	Jan, Feb, Mar, April	Add all members as children of an existing parent (possibly a "dummy" parent)	Add as child of the specified parent	
	800-10, 800-20	Add all members at the end of the dimension	Add as sibling at the lowest level	
	800-10, 800-20	Add each new member to the dimension that contains similar members	Add as sibling to a member with a matching string	
A list of base dimension members and their attributes	Cola 16oz Can, Root Beer 14oz Bottle	Add members to an attribute dimension and associate the added members with the appropriate members of the base dimension	Generation, level, or parent-child references, depending on the organization of the source data	The number for each field. The number is either the generation or level number of the associated member of the base dimension or zero.

1 Using a level references build, you cannot create an alias that has the same name as its member. This restriction does not apply if you use other build methods, including the generation references build method.

Selecting Build Method for a New Dimension Build Rule

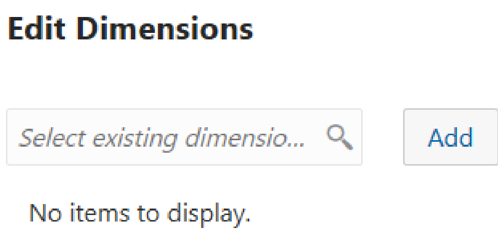
When you use application workbooks to build dimensions, you specify the build method in the definitions of the dimension worksheet. For example, this definition specifies a parent-child build method.

Definitions	
File Name	Dim_Product
Rule Name	Dim_Product
Build Method	PARENT-CHILD
Incremental Mode	Merge

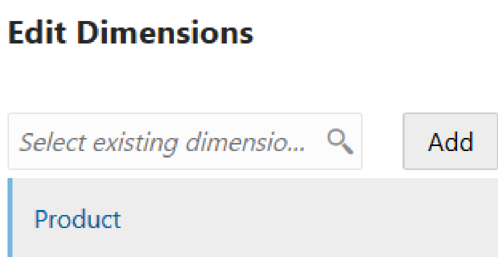
When you design a new dimension build rule in the Essbase web interface, you can select a build method when you first associate a dimension with the rule. For example,

1. In a new dimension build rule, click **Dimensions**.

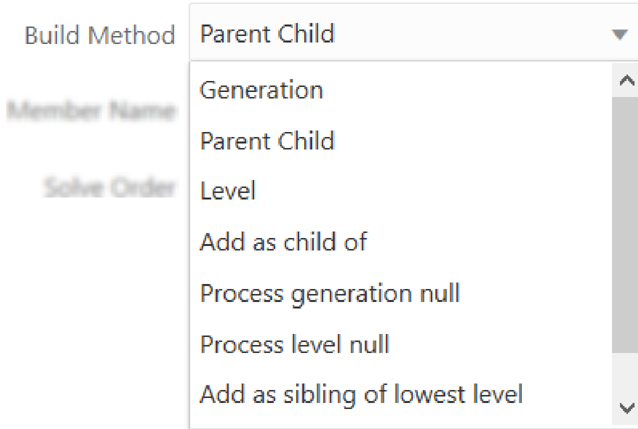
2. Type the dimension name in the text field, and click **Add**.



3. Click the name of the dimension you just added. For example, click **Product**.



4. Select one of the build methods and click OK.

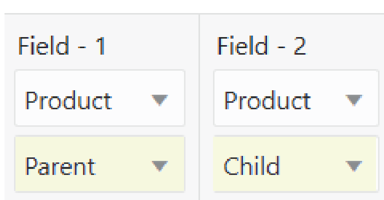


The following build method options are available.

Build Method	Use Case
Generation	Generation References in Dimension Builds
Parent Child	Using Parent-Child References
Level	Level References in Dimension Builds
Add as child of	Not currently available
Process generation null	Generation References in Dimension Builds
Process level null	Level References in Dimension Builds (see "Dealing With Empty Fields)
Add as sibling of lowest level	Adding Members as Siblings of the Lowest Level
Add as sibling of matching string	Adding Members Based On String Matches

Identifying Build Method for an Existing Dimension Build Rule

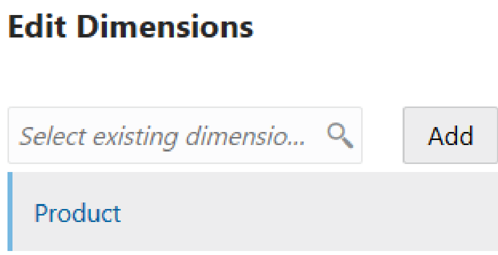
When you view or edit a dimension build rule in the Essbase web interface, examine the order and type of fields typically indicate the build method that was used. For example, the fields in this load rule are ordered and typed to specify a parent-child build method.



To check the build method in the rule settings,

1. Click **Dimensions**.

2. Click the name of the dimension this rule is associated with. For example, click **Product**.

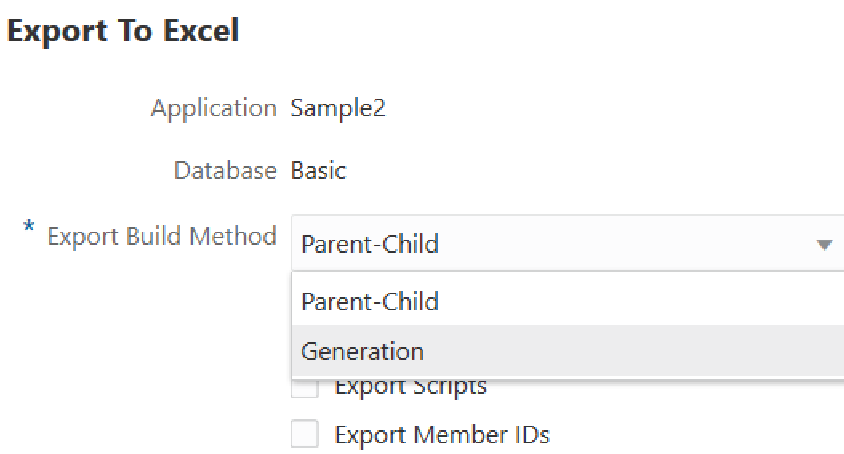


3. The build method is visible on the **General** tab.

Exporting an Existing Cube with a Different Build Method

You can export any cube to an application workbook, and select a new build method upon export. Export supports Parent-Child and Generation build methods.

1. On the **Applications** page, expand the application.
2. From the **Actions** menu, to the right of the cube name, click **Export To Excel**.
3. Select the build method.



4. Select one of the build methods and click OK.

See Also

Set Dimension Build Field Type Information

Generation References in Dimension Builds

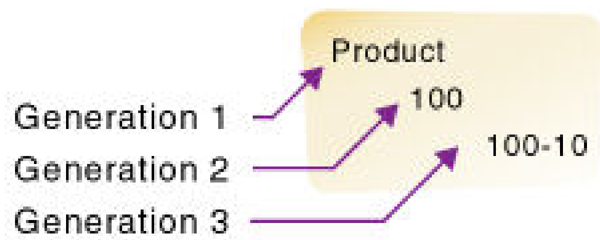
Essbase dimension members occupying a common branch are in the same generation. Generation 1 is the dimension name, and each lower level increments the generation number. When building dimensions from a top-down source, use the **Generation** build method. If source data contains nulls, use the **Process generation nulls** build method.

Top-down sources of data are organized left to right from the highest level to the lowest level. Each record begins with the most general information and progresses to the most specific information. The name of the new member is at the end of the record. When using a top-down source of data, use the generation references build method. In the rules file, specify the generation number and the field type of each field of the source of data.

Essbase numbers members within a dimension according to the hierarchical position of the member within the dimension. The numbers are called *generation references*. A dimension is always generation 1. All members at the same branch in a dimension are called a *generation*. Generations are numbered top-down according to their position relative to the dimension; that is, relative to dimension 1.

For example, as illustrated below, the Product dimension is generation 1. Product has a 100 member, which is generation 2. 100 has members, such as 100-10, which are generation 3. To use the generation references build method, specify the generation reference number in the rules file.

Figure 15-1 Generations



Assume you have a cube with a Product dimension. Dimensions are generation 1, so Product is generation 1. Assume you want to use the following top-down, tab-delimited data file to build the Product dimension:

```
500     500-10     500-10-10
500     500-10     500-10-20
500     500-20     500-20-12
500     500-20     500-20-15
500     500-20     500-20-20
```

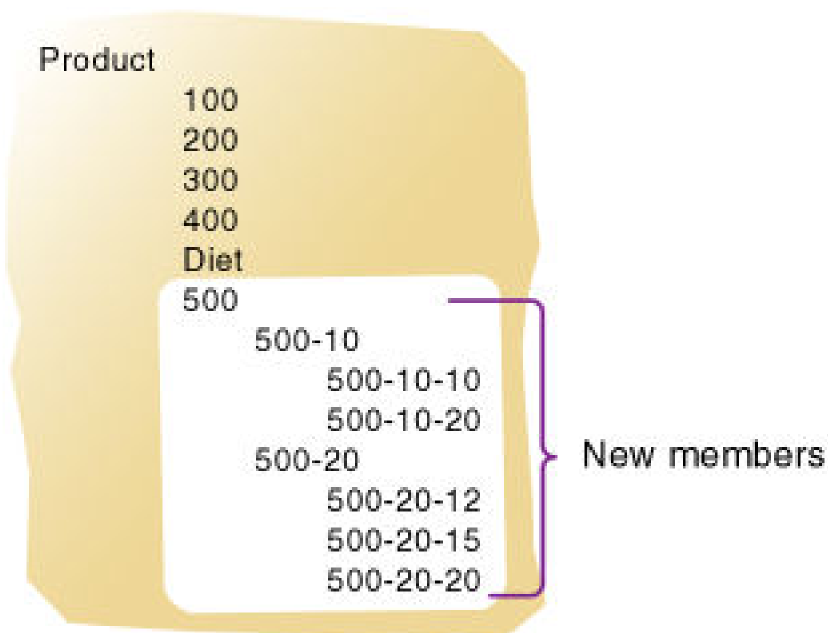
Typically, you would run a dimension build using the **Generation** build method when the lowest-level members are sequenced right to left as shown above.

You can use the following rule to build the dimensions using generation references. The rule specifies the generation number for each field in the source of data. You must also select the **Generation** build method in the dimension properties of the rule, unless your source data contains nulls (for that use case, see Dealing with Empty Fields, below).

Field - 1	Field - 2	Field - 3
Product	Product	Product
Generation	Generation	Generation
2	3	4
Category	Product SKU	Promotions
500	500-10	500-10-10
500	500-10	500-10-20
500	500-20	500-20-12
500	500-20	500-20-15
500	500-20	500-20-20

Essbase builds the following hierarchy from the source of data and rule:

Figure 15-2   Generation References



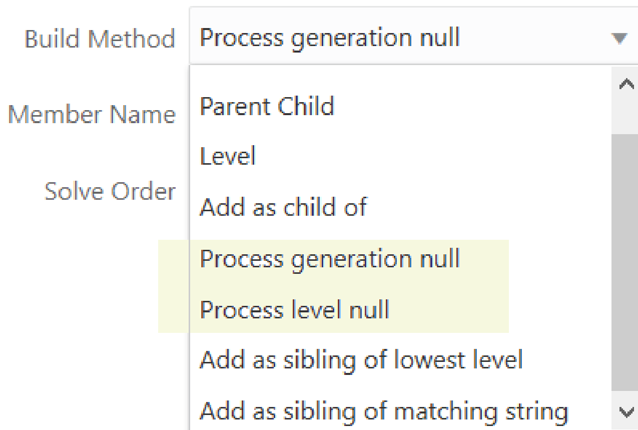
Dealing with Empty Fields

When you build dimensions from generation or level references in the source data, you can choose to process null values. Null processing specifies what actions Essbase should take when it encounters empty fields in the source of data.

> **Note:**
>
> Null processing options are available only for regular dimension build rules (not index-based).

If null processing is not enabled, Essbase rejects all records with null values and writes an error to the error log.

To enable null processing, select one of the null processing build methods in the dimension properties when you create the new dimension build rule.



If you use the **Process generation null** build method, it tells Essbase to expect some null values while processing generation references in the source data, which helps prevent errors.

Example

For this dimension build example, assume you want to add a ragged/asymmetric hierarchy like the following into a new Channel dimension on Sample Basic.

The data source is top down, meaning that the higher levels (close to the top dimension member) appear first in each record. Therefore, it makes sense to use a generation references build method. However, null handling instructions will be required. The first three source data records contain null fields, as there is no location data for the indirect channels.

```
"All Channels",Indirect,,Channel-04
"All Channels",Indirect,,Channel-07
"All Channels",Indirect,,Channel-24
"All Channels",Direct,Outlet,Channel-21
"All Channels",Direct,Mall,Channel-29
"All Channels",Direct,Kiosk,Channel-31
```

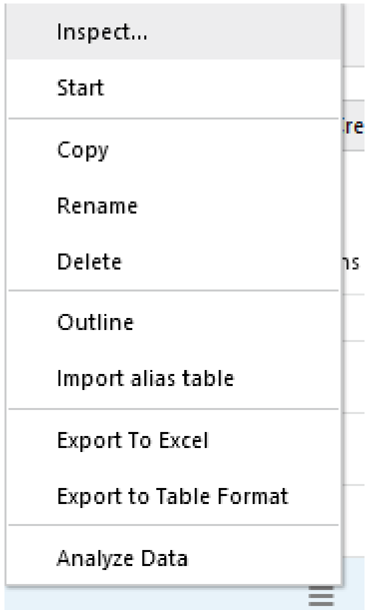
You can create a rule to build the dimensions and avoid errors relating to nulls, using the **Process generation null** build method.

To create a rule for this example,

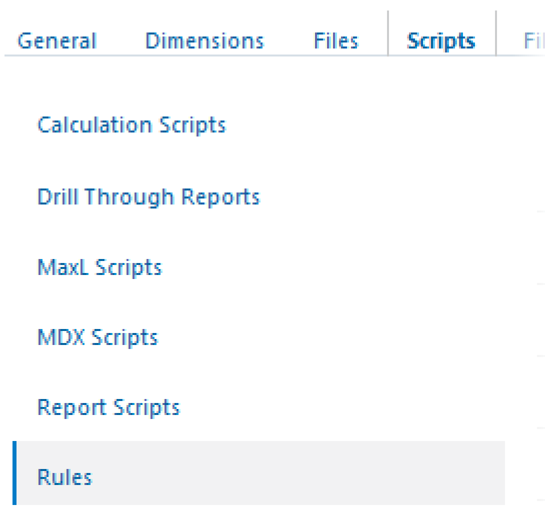
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a comma-delimited data file like the following, name it `genchannel.txt`, and upload it to the Sample Basic cube directory.

   ```
   "All Channels",Indirect,,Channel-04
   "All Channels",Indirect,,Channel-07
   "All Channels",Indirect,,Channel-24
   "All Channels",Direct,Outlet,Channel-21
   "All Channels",Direct,Mall,Channel-29
   "All Channels",Direct,Kiosk,Channel-31
   ```

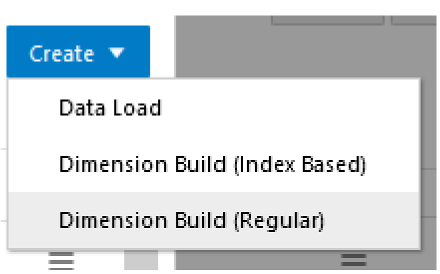
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.



6. Click **Create** and choose **Dimension Build (Regular)**.



7. In the New Rule dialog,
   a. Enter a rule name; for example, **genchannel**.

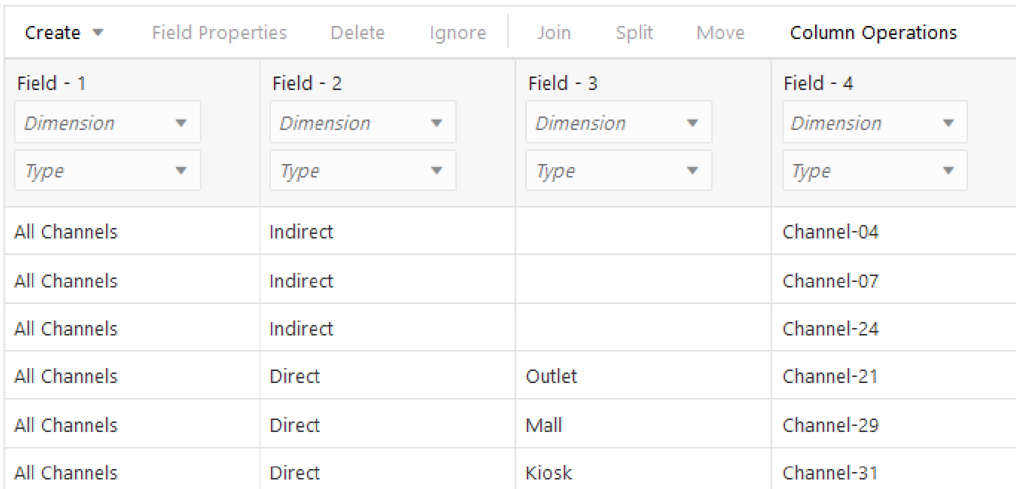
b. For **Source Type** select **File**, click **Catalog** and navigate to `genchannel.txt`.

   When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.

c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.

d. Click **Proceed**.

e. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.



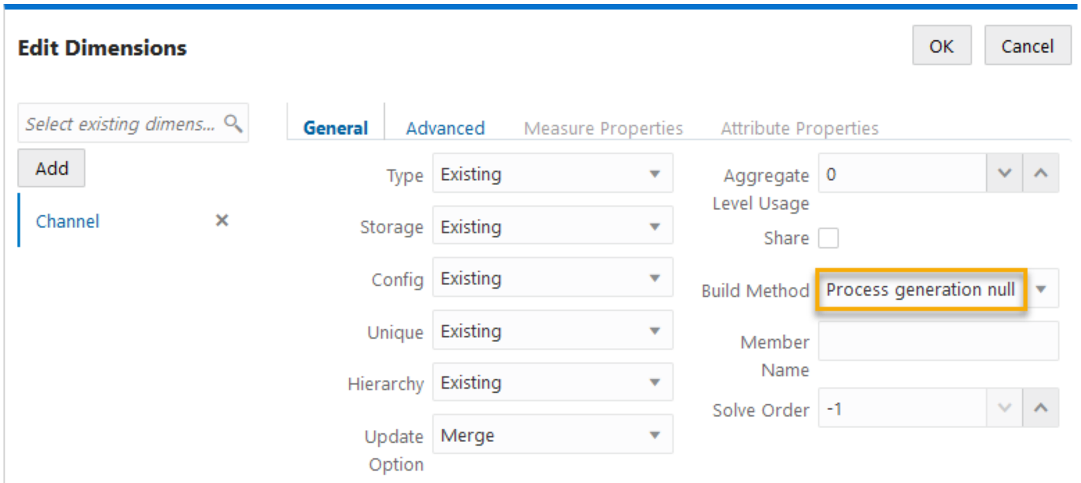
8. As this is a new rule, there are no dimensions associated yet.

   a. Click **Dimensions**.

   b. Type the new dimension name **Channel**, and click **Add**.

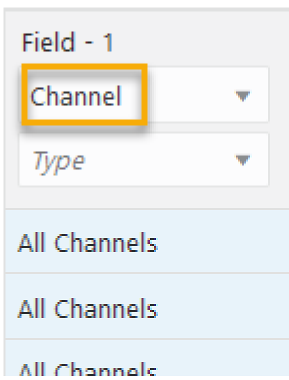
9. Now you need to define the build method and check other dimension build operational instructions. Click the **Channel** link to edit the dimension properties.

10. Change **Build Method** to **Process generation null**.

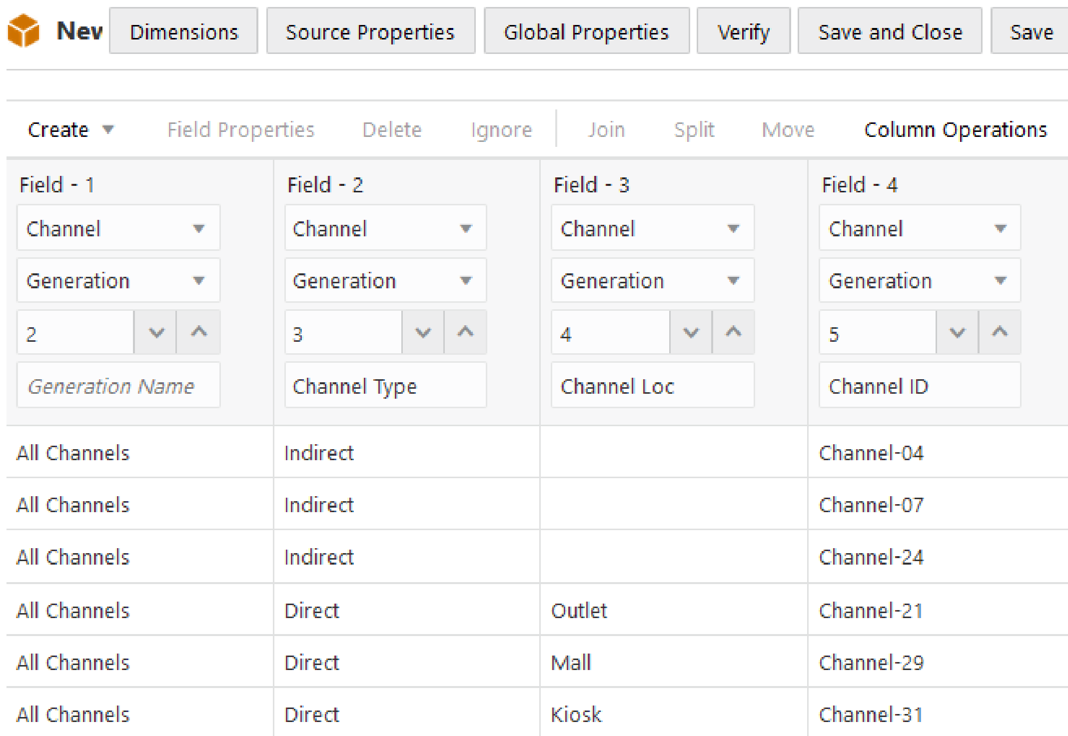


Click OK.

11. Click the *Dimension* selector in Field 1, and select **Channel**.



12. Change the *Type* selector of Field 1 to **Generation**.
13. Increment the *Generation* to 2, as the All Channels hierarchy member is at at level 2. The generation 1 member is always the dimension name; in this case, Channel.
14. In Fields 2 - 4, **Channel** should now be selected as the dimension. Change the *Type* selectors to **Generation**, and mark the generation numbers as **3** for Field 2, **4** for Field 3, and **5** for Field 4.



15. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.
16. Run the dimension build job.
17. View the outline to confirm that the Channel dimension and the expected hierarchy of members were added.

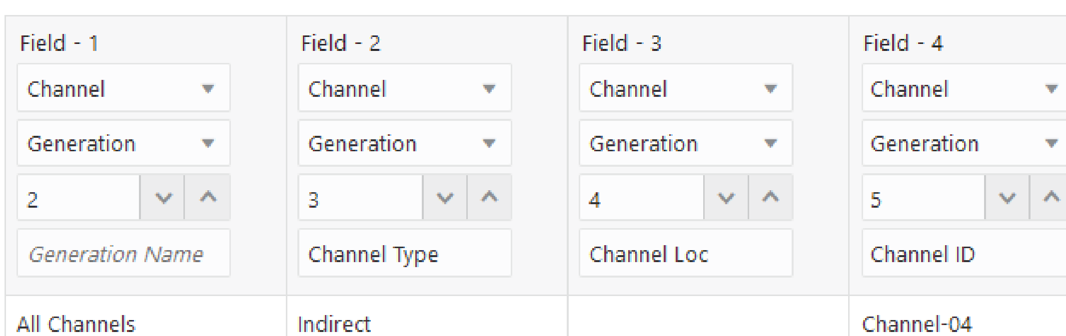
Null Processing Flow for Generation-based Dimension Builds

If null processing is enabled, Essbase processes generation nulls in the following ways:

- **Null generation field:** If the null occurs where Essbase expects a Generation field, Essbase promotes the next Generation field to replace the missing field.

  Example

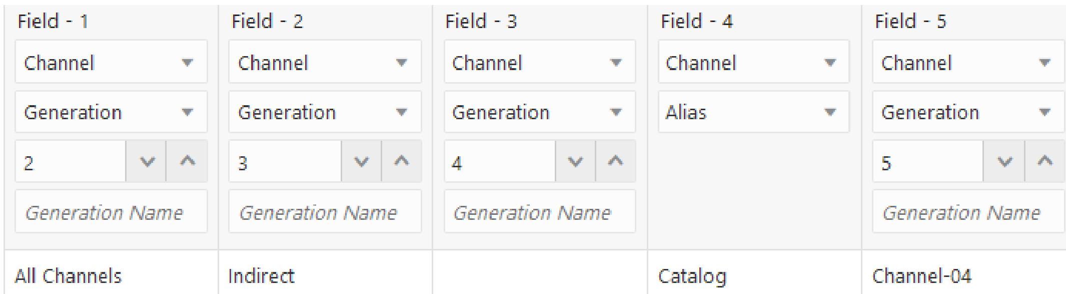
  The dimension build promotes the channel ID to Generation 4 instead of Generation 5, because Generation 4 contains a null.



- **Null field before secondary field:** If a null occurs directly before a secondary field, Essbase ignores the secondary field. (Secondary field types are alias, property, formula, duplicate generation, duplicate generation alias, currency name, currency category, attribute parent, UDA, and name of an attribute dimension.)

  Example

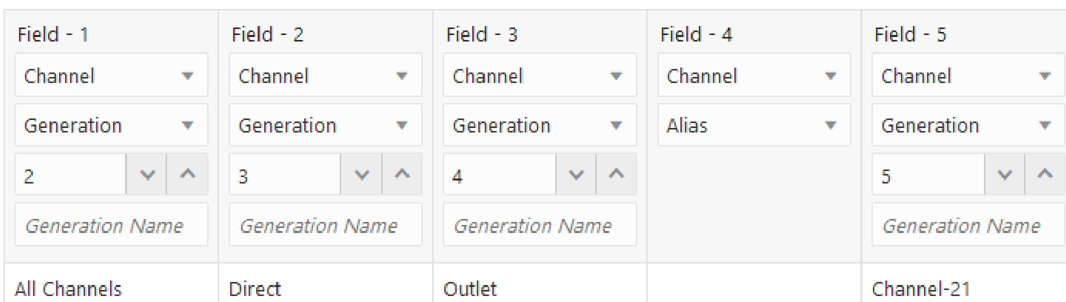
  The dimension build ignores the alias in field 4, because Generation 4 contains a null, and promotes the channel ID to Generation 4 instead of Generation 5.

- **Null secondary field:** If the null occurs where Essbase expects a secondary field, Essbase ignores the secondary null field and continues loading (this is the same behavior as for the **Generation** build method).

  Example

  The dimension build ignores the alias in Field 4, because that field contains a null. The member Outlet is added to the outline without any alias.

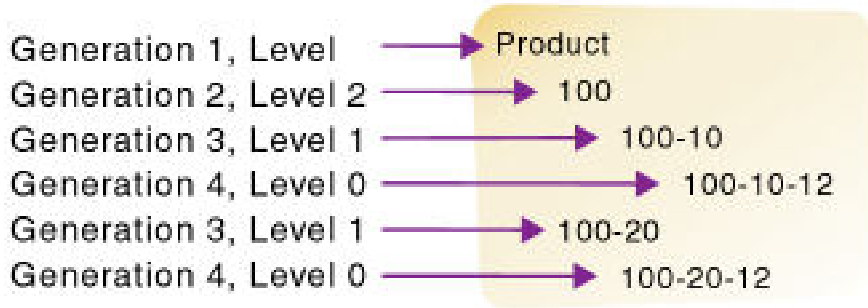
Level References in Dimension Builds

Levels are defined from a bottom-up hierarchy, where leaf members are level 0, and each step closer to the dimension name increments the level by 1. When building Essbase dimensions from a bottom-up source, use the **Level** build method. If source data contains nulls, use the **Process level nulls** build method.

In a bottom-up source of data, each record defines a single member of a dimension. The definition begins with the most specific information about the member and provides progressively more general information. A typical record specifies the name of the new member, then the name of its parent, then its parent's parent, and so forth.

For example, in the outline illustrated below, the lowest-level members are at the bottoms of the branches of the Product dimension.

Figure 15-3 Generation and Level Numbers



To build the outline, you can use the following bottom-up source of data:

```
100-10-12  100-10  100
100-20-12  100-20  100
```

Typically, you would run a dimension build using the **Level** build method when the lowest-level members are sequenced left to right. Level 0 (leaf level) members are in the first field, level 1 members are in the second field, and so on. This organization is the opposite of how data is presented for generation references (top-down).

Assume you have a cube with a Product dimension, and you want to use the following bottom-up, tab-delimited data file to build more product members to the Product dimension.

```
600-10-11    600-10    600
600-20-10    600-20    600
600-20-18    600-20    600
```

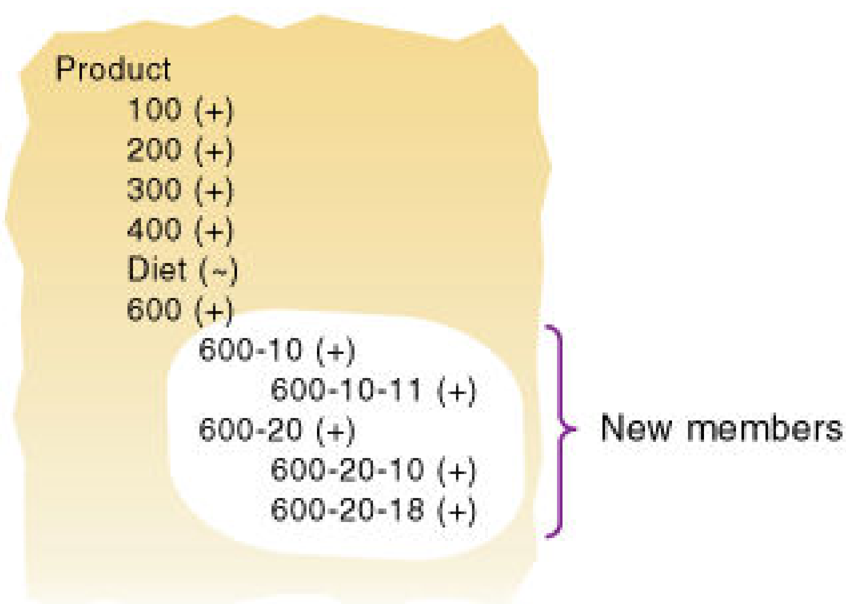
The source of data is "bottom-up" in that the first column of the source of data contains new leaf level members (600-10-11, 600-20-10, and 600-20-18). The second column contains the parents of the new members (600-10 and 600-20), and the third column contains parents of the parents (600).

The rule file uses the level reference build method to add the members to the Product dimension. The rule specifies the level number and the field type for each field of the source of data. You must also select the **Level** build method in the dimension properties of the rule, unless your source data contains nulls (for that use case, see Dealing with Empty Fields, below).

Field - 1	Field - 2	Field - 3
Product	Product	Product
Level	Level	Level
0	1	2
Promotions	Product SKU	Category
600-10-11	600-10	600
600-20-10	600-20	600
600-20-18	600-20	600

Essbase builds the following hierarchy from the source of data and rule:

Figure 15-4 Levels



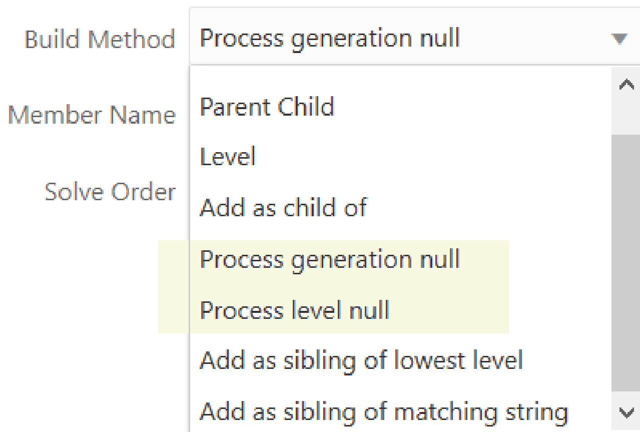
Dealing with Empty Fields

When you build dimensions from generation or level references in the source data, you can choose to process null values. Null processing specifies what actions Essbase should take when it encounters empty fields in the source of data.

> **Note:**
>
> Null processing options are available only for regular dimension build rules (not index-based).

If null processing is not enabled, Essbase rejects all records with null values and writes an error to the error log.

To enable null processing, select one of the null processing build methods in the dimension properties when you create the new dimension build rule.



If you use the **Process level null** build method, it tells Essbase to expect some null values while processing level references in the source data, which helps prevent errors.

Example

For this dimension build example, assume you want to add a ragged/asymmetric hierarchy like the following into a new Channel dimension on Sample Basic.

The data source is bottom up, meaning that the lowest levels appear first in each record. Therefore, it makes sense to use a level references build method. However, null handling instructions will be required. The first three source data records contain null fields, as there is no location data for the indirect channels.

```
Channel-04,,Indirect,"All Channels"
Channel-07,,Indirect,"All Channels"
Channel-24,,Indirect,"All Channels"
Channel-21,Outlet,Direct,"All Channels"
Channel-29,Mall,Direct,"All Channels"
Channel-31,Kiosk,Direct,"All Channels"
```

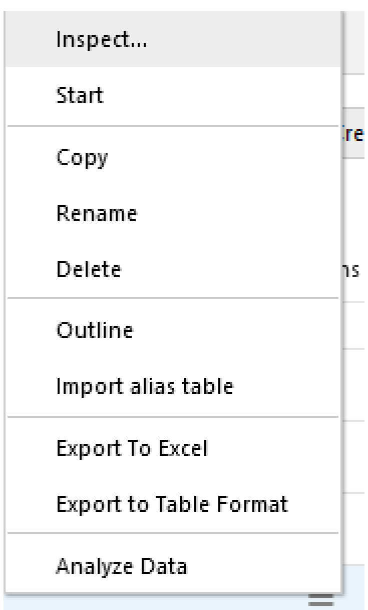
You can create a rule to build the dimensions and avoid errors relating to nulls, using the **Process level null** build method.

To create a rule for this example,

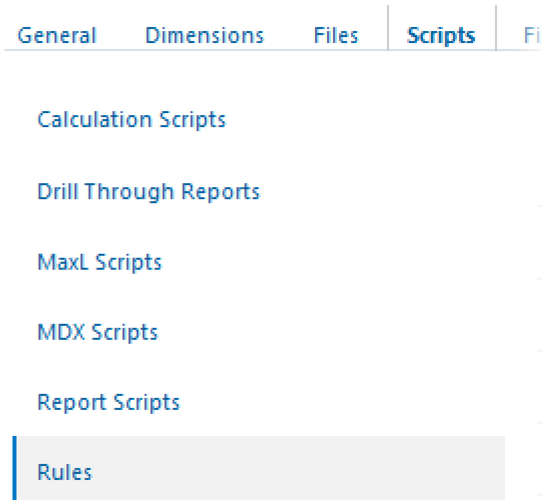
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a comma-delimited data file like the following, name it `levchannel.txt`, and upload it to the Sample Basic cube directory.

   ```
   Channel-04,,Indirect,"All Channels"
   Channel-07,,Indirect,"All Channels"
   Channel-24,,Indirect,"All Channels"
   Channel-21,Outlet,Direct,"All Channels"
   Channel-29,Mall,Direct,"All Channels"
   Channel-31,Kiosk,Direct,"All Channels"
   ```

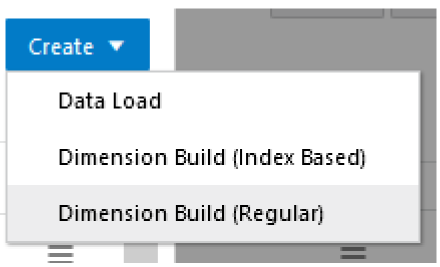
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.

5. Select the **Scripts** tab, and then click **Rules**.

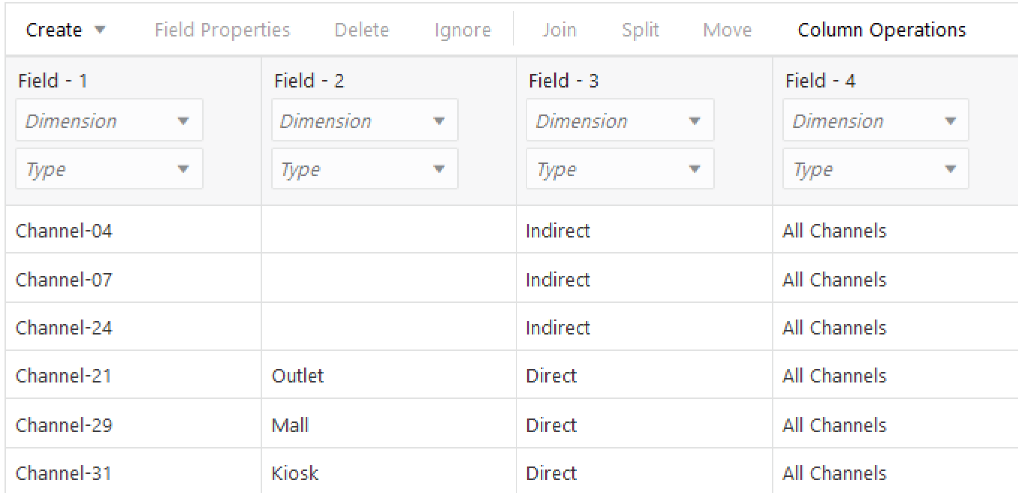


6. Click **Create** and choose **Dimension Build (Regular)**.



7. In the New Rule dialog,
   a. Enter a rule name; for example, **levchannel**.
   b. For **Source Type** select **File**, click **Catalog** and navigate to `levchannel.txt`.

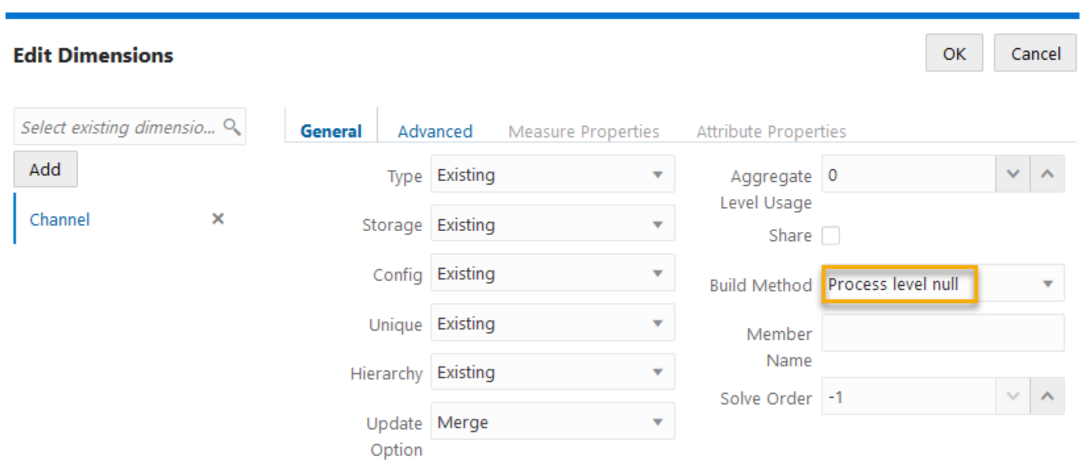
      When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.
   c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.
   d. Click **Proceed**.
   e. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.



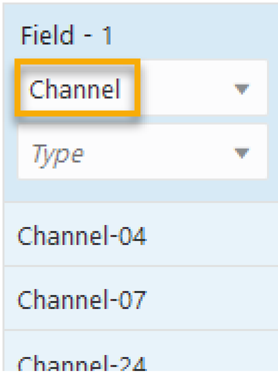
Field - 1	Field - 2	Field - 3	Field - 4
Dimension	Dimension	Dimension	Dimension
Type	Type	Type	Type
Channel-04		Indirect	All Channels
Channel-07		Indirect	All Channels
Channel-24		Indirect	All Channels
Channel-21	Outlet	Direct	All Channels
Channel-29	Mall	Direct	All Channels
Channel-31	Kiosk	Direct	All Channels

8. As this is a new rule, there are no dimensions associated yet.
   a. Click **Dimensions**.

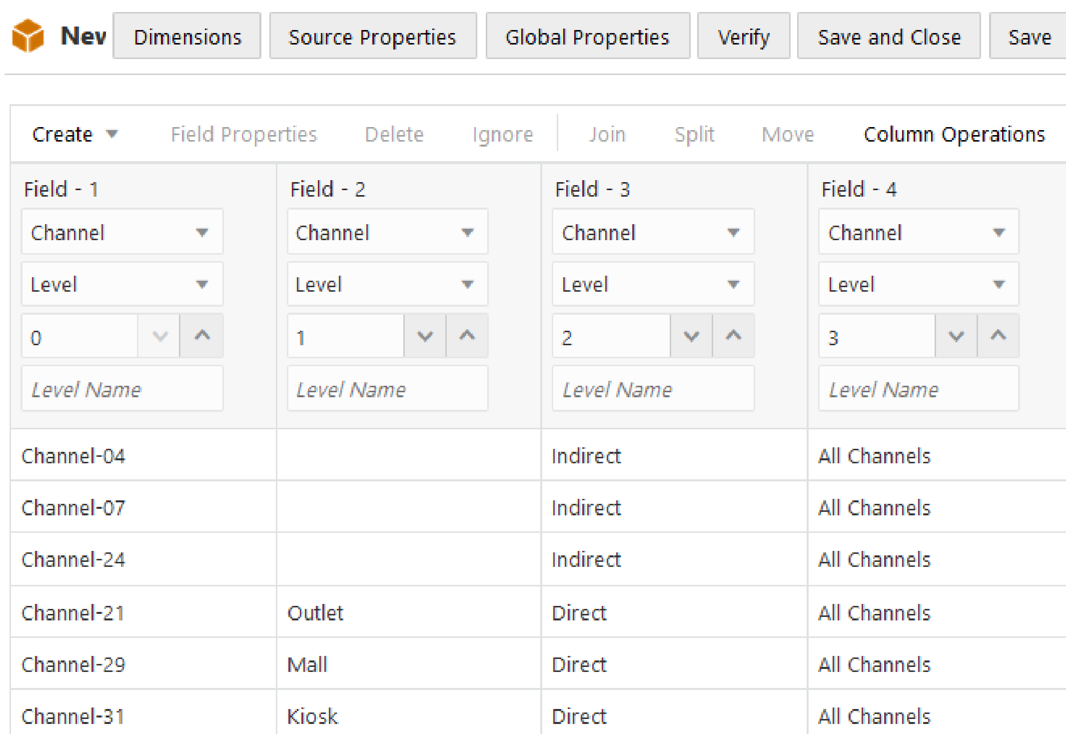
   b. Type the new dimension name **Channel**, and click **Add**.
9. Now you need to define the build method and check other dimension build operational instructions. Click the **Channel** link to edit the dimension properties.
10. Change **Build Method** to **Process level null**.



Click OK.
11. Click the *Dimension* selector in Field 1, and select **Channel**.



12. Change the *Type* selector of Field 1 to **Level**.
13. Increment the *Level* to 0, as the channel IDs are at level 0.
14. In Fields 2 - 4, **Channel** should now be selected as the dimension. Change the *Type* selectors to **Level**, and mark the level numbers as **1** for Field 2, **2** for Field 3, and **3** for Field 4.



15. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.
16. Run the dimension build job.
17. View the outline to confirm that the Channel dimension and the expected hierarchy of members were added.

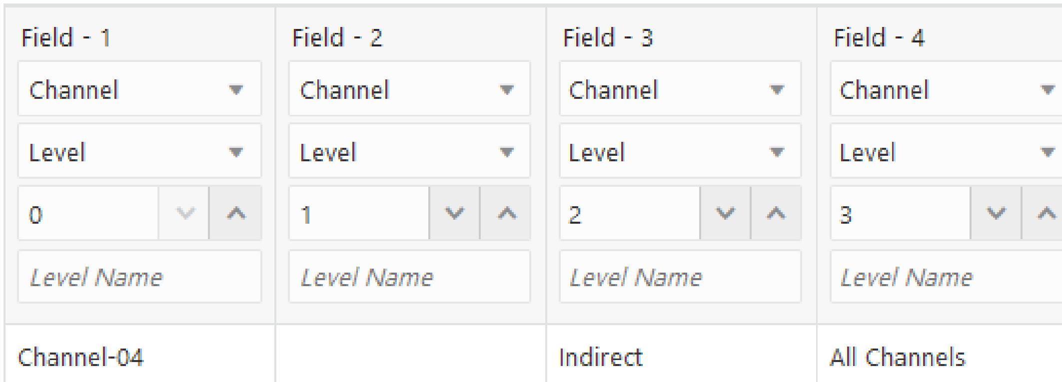
Null Processing Flow for Level-based Dimension Builds

If null processing is enabled, Essbase processes level nulls in the following ways:

- **Null level field:** If a null occurs where Essbase expects a Level field, Essbase promotes the next Level field to replace the missing field.

  Example

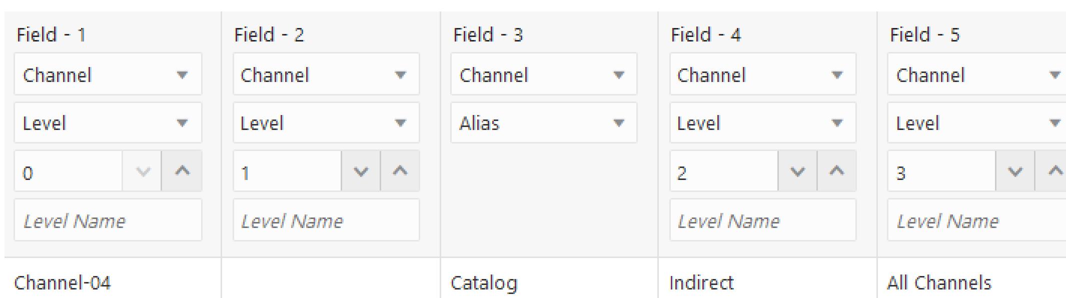
  The dimension build places Indirect in Level 1 instead of Level 2, because Field 2 for Level 1 contains a null.

- **Null field before a secondary field:** If a null occurs directly before a secondary field, Essbase ignores the secondary field. (Secondary field options are alias, property, formula, duplicate level, duplicate level alias, currency name, currency category, attribute parent, UDA, and a name of an attribute dimension.)

  Example

  The dimension build ignores the alias in Field 3, because Field 2 contains a null, and places Indirect in Level 1, and All Channels in Level 2.

- **Null secondary field:** If a null occurs where Essbase expects a secondary field, Essbase ignores the secondary null field and continues loading (this is the same behavior as for the **Level** build method).

  Example

  The dimension build ignores the first alias in Field 3, because that field contains a null. The member Outlet is added to the outline without any alias.

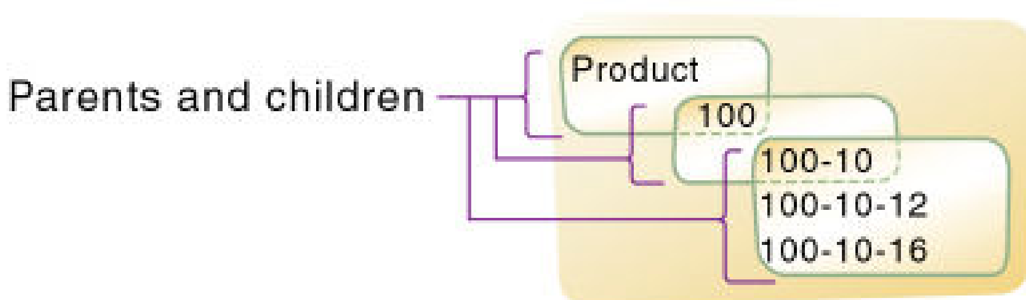
Field - 1	Field - 2	Field - 3	Field - 4	Field - 5
Channel	Channel	Channel	Channel	Channel
Level	Level	Alias	Level	Level
0	1		2	3
Level Name	Level Name		Level Name	Level Name
Channel-21	Outlet		Direct	All Channels
Channel-29	Mall	City Retail	Direct	All Channels
Channel-31	Kiosk	Mini Retail	Direct	All Channels

Using Parent-Child References

When performing Essbase dimension builds, use a parent-child build method when every record of the source of data indicates a hierarchical relationship including the name of a new member and the name of its parent.

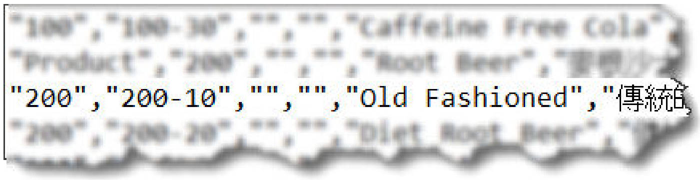
Members in a database exist in a parent-child relationship. The image below shows part of the Product dimension with its parent and children relationships identified. Product is the parent of 100. 100 is the child of Product and the parent of 100-10, 100-10-12, and 100-10-16. 100-10, 100-10-12, and 100-10-16 are the children of 100.

Figure 15-5   Parents and Children



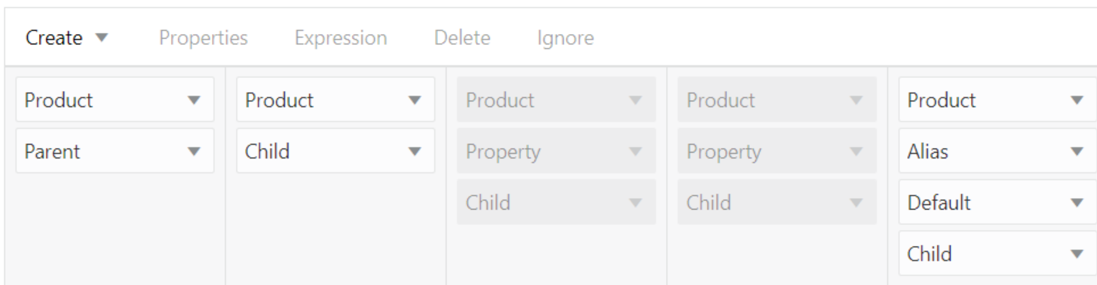
A parent-child source of data must contain at least two columns: a parent column and a child column, in that order. The source of data can include columns with other information (for example, the alias, the attributes, or the properties of the new member). A record within a parent-child source of data cannot specify multiple parents or multiple children, and cannot reverse the order of the parent and child columns.

In a parent-child build, the rules file specifies which column is the parent and which column is the child. For example, consider the source file (`Dim_Product.txt`), in which each record specifies the name of a parent and the name of its child, in that order.

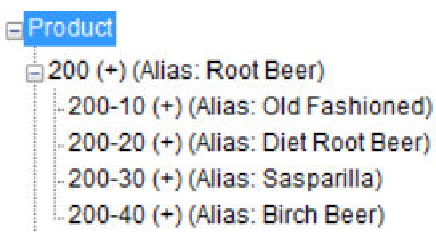


The rules file (`Dim_Product.rul`) specifies which column is the parent and which column is the child. Additionally, this example associates aliases with the child field. Null fields, reserved for properties, are ignored.

Edit Rule - Dim_Product



Essbase builds the following hierarchy from this source data and rules file.



> **Note:**
>
> For duplicate member situations, the parent field must contain the qualified member name. See Building Qualified Member Names Through the Rule File.

Adding a List of New Members

When you need to add new members to the Essbase outline, but the source data does not specify their ancestors, use one of the **Add as** dimension build methods: **Add as sibling of matching string**, **Add as a sibling at the lowest level**, or **Add as child of**.

Essbase provides the following dimension build methods for working with source data that does not specify parent members:

- Add each new member as a sibling of the existing member whose text most closely matches its own.

  See Adding Members Based On String Matches.

- Add each new member as a sibling of the lowest-level existing member.

  See Adding Members as Siblings of the Lowest Level.

- Add all new members as children of a specified parent. See Adding Members to a Specified Parent.

After Essbase adds all new members to the outline, it may be necessary to move the new members into their correct positions using Outline Editor. See Positioning Dimensions and Members.

> **Note:**
>
> Essbase does not support concurrent attribute association with the **Add as** build methods.

Adding Members Based On String Matches

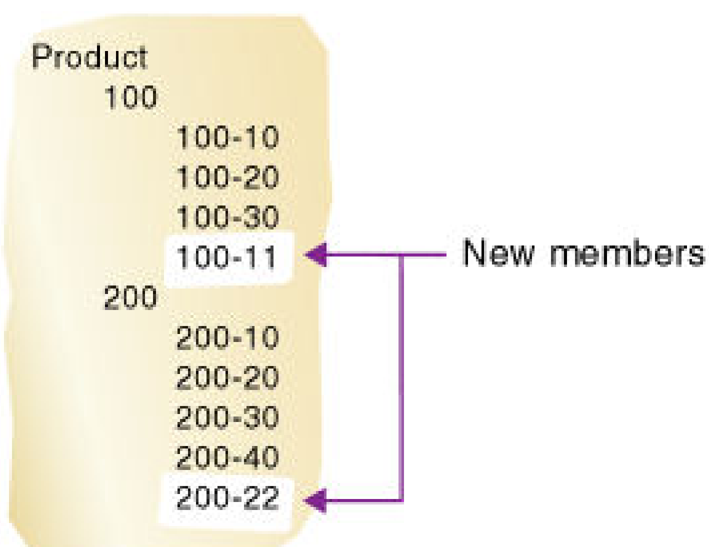
When you need to add new members to the Essbase outline but the source data does not specify their ancestors, you can design a dimension build rule that adds them as siblings to members with similar names, based on string matches. Use the **Add as sibling of matching string** dimension build method.

If you use this build method, when Essbase encounters a new member in the source, it scans the outline for a member name with similar text and adds the new member as a sibling of the member with the closest string match.

For example, the following text file of data contains two new members (100-11 and 200-22) to add to the Product dimension. The new members are similar to strings in the Product dimension: they contain three digits, one dash, and two digits.

```
100-11    Texas    Sales    100    120    100
200-22    Texas    Sales    111    154    180
```

The following hierarchy shows 100-11 added as a sibling of 100, and 200-22 added as a sibling of 200.

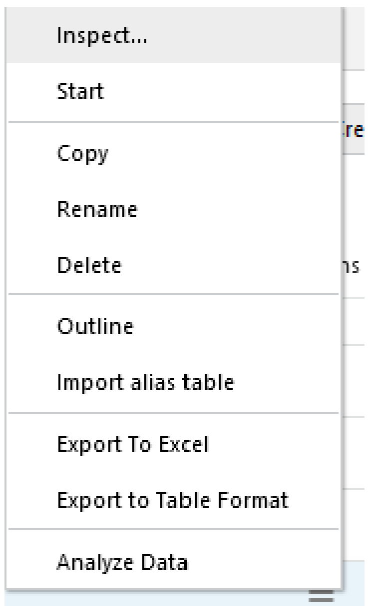


To recreate this example,

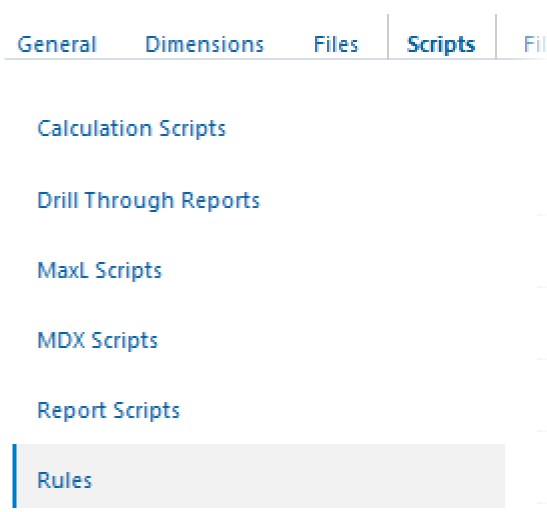
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `stringmatch.txt`, and upload it to the Sample Basic cube directory.

```
100-11    Texas    Sales    100    120    100
200-22    Texas    Sales    111    154    180
```

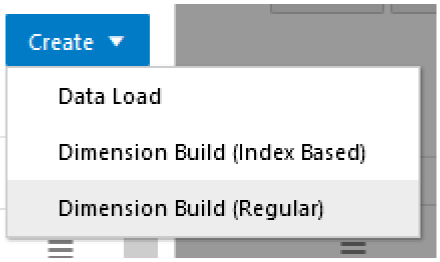
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.

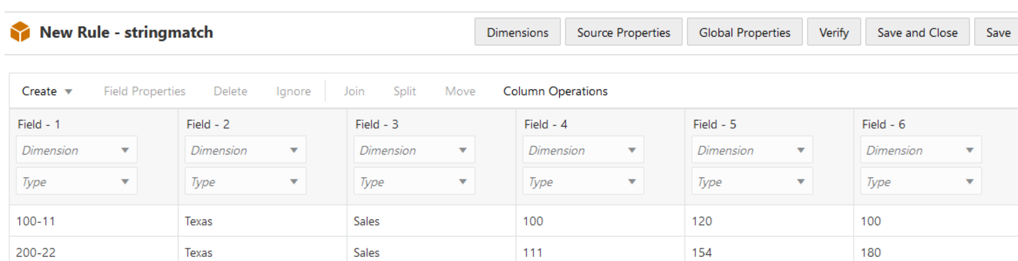


6. Click **Create** and choose **Dimension Build (Regular)**.

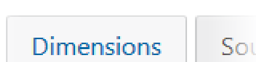


7. In the New Rule dialog,
   a. Enter a rule name; for example, **stringmatch**.
   b. For **Source Type** select **File**, click **Catalog** and navigate to `stringmatch.txt`.

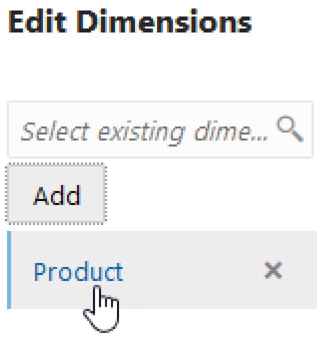
      When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.
   c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.
   d. Change the Delimiter value from Comma to **Tab**.
   e. Click **Proceed**.
   f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.



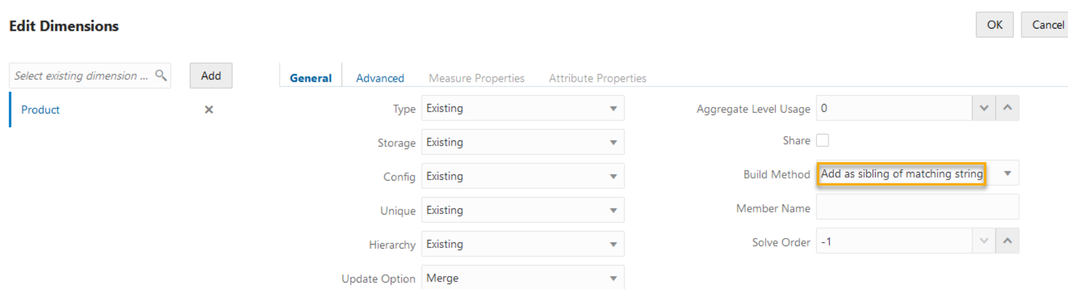
8. As this is a new rule, there are no dimensions associated yet.
   a. Click **Dimensions**.



   b. Select or type the dimension name **Product**, and click **Add**.
9. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.

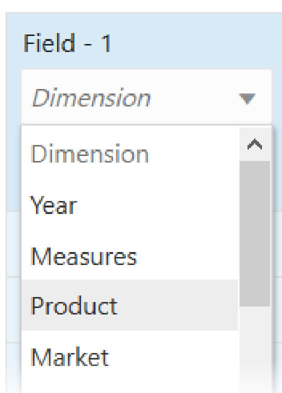


10. Change **Build Method** to **Add as sibling of matching string**, and leave the Member Name field underneath it blank.



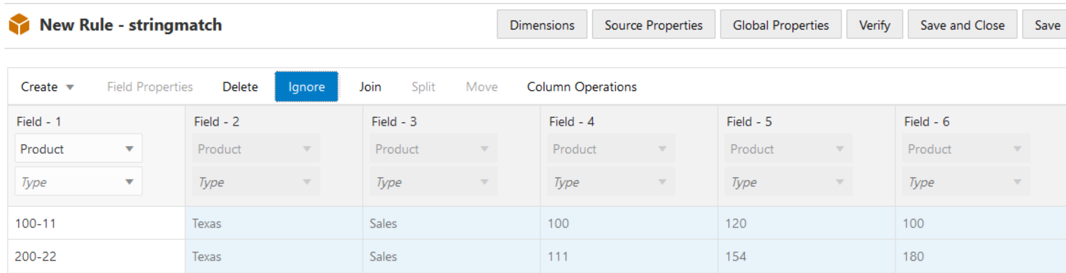
Click OK.

11. Click the *Dimension* selector in Field 1, and select **Product**.



12. Leave the *Type* selector of Field 1 blank.

13. Select Fields 2-6. To do this, click on the inactive header area labeled Field 2, hold down Shift key, and click the Field 6 header area. Click **Ignore**.



As a result, this rule will only operate on the first field of each record. Ignoring fields can be helpful when you don't need to use every field in the source for your dimension build, yet you don't want to prepare different sources.

14. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.
15. Run the dimension build job.
16. View the outline, and inspect the children Product members 100 and 200 to confirm that the members were added.

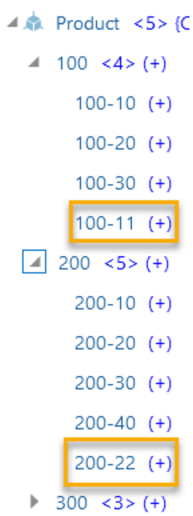


Table 15-1 Summary of Adding Members Using String Matches

Field	Value	More Information
Product dimension	Select the **Add as sibling of matching string** build method	Select a Build Method
Field 1 (Product)	• Do not select a field type for the field • Set the dimension for the field to Product	Set Dimension Build Field Type Information
Fields 2 through 6	Ignore the fields	Ignoring Fields

Adding Members as Siblings of the Lowest Level

When you need to add new members to the Essbase outline but the source data does not specify their ancestors, you can design a dimension build rule that adds them as siblings of the lowest-level existing member. Use the **Add as sibling of lowest level** dimension build method.

If you use this build method, when Essbase encounters a new member in the source data, it scans the outline for the level 0 branch of members and adds the new member(s) as a sibling of these members.

> **Note:**
>
> If the outline contains multiple groups of level 0 members, Essbase adds the new member(s) to the first group that it encounters.

For this dimension build example, assume you want to add A100-20 and A100-99 as siblings of the lowest level members in the Measures dimension.

Assume you want to use the following text data file to add the new members:

```
100-10     Texas     A100-10     100     120     100
200-20     Texas     A100-99     111     154     180
```

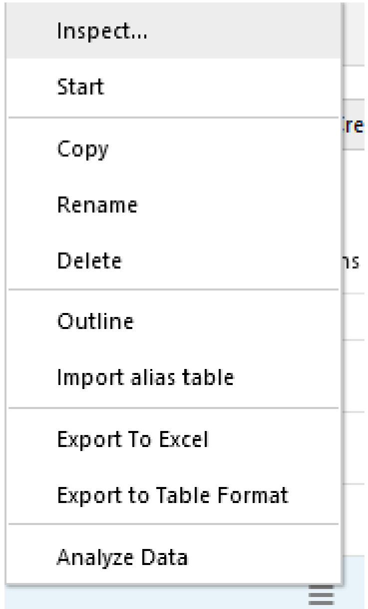
You can create the following rule file to build the dimensions using the **Add as a sibling at the lowest level** build method.

To recreate this example,

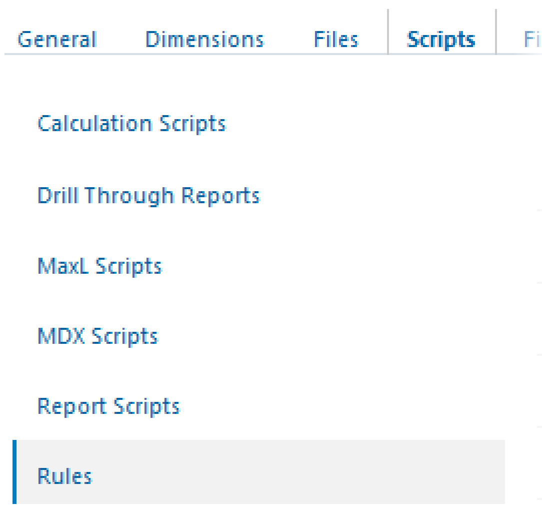
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `siblow.txt`, and upload it to the Sample Basic cube directory.

   ```
   100-10     Texas     A100-10     100     120     100
   200-20     Texas     A100-99     111     154     180
   ```

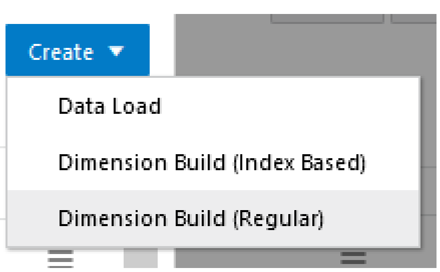
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.



6. Click **Create** and choose **Dimension Build (Regular)**.



7. In the New Rule dialog,
   a. Enter a rule name; for example, **siblow**.

b. For **Source Type** select **File**, click **Catalog** and navigate to `siblow.txt`.

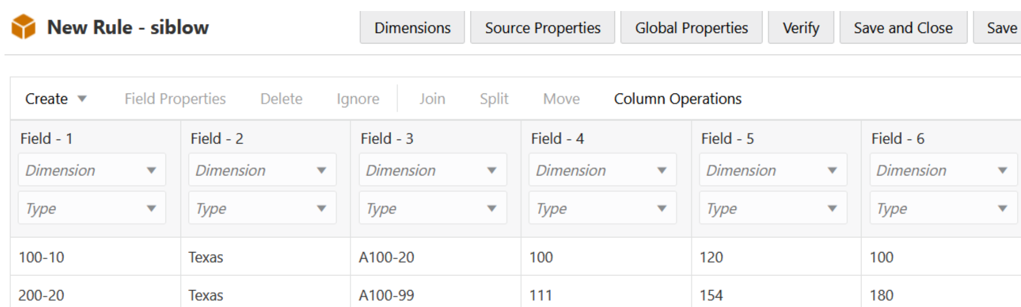
   When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.

c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.

d. Change the Delimiter value from Comma to **Tab**.

e. Click **Proceed**.

f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.



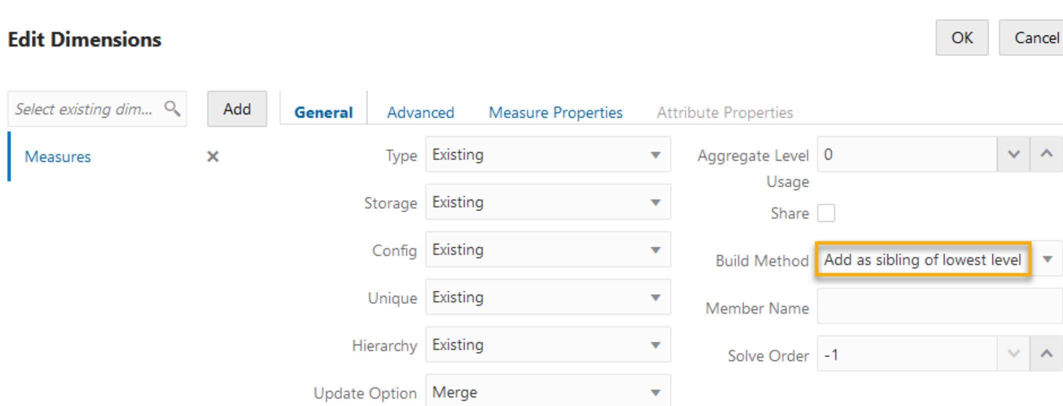
8. As this is a new rule, there are no dimensions associated yet.

   a. Click **Dimensions**.

   b. Select or type the dimension name **Measures**, and click **Add**.

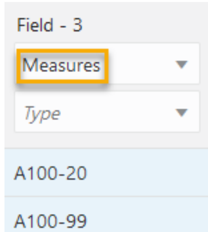
9. Now you need to define the build method and check other dimension build operational instructions. Click **Measures** to edit the properties.

10. Change **Build Method** to **Add as sibling of lowest level**, and leave the **Member Name** field below it blank.

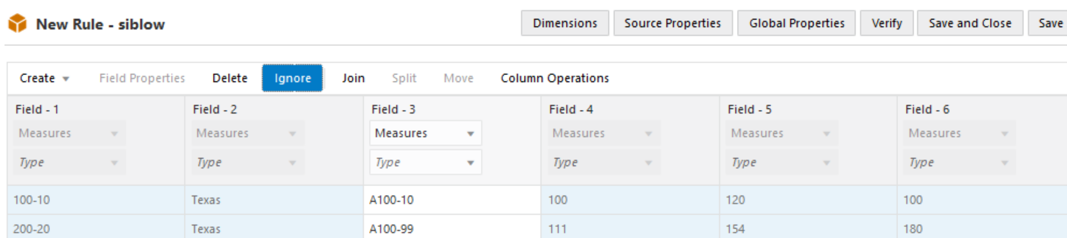


Click OK.

11. Click the *Dimension* selector in Field 3, and select **Measures**.



12. Leave the *Type* selector of Field 3 blank.
13. Select Fields 1 - 6. To do this, click on the inactive header area labeled Field 1, hold down Shift key, and click the Field 6 header area. Now hold the Control key and click field 3 to deselect it. Click **Ignore**.



As a result, this rule will only operate on the third field of each record. Ignoring fields can be helpful when you don't need to use every field in the source for your dimension build, yet you don't want to prepare different sources.

14. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.
15. Run the dimension build job.
16. View the outline, and inspect the children of **Margin** (in the Measures dimension) to confirm that the members were added.

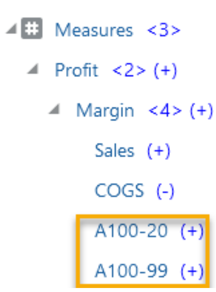


Table 15-2 Summary of Adding Members as Siblings of the Lowest Level

Field	Value	More Information
Measures dimension	Select the **Add as sibling of lowest level** build method	Select a Build Method

Table 15-2 (Cont.) Summary of Adding Members as Siblings of the Lowest Level

Field	Value	More Information
Field 3 (Measures)	• Do not select a field type for the field • Set the dimension for the field to Measures	Set Dimension Build Field Type Information
Fields 1, 2, 4, 5, and 6	Ignore the fields	You can ignore all fields of a specified column of the source data. See Ignoring Fields.

Adding Members to a Specified Parent

When you need to add new members to the Essbase outline but the source data does not specify their ancestors, design a dimension build rule that adds them to a specified parent member. Use the **Add as child of** dimension build method to add all new members as children of a specified parent.

After Essbase adds all new members to the outline, you can review the added members and move or delete them in the outline.

For this dimension build example, assume you want to add 600-54 and 780-22 as children of an existing outline member, NewProducts.

Figure 15-6 Example for Adding Members as a Child of a Specified Parent

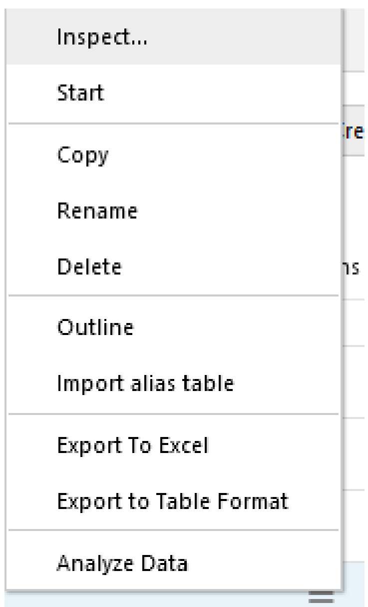
This use case calls for the **Add as child of** build method. In this build method, when Essbase encounters a new member in the source data, it adds the new member as a child of the parent name that you specify. The parent must be part of the outline before you start the dimension build.

To recreate this example,

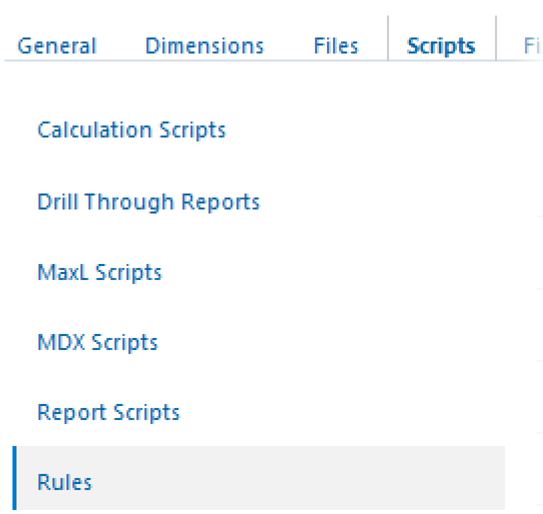
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `sibpar.txt`, and upload it to the Sample Basic cube directory.

```
600-54    Texas    Sales    100    120    100
780-22    Texas    Sales    111    154    180
```

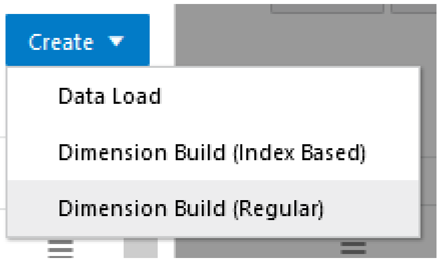
3. Open the Sample Basic outline for editing in the Essbase web interface, and, to the Product dimension, add a member named **NewProducts**.
4. Verify and save the outline.
5. On the **Applications** page, expand the application (Sample).
6. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



7. Select the **Scripts** tab, and then click **Rules**.

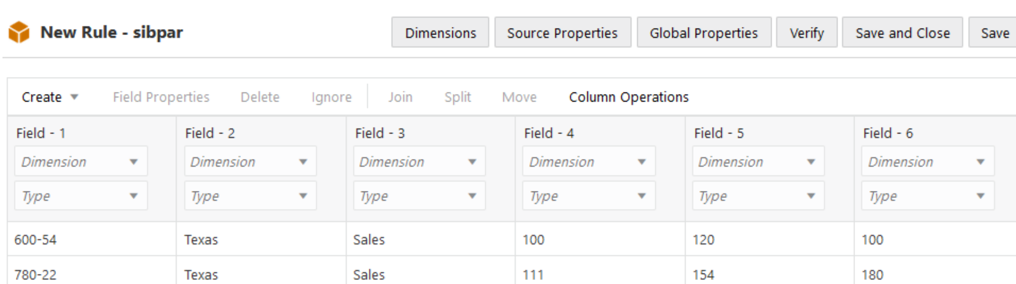


8. Click **Create** and choose **Dimension Build (Regular)**.

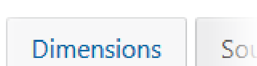


9. In the New Rule dialog,
   a. Enter a rule name; for example, **sibpar**.
   b. For **Source Type** select **File**, click **Catalog** and navigate to `sibpar.txt`.

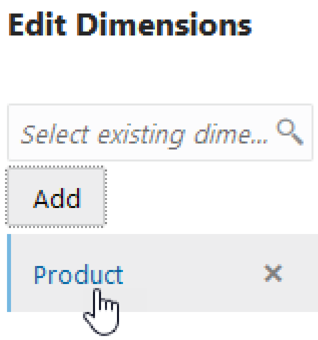
      When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.
   c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.
   d. Change the Delimiter value from Comma to **Tab**.
   e. Click **Proceed**.
   f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.



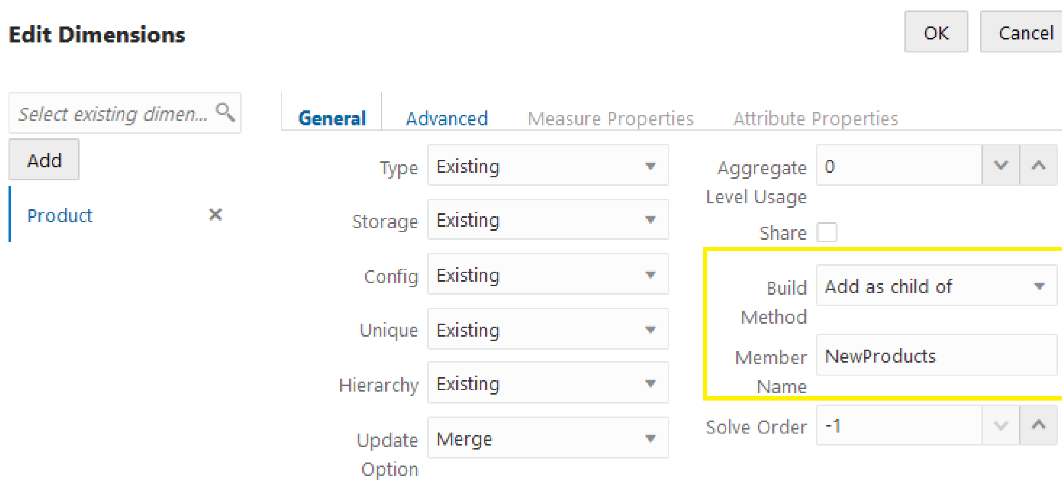
10. As this is a new rule, there are no dimensions associated yet.
    a. Click **Dimensions**.



    b. Select or type the dimension name **Product**, and click **Add**.
11. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.

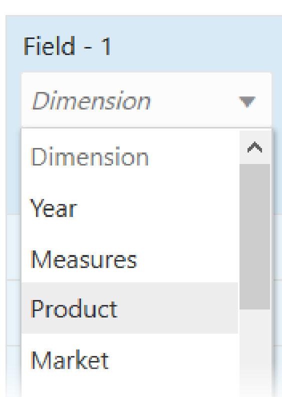


12. Change **Build Method** to **Add as child of**, and type the destination parent member name underneath it (**NewProducts**).

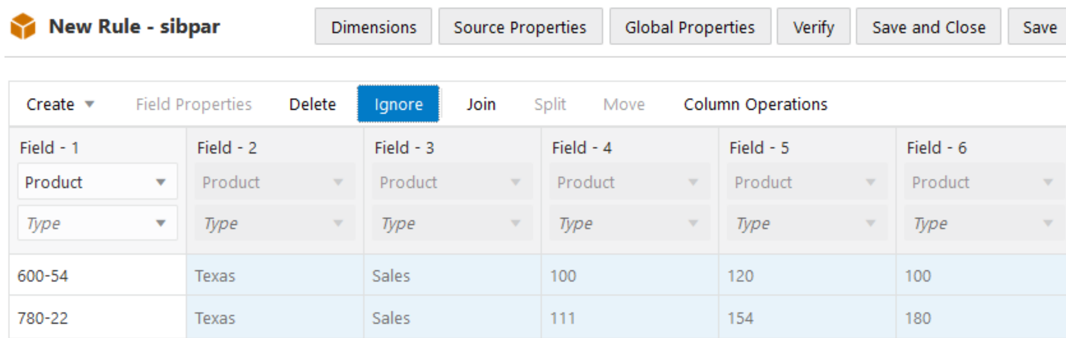


Click OK.

13. Click the *Dimension* selector in Field 1, and select **Product**.



14. Leave the *Type* selector of Field 1 blank.
15. Select Fields 2-6. To do this, click on the inactive header area labeled Field 2, hold down Shift key, and click the Field 6 header area. Click **Ignore**.



As a result, this rule will only operate on the first field of each record. Ignoring fields can be helpful when you don't need to use every field in the source for your dimension build, yet you don't want to prepare different sources.

16. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.

17. Run the dimension build job.

    If the jobs fails with the following error, it likely means that the outline does not contain the required parent member name (NewProducts, in this example).

    ```
    Member missing for add as child of dimension setting for dimension <name>
    ```

18. View the outline, and inspect the Product member **NewProducts** to confirm that the members were added.

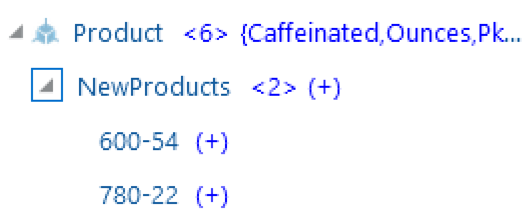


Table 15-3 Summary of Adding Members as a Child of a Specified Parent

Field	Value	More Information
Product dimension	Select the Add as child of build method	Select a Build Method Type `NewProducts` in the **Add as Child of** text box.
Field 1 (Product)	• Do not select a field type for the field • Set the dimension for the field to Product	Set Dimension Build Field Type Information
Fields 2 through 6	Ignore the fields	You can ignore all fields of a specified column of the source data. See Ignoring Fields.

Building Attribute Dimensions and Associating Attributes

When the data source for an Essbase dimension build contains attribute information, use one or more load rules to build attribute dimensions and to associate attributes with members of their base dimensions.

You can use load rules to build attribute dimensions dynamically, to add and delete members, and to establish or change attribute associations.

Working with attributes involves the following operations:

- If the base dimension does not exist, you must build it.
- You must build the attribute dimension.
- You must associate members of the base dimension with members of the attribute dimension.

You can use any of the following approaches to perform these operations:

- Build the base and attribute dimensions and perform the associations all simultaneously. Doing so, you use a single rules file to build the base dimension and one or more attribute dimensions to associate each attribute with the appropriate member of the base dimension. Because this approach uses a single rules file, it can be the most convenient. Use this approach if the base dimension does not exist and each source data record contains all attribute information for each member of the base dimension.
- Build the attribute dimension and perform the associations in one rules file. Assuming that the base dimension is built in a separate step or that the base dimension already exists, you can build an attribute dimension and associate the attributes with the members of the base dimension in one step. You need only to define the attribute associations in the rules file. See Associating Attributes in a Dimension Build.
- Build the attribute dimension and then perform the associations using separate rules files. Assuming that the base dimension is built in a separate step or that the base dimension already exists, you can build an attribute dimension and associate the attributes with the members of the base dimension in separate steps. Build the attribute dimension, and then associate the attribute members with members of the base dimension. Use this approach when you build numeric attribute dimensions that are multilevel or that have members that represent different-sized ranges.

See:

- Building Attribute Dimensions
- Associating Attributes in a Dimension Build
- Updating Attribute Associations
- Working with Numeric Ranges
- Reviewing the Rules for Building Attribute and Base Dimensions

Building Attribute Dimensions

Before you build attribute dimensions in a database, you must define the attribute member name formats for the outline. See Setting Member Names in Attribute Dimensions.

You can build attribute dimensions in one of the following ways:

- The same way in which you build standard dimensions.

See Data Load and Dimension Build Workflow.

- Simultaneously, as you associate attributes with members of the base dimension.

  See Associating Attributes in a Dimension Build.

Essbase does not support concurrent attribute association with the Add as build methods.

When you define the rules file for building attribute dimensions, specify the base dimension and the name of the attribute dimension file.

Associating Attributes in a Dimension Build

When a source of data contains attribute information, use one or more dimension builds to add attribute dimensions to the Essbase outline and to associate the attribute members with base dimension members.

Whether you build the attribute dimension and associate the attribute members with the members of the base dimension in one step or in separate steps, define the fields as described in this section.

Every record of the source data must include at least two columns: one for the member of the base dimension and one for the attribute value of the base dimension member. In the same source data record, you can include additional columns for other attributes that you want to associate with the member of the base dimension. You must position the base member fields before the corresponding attribute member fields.

Example

You can have Essbase build members of base dimensions and attribute dimensions, and make attribute associations, in the same dimension build job.

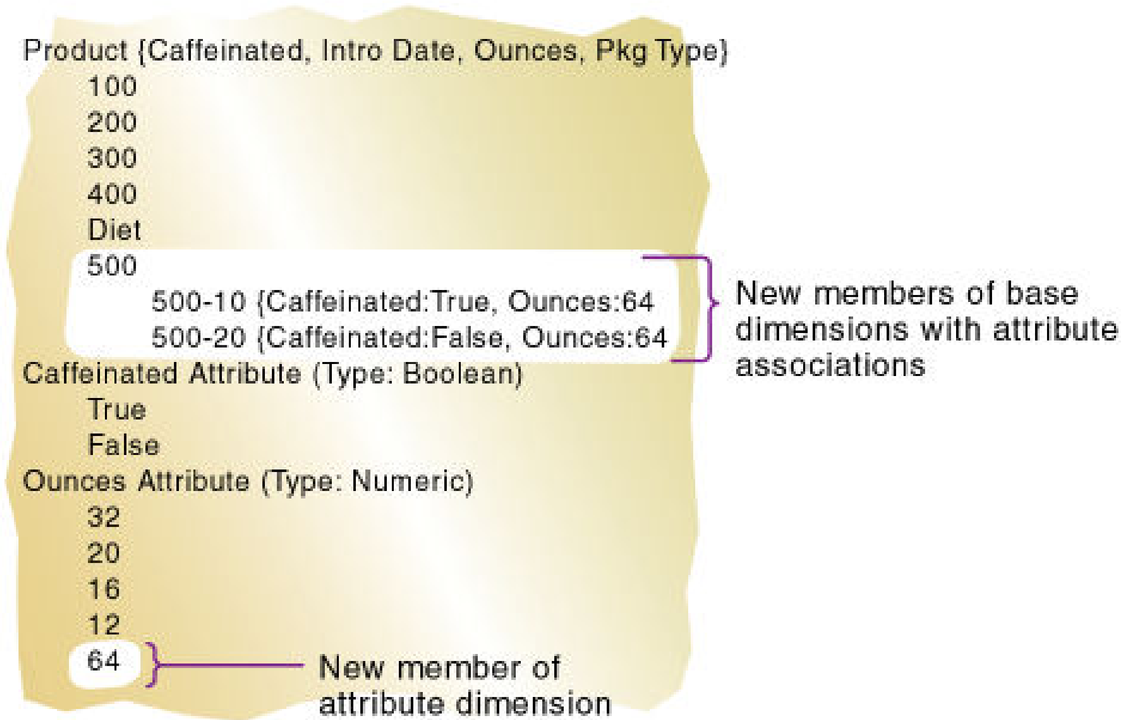
Assume you want to perform a dimension build that adds to Sample Basic new Product members 500-10 and 500-20, as children of 500.

In the same dimension build, you want to add a new numeric attribute member, 64, to the Ounces attribute dimension.

Finally, you want the dimension build to associate 500-10 and 500-20 with the 64 Ounces numeric attribute, and to associate 500-10 with the existing Caffeinated_True attribute, and 500-20 with Caffeinated_False. The text data file you are using for the dimension build looks as follows:

```
500     500-10     64     True
500     500-20     64     False
```

After you run the dimension build job, it builds members and associates attributes as shown in the outline below. Member 64 is a new member of the Ounces attribute dimension. Members 500, 500-10, and 500-20 are new members of the base dimension, Product, and are associated with member 64.

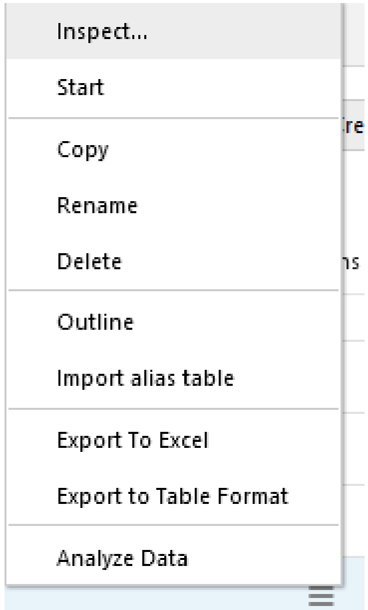


To recreate this example,

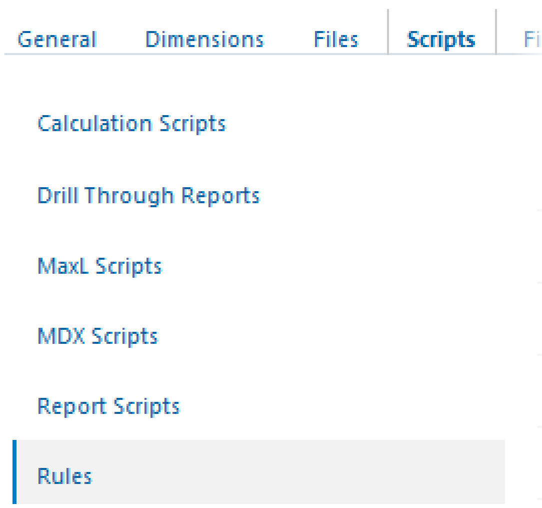
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `attrprod.txt`, and upload it to the Sample Basic cube directory.

```
500     500-10      64     True
500     500-20      64     False
```

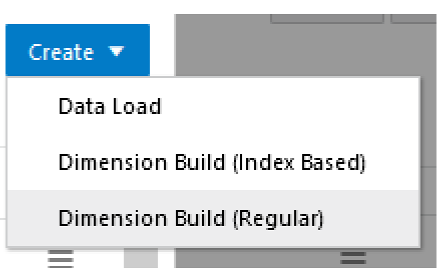
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.



6. Click **Create** and choose **Dimension Build (Regular)**.



7. In the New Rule dialog,
   a. Enter a rule name; for example, **attrprod**.

b. For **Source Type** select **File**, click **Catalog** and navigate to `attrprod.txt`.

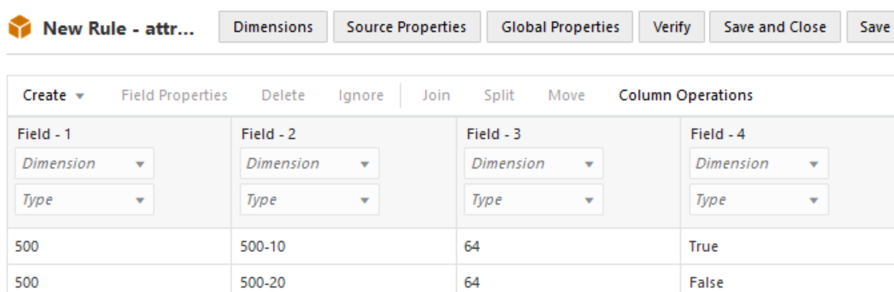
   When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.

c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.

d. Change the Delimiter value from Comma to **Tab**.

e. Click **Proceed**.

f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.

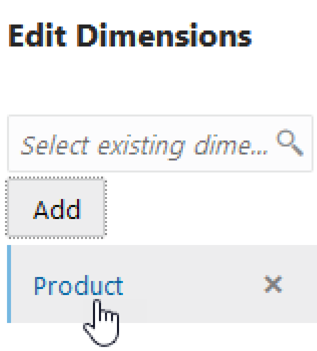


8. As this is a new rule, there are no dimensions associated yet.

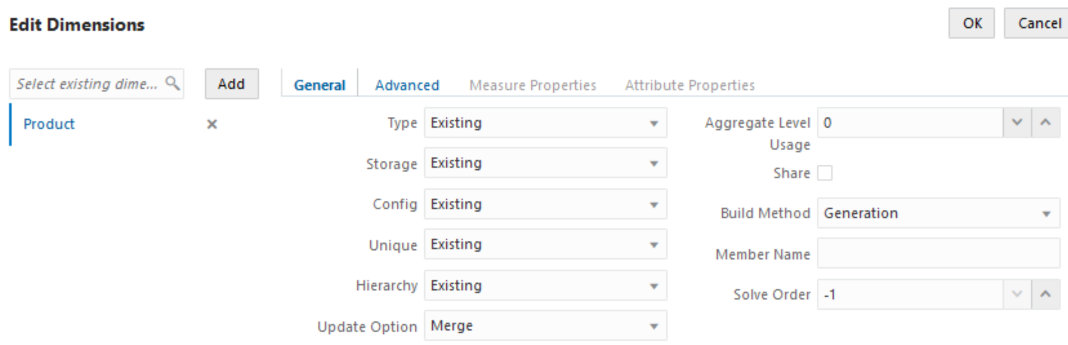
   a. Click **Dimensions**.

   b. Select or type the dimension name **Product**, and click **Add**.

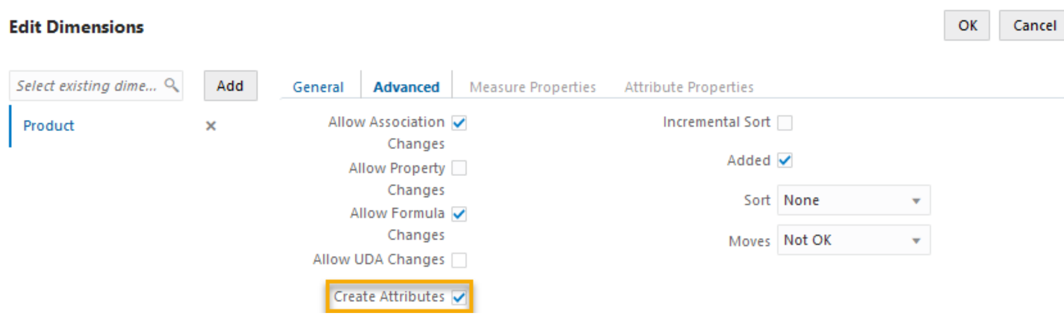
9. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.



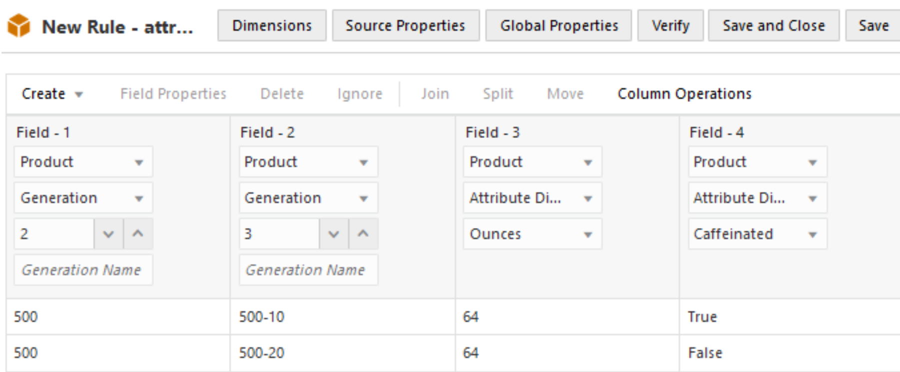
10. Ensure **Build Method** is **Generation**. Leave the **Member Name** field below it blank.



11. Click **Advanced**, and select Create Attributes.

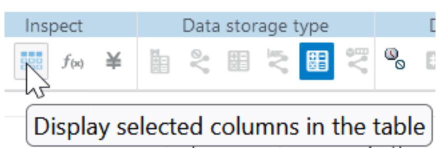


12. Click **OK**.
13. In the attrprod rule:
    a. In Field 1, select **Product** for the Dimension, and **Generation** for the Type. Increment the Generation field to **2**.
    b. In Field 2, select **Product** for the Dimension, and **Generation** for the Type. Increment the Generation field to **3**.
    c. In Field 3, select **Product** for the dimension, and **Attribute Dimension** for the Type. Below it, select **Ounces**.
       Field 3 contains members of the Ounces attribute dimension associated with the Product dimension. Because this field immediately follows the data field defined as the generation 3 member of the base dimension Product, Essbase associates the attribute member 64 with the 500-10 and 500-20 members.
    d. In Field 4, select **Product** for the dimension, and **Attribute Dimension** for the Type. Below it, select **Caffeinated**.
       Field 4 shows how to associate an attribute from an additional single-level attribute dimension. Because the base dimension is already specified, you need only to define an additional field for each attribute that you want to associate with the member of the base dimension.

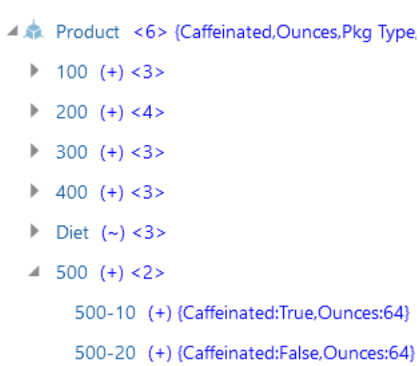


> **Note:**
>
> When you are working with numeric ranges, you may need to build attribute dimensions and perform associations in separate steps. See Working with Numeric Ranges.

14. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.
15. Run the dimension build job.
16. Open the outline. Before checking results of your dimension build, adjust the displayed columns to show attribute associations. On the outline toolbar, select the **Display selected columns in the table** icon.



17. Scroll down. For property name **Attributes**, click **Show in name**. Click **Apply and Close**.
18. Expand the **Product** dimension, and then expand the member, **500**.
After you run the dimension build job, it builds members and associates attributes as shown in the outline below. Member 64 is a new member of the Ounces attribute dimension. Members 500, 500-10, and 500-20 are new members of the base dimension, Product, and are associated with member 64.



Note:

You should not create an outline in which a shared member is located before its prototype member. If you do this, you will encounter an error while validating the outline.

Updating Attribute Associations

You can also use the rules file shown in Associating Attributes in a Dimension Build to change attribute associations. Ensure that you allow association changes.

For duplicate member situations, the field to which the attribute is associated must contain the qualified member name. See Building Qualified Member Names Through the Rule File.

Removing Attribute Associations

To remove attribute associations, use the same process as for updating them, plus the following steps:

- In the Dimension Build Properties tab of the Field Properties dialog box, select **Delete when the field is empty** for the attribute field. (This option is ignored if **Allow association changes** is not selected.)
- Leave the field empty or NULL in the data source.

Working with Numeric Ranges

In many cases, you can use one rules file in a dimension build operation to dynamically build attribute dimensions for numeric ranges and to associate the members of the base dimension with the ranges. In the following situations, however, you must use two rules files: one to build the attribute dimension and one to associate the attributes with the appropriate members of the base dimension:

- When the range size is different for different members.

  For example, you can define small ranges for towns and cities with smaller populations, larger ranges for mid-sized cities, and ranges greater than 1,000,000 for cities with large populations.
- When the ranges are members of a multilevel attribute dimension.

  For example, the Population attribute dimension can have level 1 members that categorize the population ranges as Towns, Cities, and Metropolitan Areas.

The Population attribute dimension shown below demonstrates both situations. Population is a multilevel, numeric attribute dimension with level 0 members representing ranges of different sizes.

Figure 15-7 Numeric Attribute Dimension with Different-Sized Ranges

```
Population
    Towns
        10000 (Alias: 1 to 10,000)
        50000 (Alias: 10,001 to 50,000)
        100000 (Alias: 50,001 to 100,000)
    Cities
        200000 (Alias: 100,001 to 200,000)
        400000 (Alias: 200,001 to 400,000)
        600000 (Alias: 400,001 to 600,000)
        800000 (Alias: 600,001 to 800,000)
        1000000 (Alias: 800,001 to 1,000,000)
    Metropolitan Areas
        2000000 (Alias: 1,000,001 to 2,000,00
        3000000 (Alias: 2,000,001 to 3,000,00
```

You must use one rules file to build the Population dimension and another rules file to associate the Population dimension members as attributes of members of the base dimension.

Building Attribute Dimensions that Accommodate Ranges

First, create a cube that uses the generation, level, or parent-child build method. To build the attribute dimension, specify the following information in the rule:

- The name of the attribute dimension and its associated base dimension.
- The fields for building the attribute dimension.

  See Set Dimension Build Field Type Information.

The source data must be in attribute sequence, in ascending order. If ranges have different sizes, the source data must include a record for every attribute range.

> **Note:**
>
> In later builds, you cannot insert attribute members between existing members.

To use the generation method to build the outline, you must sequence the source data in ascending sequence, based on the numeric attribute value. Define the fields in the rule as shown below.

The following example is based on Sample Basic, exported to an application workbook using the Generation build method, and re-imported.

Figure 15-8 Rule for Building a Numeric Attribute Dimension with Ranges

Field - 1	Field - 2	Field - 3
Population	Population	Population
Generation	Generation	Alias
2	3	
Generation Name	Generation Name	
Towns	10000	1 to 10,000
Towns	50000	10,001 to 50,000
Towns	100000	50,001 to 100,000
Cities	200000	100,001 to 200,000
Cities	400000	200,001 to 400,000
Cities	500000	400,001 to 600,000
Cities	800000	600,001 to 1,000,000
Cities	1000000	800,001 to 1,000,000
Metropolitan Areas	2000000	1,000,001 to 2,000,000
Metropolitan Areas	3000000	2,000,001 to 3,000,000

Associating Base Dimension Members with Their Range Attributes

After you build the numeric attribute dimension ranges, you need a rule to associate the members of the base dimension with their attributes. The source data includes fields for the members of the base dimension and fields for the data values that Essbase uses to associate the appropriate Population attribute.

Define the rule as shown below.

Figure 15-9 Rule for Associating Numeric Range Attributes

Field - 1	Field - 2	Field - 3
Market	Market	Market
Generation	Generation	Attribute Dime...
2	3	Population
Generation Name	*Generation Name*	
South	Albany, GA	117286
East	Boston, MA	3227707
East	Hartford, CT	1144574
West	Oakland, CA	2209629
Central	Rapid City, SD	87145
Central	St. Joseph, MO	97336
West	Tacoma, WA	657272

When you define the association field (for example, Field 3: *Market, Attribute Dimension, Population*), place the attribute members within a range.

Note:

The rule includes a city, Boston, whose population of 3,227,707 is outside the ranges of the attribute dimension, where the ranges extend only to 3,000,000. To allow for values in the source data that are outside the ranges in the attribute dimension, enter a range size, such as 1000000. Essbase uses the range size to add members to the attribute dimension above the existing highest member or below the existing lowest member, as needed.

> **Caution:**
>
> After you associate members of the base dimension with members of the attribute dimension, if you manually insert new members into the attribute dimension or rename members of the attribute dimension, you may invalidate existing attribute associations. Consider an example where numeric range attributes are defined as “Tops of ranges” and an attribute dimension contains members 100, 200, 500, and 1000. A base dimension member with the value 556 is associated with the attribute 1000. If you rename a attribute dimension member from 500 to 600, the base dimension member with the value 556 now has an invalid association. This base member is still associated with the attribute 1000 when it should be associated with the attribute 600. If you manually insert new members or rename existing members, to ensure that associations are correct, rerun the dimension build procedure and associate the base members with the changed attribute dimensions. For example, rerunning the attribute association procedure correctly associates the member of the base dimension with the value 556 with the new attribute 600.

Ensuring the Validity of Associations

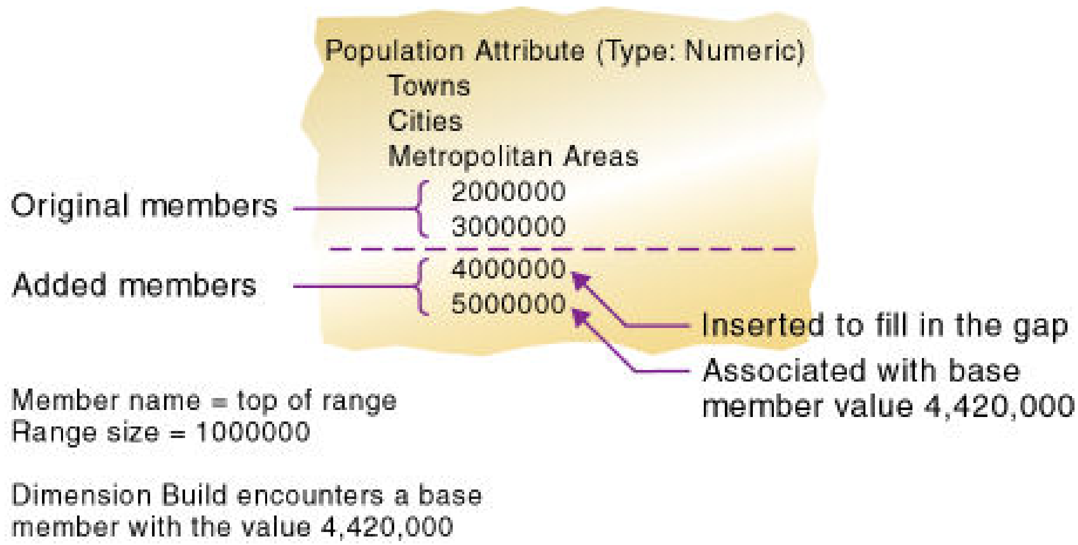
To ensure the validity of attribute associations, you must select the correct dimension building options and perform the builds in the proper sequence.

- **Adding or Changing Members of the Attribute Dimension:** After you associate members of a base dimension with their numeric attribute ranges, if you manually insert new members or rename existing members in the attribute dimension, ensure that associations between attributes and base members are correct by performing one of the following tasks:
  - Rerun the dimension build procedure that associates the base members with the changed attribute dimension.
  - Use Outline Editor to manually review and fix, as needed, the associations of all base dimensions.
- **Deleting Members from the Attribute Dimension:** You can delete all members of an attribute dimension so that you can rebuild the dimension with new data.
- **Adding Members to the Base Dimension:** You can use the same rules file to add new members to the base dimension and to associate the new members with their numeric range attributes simultaneously. Provide a value for the range size.

  If Essbase encounters a base dimension value that is greater than the highest attribute member by more than the range size or is lower than the lowest attribute member by more than the range size, it creates members in the attribute dimension to accommodate the out-of-range values.

  For example, in the image below, the numeric range attributes are defined as “Tops of ranges.” The highest value member of the Population attribute dimension is 3000000. If the source data includes a record with the population 4,420,000, and the range size is 1000000, Essbase adds two members to the attribute dimension, 4000000 and 5000000, and associates the base member with the value of 4,420,000 with the 5000000 attribute.

Figure 15-10   Dynamically Adding Attribute Range Members



When you add range members and base dimension members simultaneously, Essbase does not create aliases for the new members of the attribute dimension. If you want aliases that describe the range values for the new members of the attribute dimension, you must add the aliases in a separate operation.

Reviewing the Rules for Building Attribute and Base Dimensions

The information in this section describes areas unique to defining and associating attributes through a dimension build.

Getting Ready

- Before running a dimension build, you must define the attribute member name formats for the outline.

  See Setting Member Names in Attribute Dimensions.

- Defining new attribute dimensions in a rules file is different from defining new standard dimensions in a rules file.

Defining Fields in Rules Files

Rules files that are used to build single-level attribute dimensions require fewer field types than rules files that build and associate members of multilevel attribute dimensions.

- For single-level attribute dimensions, define the field that contains the attribute values as the field to be associated with the members of the base dimension. A dimension build uses the defined field to add new members to the attribute dimension.

  See Associating Attributes in a Dimension Build.

- For multilevel attribute dimensions, Essbase requires fields that define each generation or level in the attribute dimension and fields that define the associations. Use the new field type, Attribute Parent, to identify fields that are parent members for the attribute members being associated.

Controlling Adding New Attribute Members

When Essbase encounters attribute data values that are not members of the attribute dimension, it automatically adds the values as new members.

To prevent adding new members to attribute dimensions, select the **Do not create members** option for the attribute dimension.

Controlling Associations

You can control the following associations:

- Making changes to attribute associations

  Select the **Allow association changes** option for the attribute dimension.
- Enabling automatic association of base members with attributes that represent ranges of values

  Define the size of the range.

  See Set Dimension Build Field Type Information.
- Concurrent attribute associations

  Use any build method except the **Add as build** methods.

  See Dimension Build Methods.

> **Note:**
>
> Because attributes are defined only in the outline, the data load process does not affect them.

Building Shared Members by Using a Rules File

The data associated with a shared member comes from a prototype member with the same name as the shared member. Because the shared member stores a pointer to data contained in the prototype member, the data is shared between the members and is stored only once.

For example, member 100-20 (Diet Cola) rolls up into the 100 family and into the Diet family.

Figure 15-11 Shared Members in the Sample.Basic Database

You can share members among as many parents as you want. Diet Cola has two parents (100 and Diet), but you can define it to roll up into more parents.

You can share members at multiple generations in the outline. In Figure 15-11, Diet Cola is shared by two members at generation 2 in the outline, but it can be shared by a member at generation 3 and a member at generation 4, as shown in Figure 15-14.

Creating shared members at different generations in the outline is easy in Outline Editor; creating shared members using dimension build is more difficult. You must pick the build method and format the data source carefully.

The following sections describe how to build shared members in the outline by using a data source and a rules file.

- Sharing Members at the Same Generation
- Sharing Members at Different Generations
- Sharing Non-Level 0 Members
- Building Multiple Roll-Ups by Using Level References
- Creating Shared Roll-Ups from Multiple Data Sources

> **Note:**
>
> You should not create an outline in which a shared member is located before its prototype member. If you do this, you will encounter an error while validating the outline.

Sharing Members at the Same Generation

Sharing members at the same generation in the Essbase outline is the simplest way to share members. To build shared members at the same generation, use dimension build methods to identify generation, level, or parent-child references when you design the dimension build rule.

Members that are shared at the same generation roll up into the same branch. In the example below, based on Sample Basic, the highlighted product member 100-20 (Diet Cola) is at Generation 2, and is shared by another parent member, Diet. Both parents (100 and Diet) roll up into the same branch (the Product dimension).

In the image below, all the highlighted product members beneath 100-400 are stored *prototype members*, and all the members under Diet are unstored *shared members*.

Figure 15-12   Sample Outline: Members Shared at the Same Generation

To share members at the same generation in a dimension build, choose a build method using generation, level, or parent-child references. For examples of each build method, see:

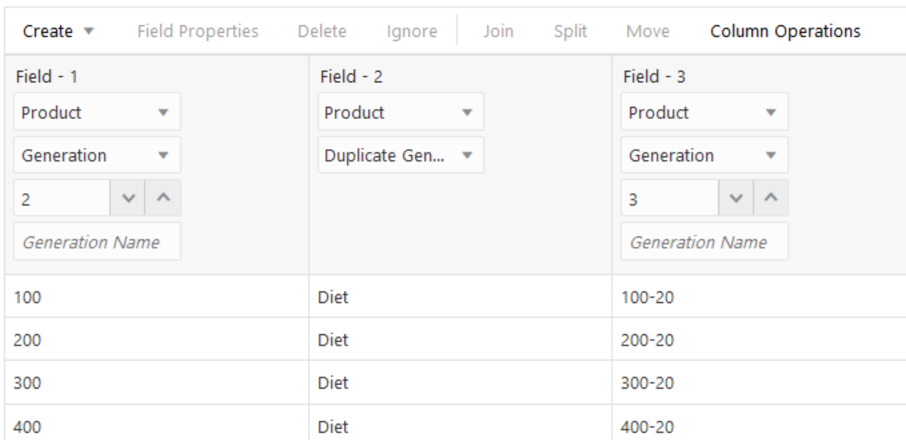
- Using Generation References to Create Same Generation Shared Members
- Using Level References to Create Same Generation Shared Members
- Using Parent-Child References to Create Same Generation Shared Members

Using Generation References to Create Same Generation Shared Members

When you design a rule to build shared members at the same generation in the Essbase outline using the Generation build method, place a duplicate generation after the prototype member parent in the dimension build rule.

To create shared member parents at the same generation by using the generation references build method, define the field type for the parent of the shared members as **Duplicate Generation**. A *duplicate generation*, sometimes known as an alternate roll-up, is a generation with shared members for children. Place it after the prototype members' parent (also known as the primary roll-up), as shown in the example.

The following dimension build rule shows the Diet parent and some shared members 100-20, 200-20, 300-20, and 400-20.

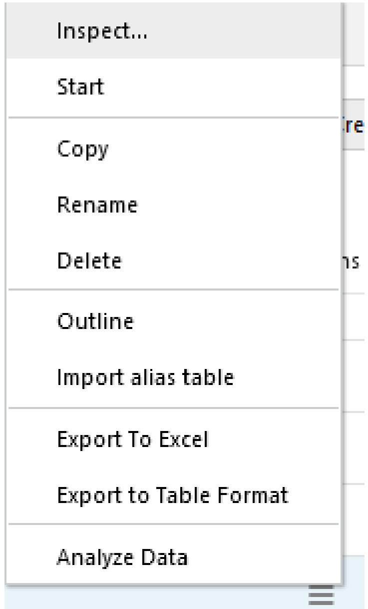


To recreate this example,

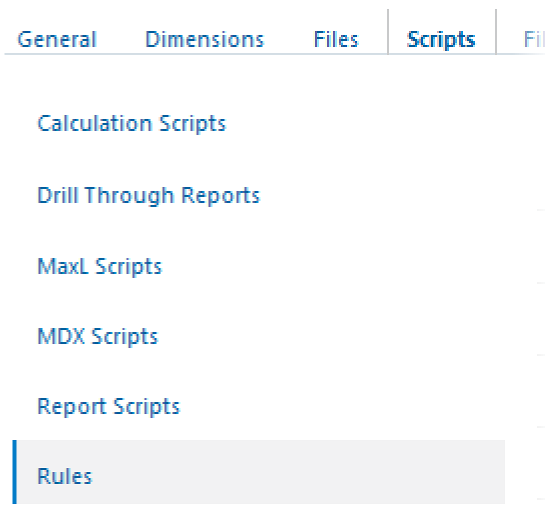
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `shgenref.txt`, and upload it to the Sample Basic cube directory.

```
100     Diet     100-20
200     Diet     200-20
300     Diet     300-20
400     Diet     400-20
```

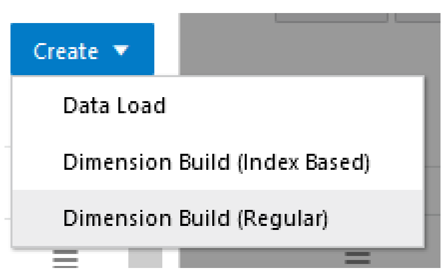
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.



6. Click **Create** and choose **Dimension Build (Regular)**.



7. In the New Rule dialog,
   a. Enter a rule name; for example, **shgenref**.

b. For **Source Type** select **File**, click **Catalog** and navigate to `shgenref.txt`.

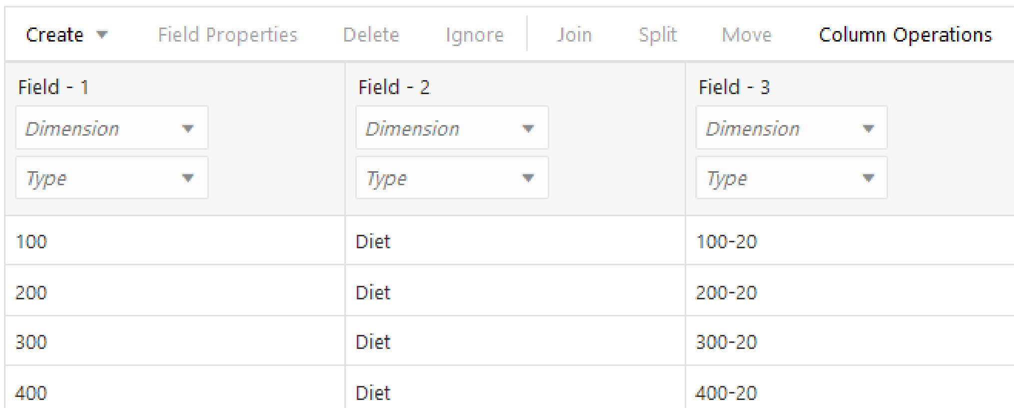
   When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.

c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.

d. Change the Delimiter value from Comma to **Tab**.

e. Click **Proceed**.

f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.

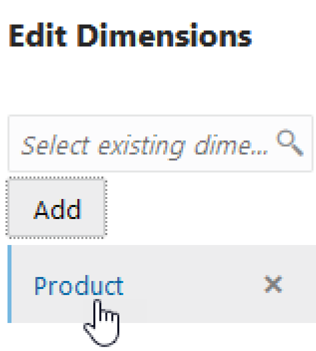


8. As this is a new rule, there are no dimensions associated yet.

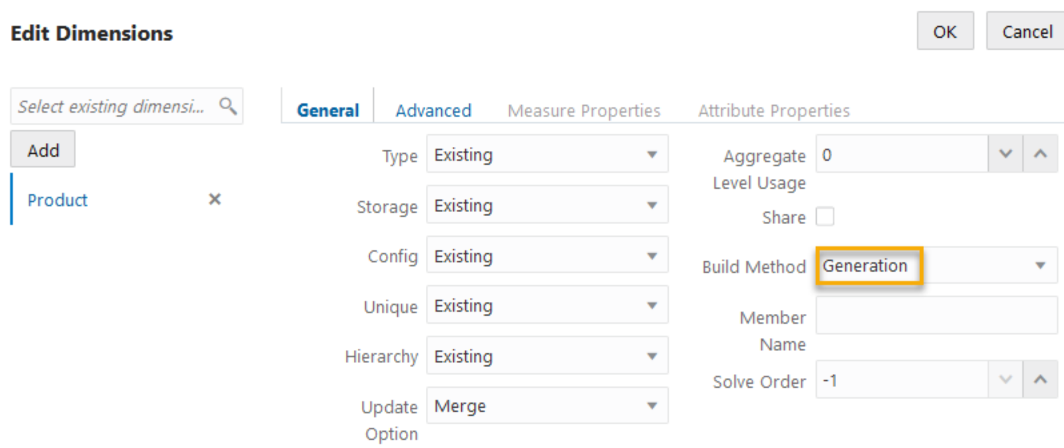
   a. Click **Dimensions**.

   b. Select or type the dimension name **Product**, and click **Add**.

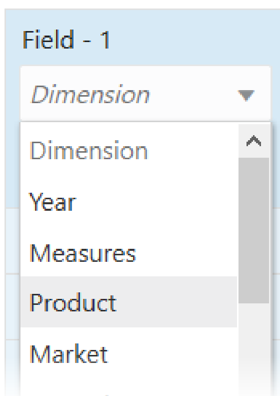
9. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.



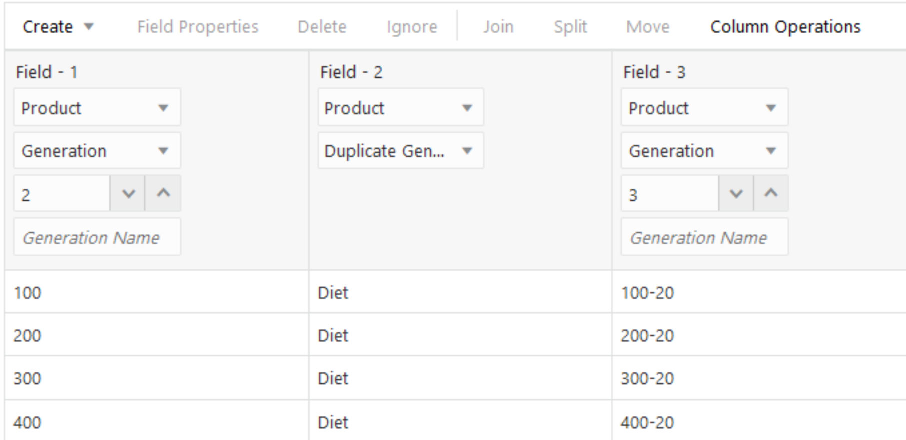
10. Ensure that **Build Method** is **Generation**, and click OK.



11. Click the *Dimension* selector in Field 1, and select **Product**.



12. Change the *Type* selector of Field 1 to **Generation**, and increment the generation number below it to **2**.
13. For Field 2, change the *Type* selector to **Duplicate Generation**.
14. For Field 3, change the *Type* selector to **Generation**, and increment the generation number below it to 3.
15. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.



16. Run the dimension build job.
17. View the outline, and inspect the Product member **Diet** to confirm that it has five shared members now (the dimension build added two more shared members, 300-20 and 400-20, to the pre-existing three).

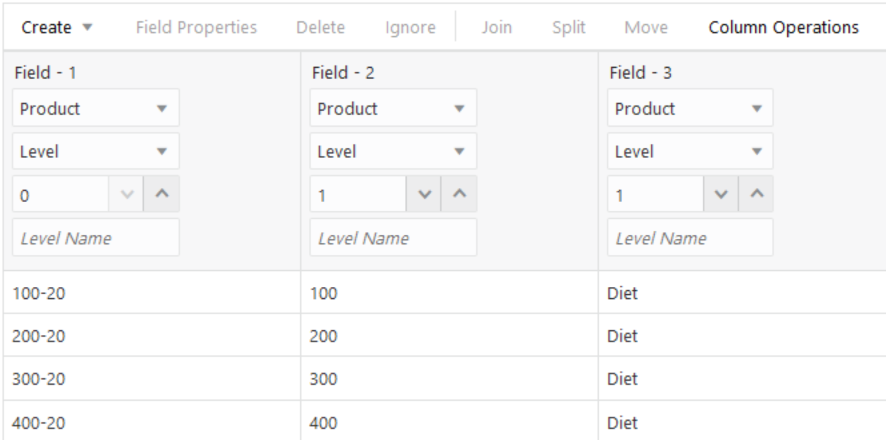
Using Level References to Create Same Generation Shared Members

When you design a rule to build shared members at the same generation in the Essbase outline using the Level build method, the prototype and shared members' parents must be in the same record in the dimension build rule.

To create shared members of the same generation by using the level references build method, first ensure that the primary and any secondary roll-ups (parent members) are specified in one record. You can specify as many secondary roll-ups as you want, as long as they are all in one record.

Define the field type for the shared member as Level. Then enter the level number in the field below. To create a shared member of the same generation, set the level number of the secondary roll-up to have the same number of levels as the primary roll-up. While processing the data source, Essbase creates a parent at the specified level and inserts the shared members under it.

The following example shows a rules file and data to create the shared members 100-20 (Diet Cola), 200-20 (Diet Root Beer), 300-20 (Diet Cream Soda), and 400-20 (Diet Fruit Soda).

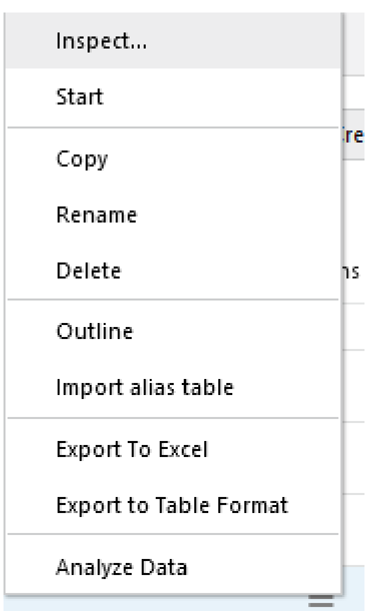


To recreate this example,

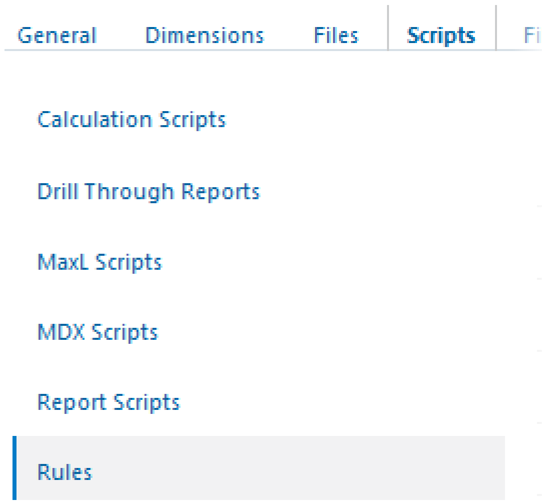
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `shlev.txt`, and upload it to the Sample Basic cube directory.

```
100-20    100    Diet
200-20    200    Diet
300-20    300    Diet
400-20    400    Diet
```

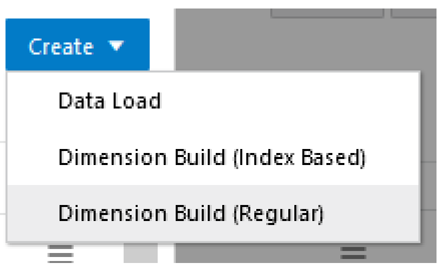
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.

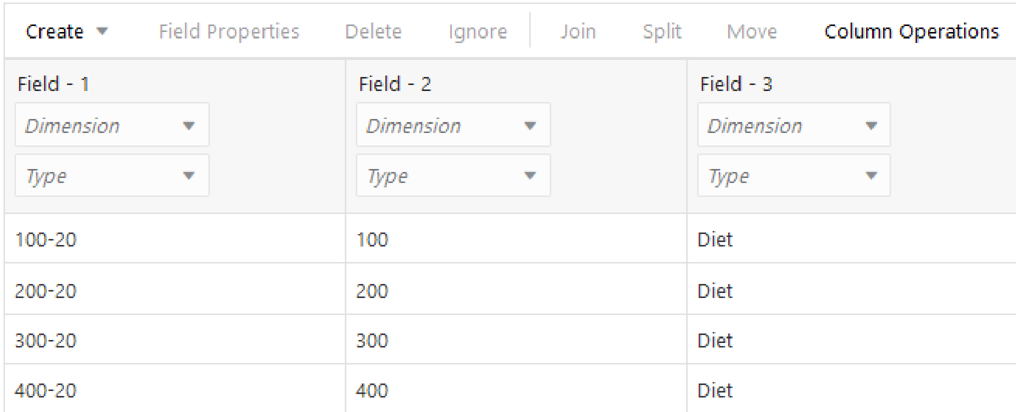


6. Click **Create** and choose **Dimension Build (Regular)**.

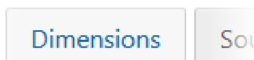


7. In the New Rule dialog,
   a. Enter a rule name; for example, **shlev**.
   b. For **Source Type** select **File**, click **Catalog** and navigate to `shlev.txt`.

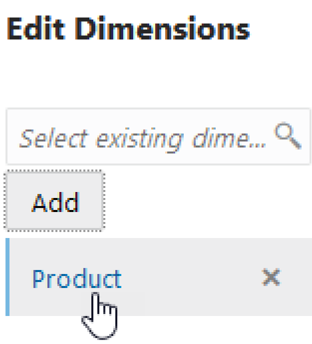
      When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.
   c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.
   d. Change the Delimiter value from Comma to **Tab**.
   e. Click **Proceed**.
   f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.



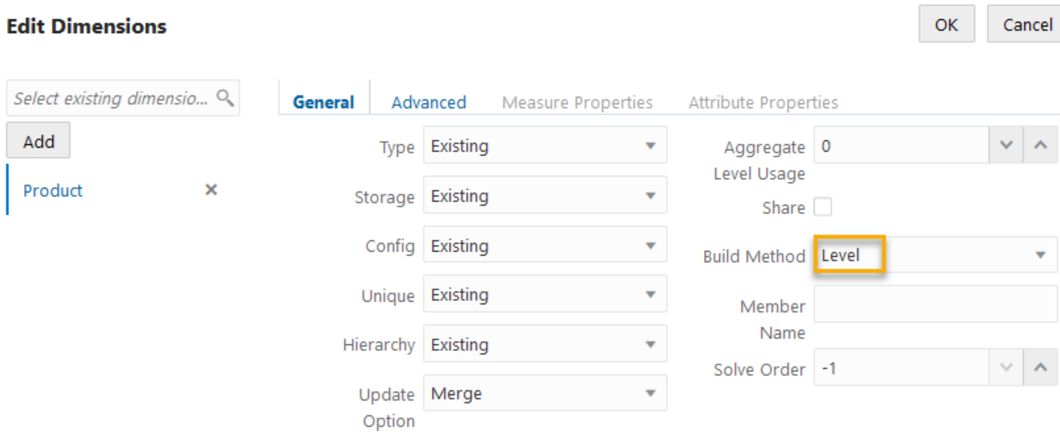
8. As this is a new rule, there are no dimensions associated yet.
   a. Click **Dimensions**.



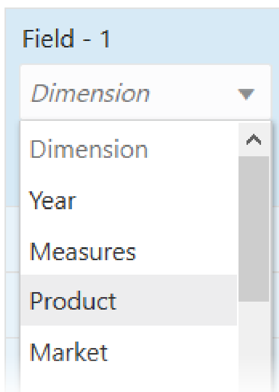
   b. Select or type the dimension name **Product**, and click **Add**.
9. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.



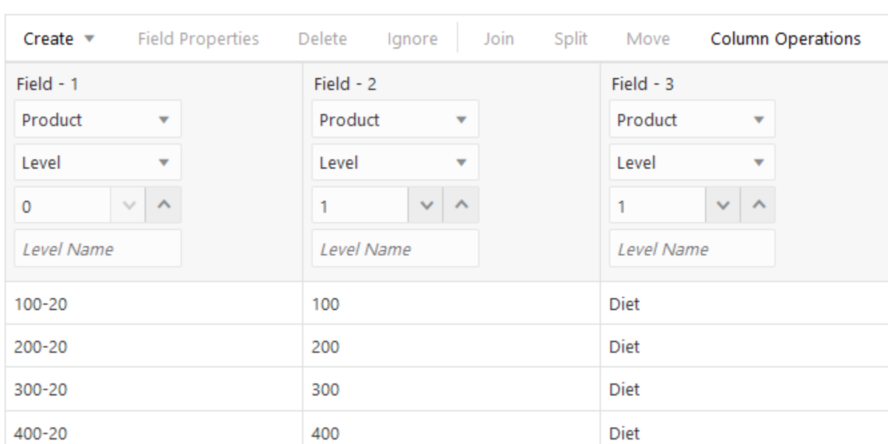
10. Change the **Build Method** to **Level**, and click OK.



11. Click the *Dimension* selector in Field 1, and select **Product**.



12. Change the *Type* selector of Field 1 to **Level**, and specify the level number below it as **0**.
13. For Fields 2 and 3, change the *Type* selector to **Level**, and specify the level numbers below them as **1**.
14. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.



15. Run the dimension build job.

16. View the outline, and inspect the Product member **Diet** to confirm that it has five shared members now (the dimension build added two more shared members, 300-20 and 400-20, to the pre-existing three).

Diet (~) <5>
- 100-20 (+) (Shared member)
- 200-20 (+) (Shared member)
- 300-30 (+) (Shared member)
- 300-20 (+) (Shared member)
- 400-20 (+) (Shared member)

Using Parent-Child References to Create Same Generation Shared Members

When you design a rule to build shared members at the same generation in the Essbase outline using the Parent-Child build method, define the Parent and Child field types in the dimension build rule, and ensure that sharing is enabled in the dimension properties.

When sharing is enabled, Essbase automatically creates duplicate members under a new parent as shared members.

Figure 15-13 Sample Rule: Members Shared at the Same Generation Using Parent-Child References

Create ▾ Field Properties Delete Ignore Join Split Move Column Operations

Field - 1	Field - 2
Product	Product
Parent	Child
100	100-20
200	200-20
300	300-20
400	400-20
Diet	100-20
Diet	200-20
Diet	300-20
Diet	400-20

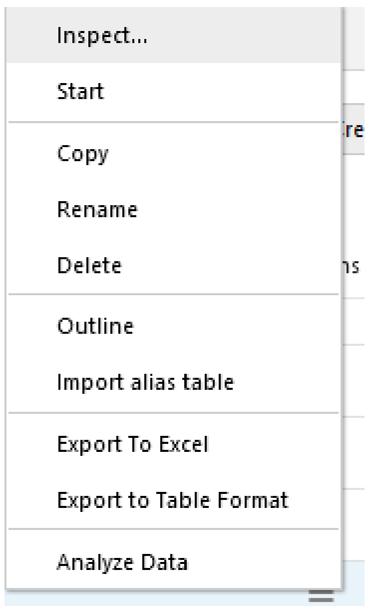
To recreate this example,

1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.

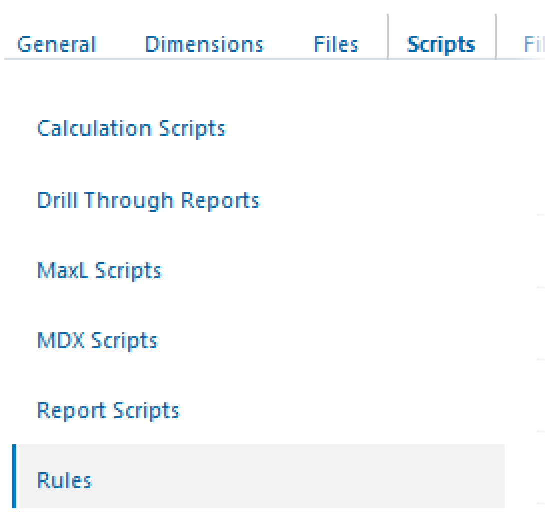
2. Create a tab-delimited data file like the following, name it `shparentchild.txt`, and upload it to the Sample Basic cube directory.

```
100     100-20
200     200-20
300     300-20
400     400-20
Diet     100-20
Diet     200-20
Diet     300-20
Diet     400-20
```

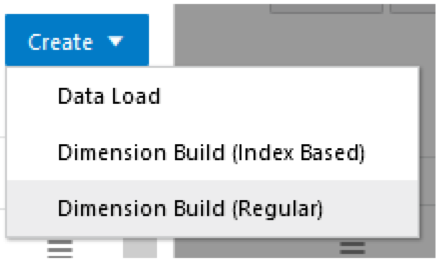
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.

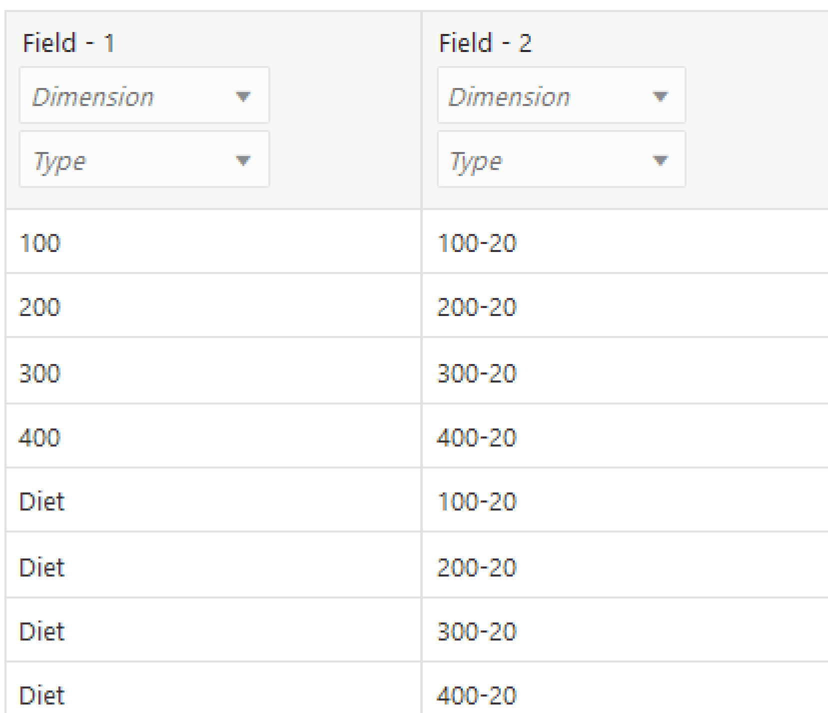


6. Click **Create** and choose **Dimension Build (Regular)**.



7. In the New Rule dialog,
   a. Enter a rule name; for example, **shparentchild**.
   b. For **Source Type** select **File**, click **Catalog** and navigate to `shparentchild.txt`.

      When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.
   c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.
   d. Change the Delimiter value from Comma to **Tab**.
   e. Click **Proceed**.
   f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.



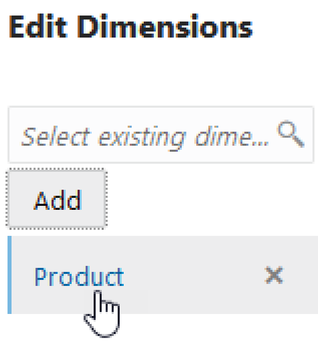
Field - 1 Dimension Type	Field - 2 Dimension Type
100	100-20
200	200-20
300	300-20
400	400-20
Diet	100-20
Diet	200-20
Diet	300-20
Diet	400-20

8. As this is a new rule, there are no dimensions associated yet.

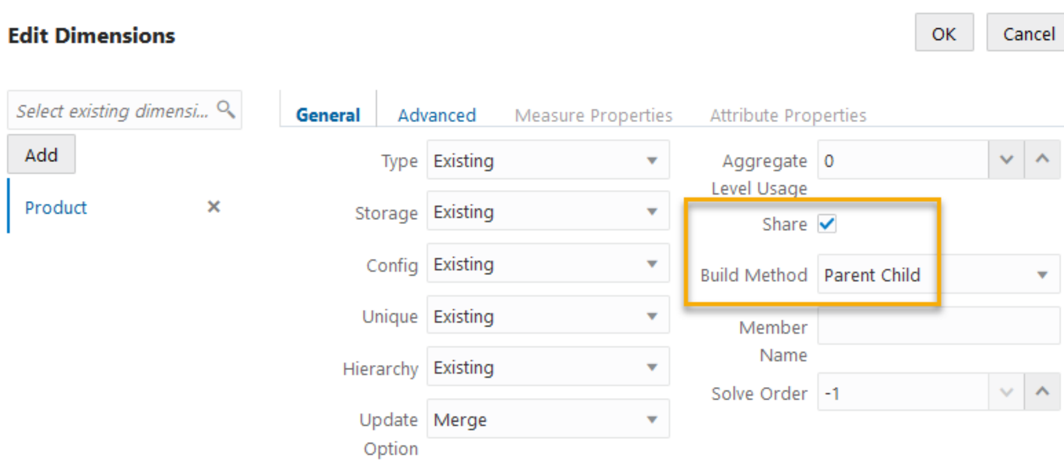
a. Click **Dimensions**.

b. Select or type the dimension name **Product**, and click **Add**.

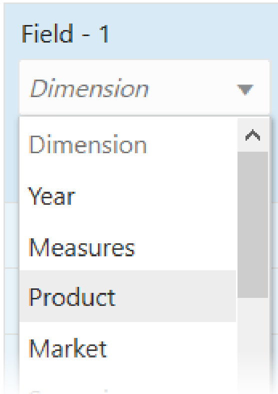
9. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.



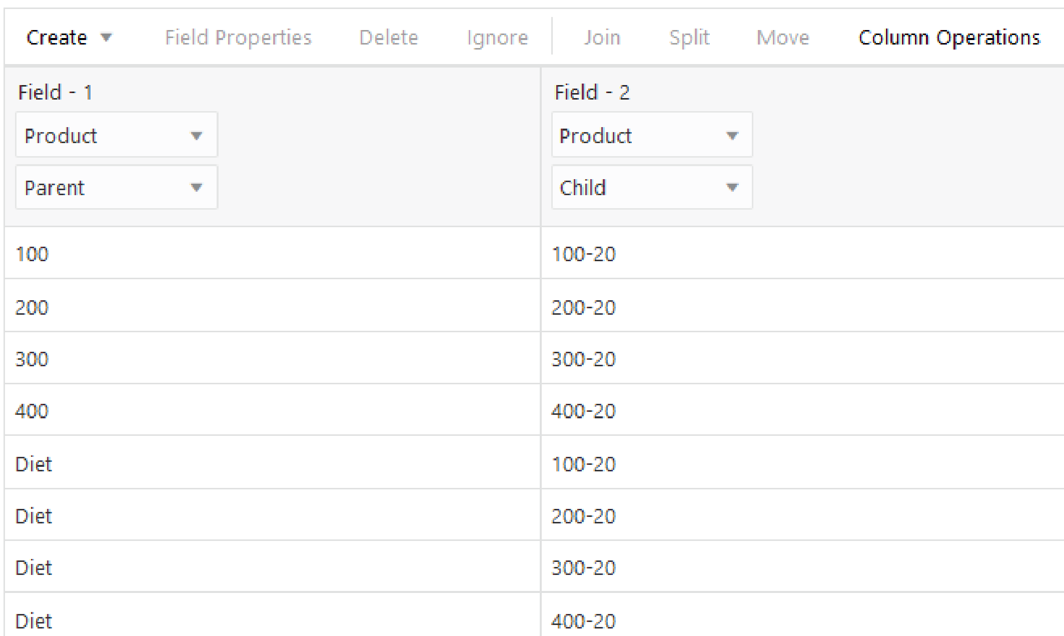
10. Change the **Build Method** to **Parent Child**, ensure that **Share** is enabled, and click OK.



11. Click the *Dimension* selector in Field 1, and select **Product**.



12. Change the *Type* selector of Field 1 to **Parent**.
13. For Field 2, change the *Type* selector to **Child**.



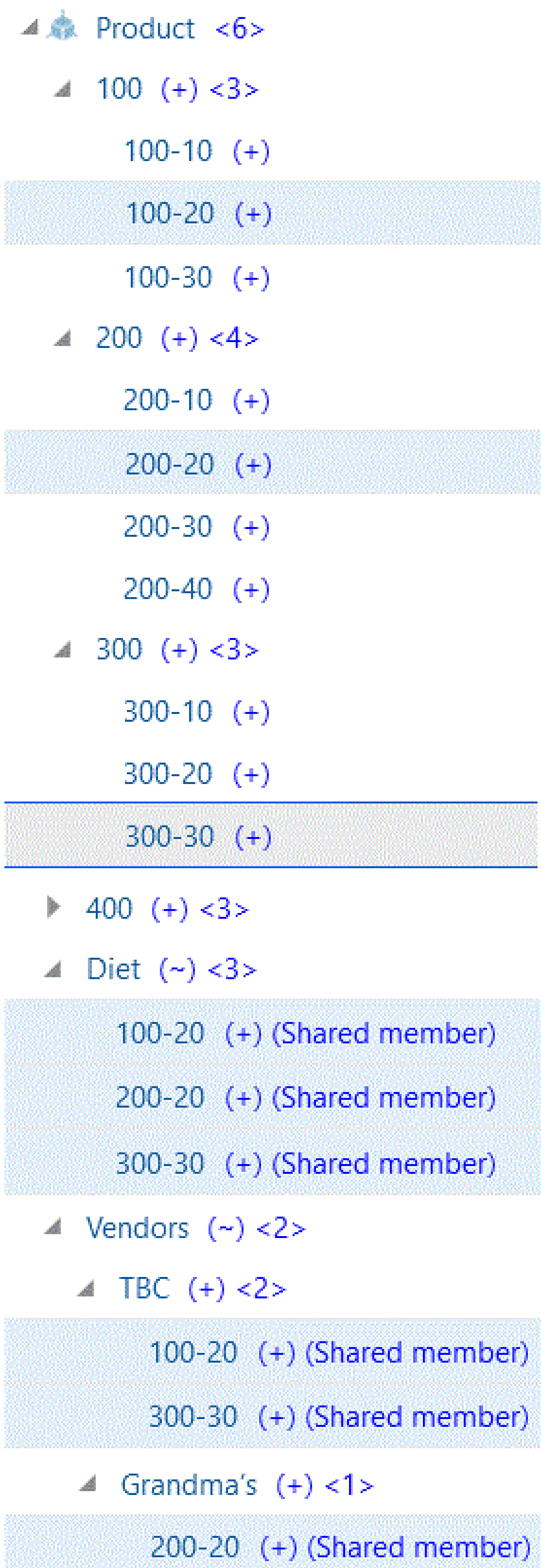
14. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.
15. Run the dimension build job.
16. View the outline, and inspect the Product member **Diet** to confirm that it has five shared members now (the dimension build added two more shared members, 300-20 and 400-20, to the pre-existing three).

Sharing Members at Different Generations

Using Essbase dimension build rules, you can build shared members at different generations. You can accomplish this using different build methods, including level references or parent-child references.

Sometimes you want shared members to roll up into parents that are at different generations in the outline. For example, in Figure 15-14, the shared members roll up into parents at generation 2 (Diet) and at generation 3 (TBC and Grandma’s). This outline assumes that TBC (The Beverage Company) buys some beverages from outside vendors: it buys 200-20 (Diet Root Beer) from a vendor named Grandma’s.

Figure 15-14   Sample Outline: Members Shared at Different Generations



- Using Parent-Child References to Create Different Generation Shared Members
- Using Level References to Create Different Generation Shared Members

Using Parent-Child References to Create Different Generation Shared Members

In an Essbase dimension build rule, to create shared members of different generations using the parent-child references build method, define the Parent and Child field types.

As long as sharing is enabled, Essbase automatically creates duplicate members under a new parent as shared members.

Figure 15-15 Sample Rule File: Members Shared at Different Generations Using Parent-Child References

Field - 1	Field - 2
Product	Product
Parent	Child
100	100-20
200	200-20
300	300-30
Diet	100-20
Diet	200-20
Diet	300-30
Vendors	TBC
Vendors	Grandma's
TBC	100-20
TBC	300-30
Grandma's	200-20

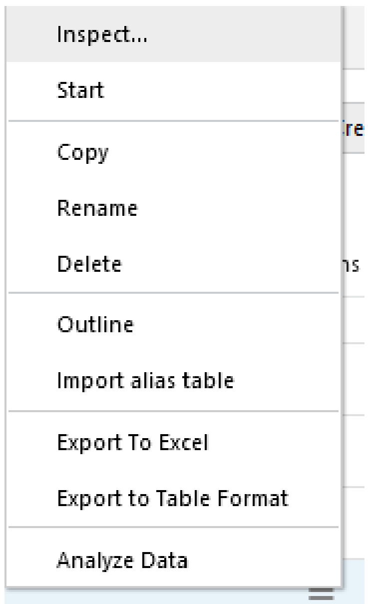
To recreate this example,

1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `shdiffgenparchild.txt`, and upload it to the Sample Basic cube directory.

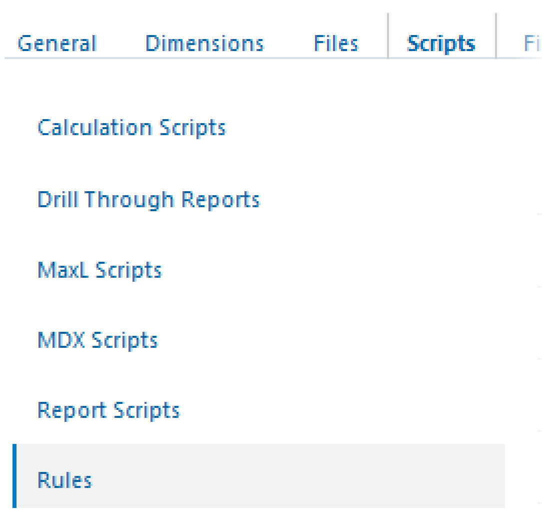
```
100     100-20
200     200-20
300     300-30
```

```
Diet    100-20
Diet    200-20
Diet    300-30
Vendors    TBC
Vendors    Grandma's
TBC    100-20
TBC    300-30
Grandma's    200-20
```

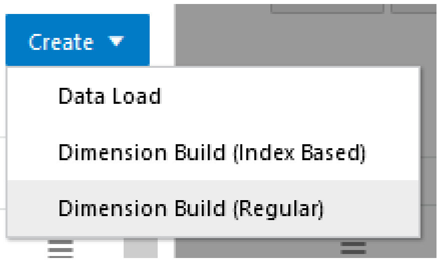
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.



6. Click **Create** and choose **Dimension Build (Regular)**.



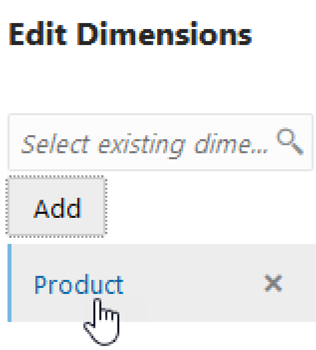
7. In the New Rule dialog,
   a. Enter a rule name; for example, **shdiffgenparchild**.
   b. For **Source Type** select **File**, click **Catalog** and navigate to `shdiffgenparchild.txt`.

      When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.
   c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.
   d. Change the Delimiter value from Comma to **Tab**.
   e. Increment the **Preview Data Count** to 11.
   f. Click **Proceed**.
   g. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.

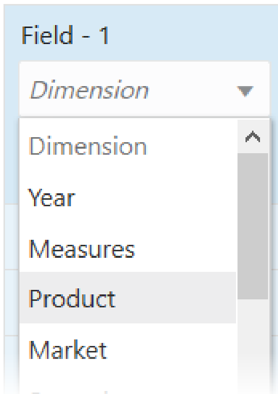
Field - 1 (Dimension, Type)	Field - 2 (Dimension, Type)
100	100-20
200	200-20
300	300-30
Diet	100-20
Diet	200-20
Diet	300-30
Vendors	TBC
Vendors	Grandma's
TBC	100-20
TBC	300-30
Grandma's	200-20

8. As this is a new rule, there are no dimensions associated yet.
   a. Click **Dimensions**.

   b. Select or type the dimension name **Product**, and click **Add**.
9. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.



10. Ensure that **Build Method** is **Parent Child**.
11. Click **Share**, and click OK.
12. Click the *Dimension* selector in Field 1, and select **Product**.



13. Change the *Type* selector of Field 1 to **Parent**.
14. For Field 2, change the *Type* selector to to **Child**.
15. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.

Field - 1 Product Parent	Field - 2 Product Child
100	100-20
200	200-20
300	300-30
Diet	100-20
Diet	200-20
Diet	300-30
Vendors	TBC
Vendors	Grandma's
TBC	100-20
TBC	300-30
Grandma's	200-20

16. Run the dimension build job.
17. View the outline, and inspect the Product dimension to confirm that it has new members Vendor, TBC, and Grandma's, with shared members that are also used in Diet.

- Diet (~) <3>
  - 100-20 (+) (Shared member)
  - 200-20 (+) (Shared member)
  - 300-30 (+) (Shared member)
- Vendors (~) <2>
  - TBC (+) <2>
    - 100-20 (+) (Shared member)
    - 300-30 (+) (Shared member)
  - Grandma's (+) <1>
    - 200-20 (+) (Shared member)

Using Level References to Create Different Generation Shared Members

In an Essbase dimension build rule, to create shared members of different generations using the level references build method, ensure that primary and secondary roll-ups are specified in one record. You can specify as many secondary roll-ups as you want, as long as they are all in one record.

Define the field type for the shared member as **Level**, then enter the level number. While processing the data source, Essbase creates a parent at the specified level and inserts the shared members under it.

For example, to share the products 100-20, 200-20, and 300-30 with a parent called Diet and two parents called TBC and Grandma's, use the sample data and rule shown below:

Figure 15-16 Sample Rule File: Members Shared at Different Generations Using Level References

Create ▾ Field Properties Delete Ignore Join Split Move Column Operations

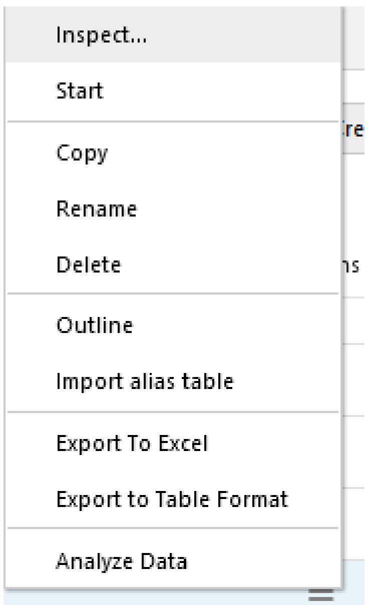
Field - 1	Field - 2	Field - 3	Field - 4	Field - 5
Product	Product	Product	Product	Product
Level	Level	Level	Level	Level
0	1	1	1	2
Level Name	*Level Name*	*Level Name*	*Level Name*	*Level Name*
100-20	100	Diet	TBC	Vendors
200-20	200	Diet	Grandma's	Vendors
300-30	300	Diet	TBC	Vendors

To recreate this example,

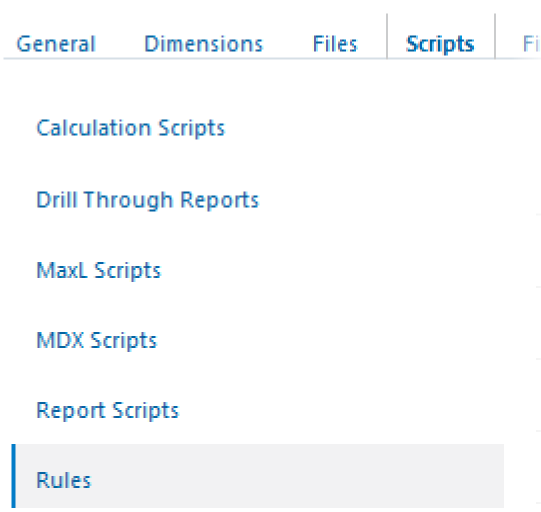
1. Import/build the Sample Basic cube, using the application workbook available from the gallery in the file catalog on the Essbase Server.
2. Create a tab-delimited data file like the following, name it `shdifflevref.txt`, and upload it to the Sample Basic cube directory.

```
100-20    100    Diet    TBC      Vendors
200-20    200    Diet    Grandma's    Vendors
300-30    300    Diet    TBC      Vendors
```

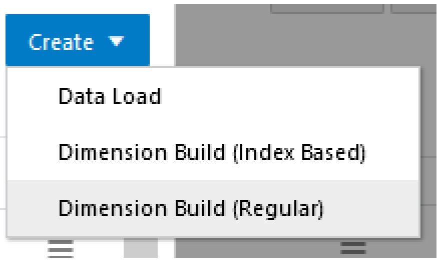
3. On the **Applications** page, expand the application (Sample).
4. From the **Actions** menu, to the right of the cube name (Basic), launch the inspector.



5. Select the **Scripts** tab, and then click **Rules**.

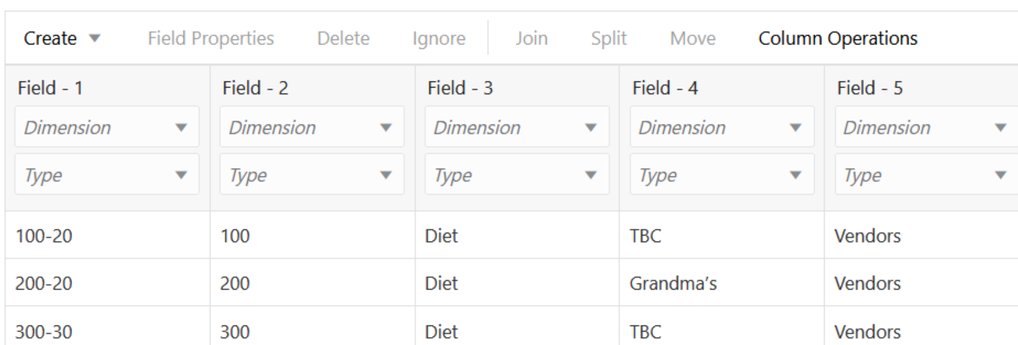


6. Click **Create** and choose **Dimension Build (Regular)**.



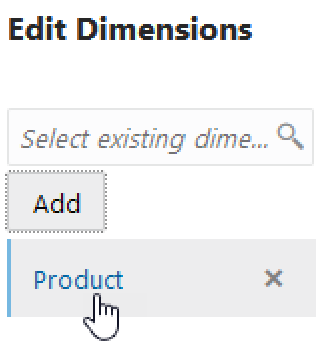
7. In the New Rule dialog,
   a. Enter a rule name; for example, **shdifflevref**.
   b. For **Source Type** select **File**, click **Catalog** and navigate to `shdifflevref.txt`.

      When you click **Catalog**, the file is expected to be located in the cube directory on the Essbase Server by default, so you must have already uploaded it using the **Files** section of the Essbase web interface. If the file is on your client machine, click **File Browser** to locate the file.
   c. Leave the **Header Record Number** and **Dimension Build Record Number** fields as 0. Header records are not useful for dimension builds.
   d. Change the Delimiter value from Comma to **Tab**.
   e. Click **Proceed**.
   f. The dimension build rule opens with undefined fields, and preview data (from the text file) populating the grid below the fields.

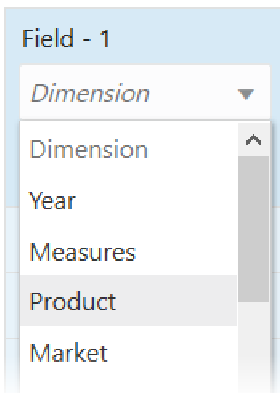


8. As this is a new rule, there are no dimensions associated yet.
   a. Click **Dimensions**.

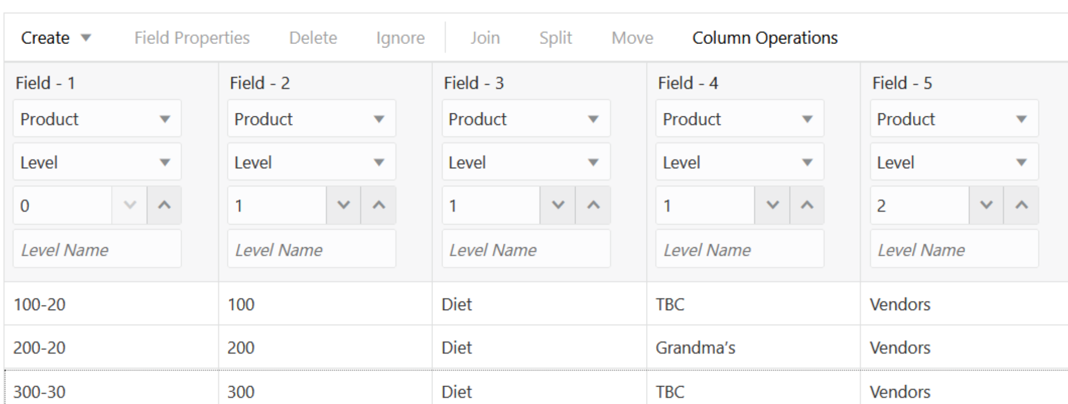
   b. Select or type the dimension name **Product**, and click **Add**.
9. Now you need to define the build method and check other dimension build operational instructions. Click **Product** to edit the properties.



10. Ensure that **Build Method** is **Level**, and click OK.
11. Click the *Dimension* selector in Field 1, and select **Product**.



12. Change the *Type* selector of Field 1 to **Level**, and enter the level number below it as **0**.
13. For Fields 2-4, change the *Type* selector to to **Level**, and enter the level numbers as **1**.
14. For Field 5, change the *Type* selector to **Level**, and enter the level number as **2**.
15. Verify the rule, then save and close. If there are any errors, see Requirements for Valid Dimension Build Rule Files.



16. Run the dimension build job.

17. View the outline, and inspect the Product dimension to confirm that it has new members Vendor, TBC, and Grandma's, with shared members that are also used in Diet.

```
Diet (~) <3>
    100-20 (+) (Shared member)
    200-20 (+) (Shared member)
    300-30 (+) (Shared member)
Vendors (~) <2>
    TBC (+) <2>
        100-20 (+) (Shared member)
        300-30 (+) (Shared member)
    Grandma's (+) <1>
        200-20 (+) (Shared member)
```

Sharing Non-Level 0 Members

Sometimes you want to share non-level 0 members (members that are not at the lowest generation). For example, in Figure 15-17, 100, 200, and 300 are shared by TBC and Grandma's. This outline assumes that TBC buys some of its product lines from outside vendors; it buys 200 (all root beer) from a vendor named Grandma's.

Figure 15-17 Sample Outline: Non-Level 0 Members Shared at Different Generations

```
Product
    Soda
        100
            100-20
        200
            200-20
        300
            300-20
        Diet
            100-20 (Shared Member)
            200-20 (Shared Member)
            300-20 (Shared Member)
    Vendors
        TBC
            100 (Shared Member)
            300 (Shared Member)
        Grandma's
            200 (Shared Member)
```

Sharing non-level 0 members can be accomplished using parent-child references or level references.

Using Parent-Child References to Create Non-Level 0 Shared Members

The parent-child references build method is the most versatile for creating shared members. It does not have any restrictions on the position of the shared members in the outline, unlike the generation references and level references build methods.

To create non-level 0 shared members at the same generation using the parent-child references build method, define the Parent and Child field types. Ensure that Essbase is set up to allow sharing. When sharing is enabled, Essbase automatically creates duplicate members under a new parent as shared members.

Figure 15-18   Sample Rule File: Non-Level 0 Members Shared at the Same Generation Using Parent-Child References

Field - 1 Product Parent	Field - 2 Product Child
Soda	100
100	100-20
Soda	200
200	200-30
Soda	300
300	300-30
Diet	100-20
Diet	200-20
Diet	300-20
Vendors	TBC

Using Level References to Create Non-Level 0 Shared Members

To create shared non-level 0 members by using the level references build method, ensure that primary and secondary roll-ups are specified in one record. You can specify unlimited secondary roll-ups, as long as they are all in one record.

Define the field type for the parent of the shared member as duplicate level, and then enter the level number. To create a shared member of the same generation, set the level number of the secondary roll-up to have the same number of levels as the primary roll-up. While processing the data source, Essbase creates a parent at the specified level and inserts the shared members under it.

For example, to share the product lines 100, 200, and 300 with a parent called Soda and parents called TBC and Grandma's, use the sample data file and rule file shown below. This data source and rule file work only if the Diet, TBC, and Grandma's members exist in the outline. The Duplicate Level field is always created as a child of the dimension (at generation 2), unless the named level field already exists in the outline.

Figure 15-19 Sample Rules File: Non-Level 0 Members Shared at Different Generations Using Level References

Field - 1	Field - 2	Field - 3	Field - 4	Field - 5
Product	Product	Product	Product	Product
Level	Level	Level	Duplicate Level	Level
0	1	2		1
Level Name	*Level Name*	*Level Name*		*Level Name*
100-20	100	Soda	TBC	Diet
200-20	200	Soda	Grandma's	Diet
300-20	300	Soda	TBC	Diet

Building Multiple Roll-Ups by Using Level References

To enable the retrieval of totals from multiple perspectives, you can also put shared members at different levels in the outline by using the level references build method. The following example shows build instructions for levels in the Product dimension:

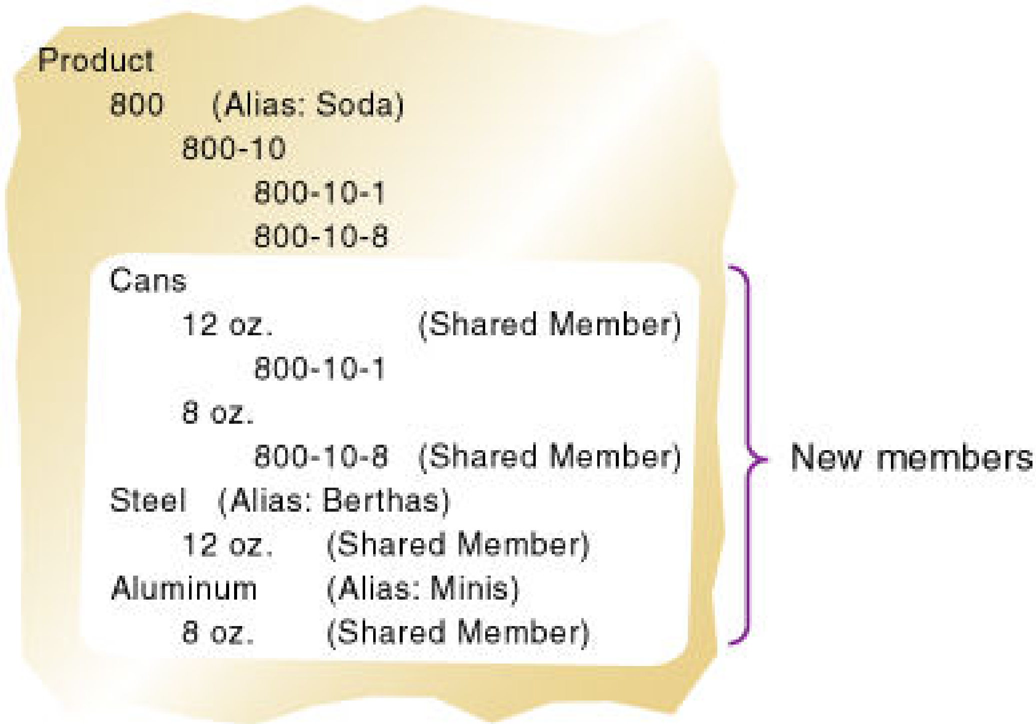
Create Properties Delete Ignore Join Split Move

Product	Product	Product	Product	Product	Product	Product	Product
Level	Level	Level	Alias	Level	Level	Duplicate Level	Duplicate Leve...
0	1	2		1	2		
Level Name	*Level Name*	*Level Name*		*Level Name*	*Level Name*		
800-10-1	800-10	800	Soda	12 oz.	Cans	Steel	Berthas
800-10-8	800-10	800	Soda	8 oz.	Cans	Aluminum	Minis

Because the record is so long, this second graphic shows the rules file scrolled to the right to show the extra members:

Running a dimension build using the rules file and data shown in the example, builds the outline shown below:

Figure 15-20   Sample Outline: Multiple Roll-Ups at Different Levels



This example enables analysis not only by package type (Cans), but also by packaging material (comparing sales of aluminum cans and steel cans).

Because Product is a sparse dimension, you can use an alternative outline design to enable retrieval of the same information. For example, consider creating a multilevel attribute dimension for package type with Steel and Aluminum as level 0 members under Can. For outline design guidelines, see Analyzing Database Design.

Creating Shared Roll-Ups from Multiple Data Sources

In many situations, the data for a dimension is in multiple data sources. If you are building dimensions from multiple data sources and want to create multiple roll-ups, load the first data source using the most appropriate build method, and then load all other data sources using the parent-child references build method. Ensure that Essbase is set up to allow sharing.

For example, using the following Product data source:

```
"Soft Drinks"   Cola
"Soft Drinks"   "Root Beer"
Cola            TBC
"Root Beer"     Grandma's
```

Essbase builds the outline illustrated in Figure 15-21:

Figure 15-21   Sample Outline: Soft Drinks

```
Product
    Soft Drinks
        Cola
            TBC
        Root Beer
            Grandma's
```

Then load the second data source below to relate the products to the vendors using the parent-child build method. Ensure that Essbase is set up to allow sharing.

```
Vendor   TBC
Vendor   Grandma's
```

Essbase builds the outline illustrated in Figure 15-22:

Figure 15-22   Sample Outline: Vendors (Shared Roll-Ups)

```
Product
    Soft Drinks
        Cola
            TBC
        Root Beer
            Grandma's
    Vendor
        TBC (Shared Member)
        Grandma's (Shared Member)
```

Building Duplicate Member Outlines

Duplicate member outlines contain multiple members with the same name, where the values are not shared. In unique member outlines, only shared members can have the same name. See Creating and Working With Duplicate Member Outlines.

The rules file enables you to set whether dimensions, levels, and generations in a duplicate member outline are unique or can include duplicate members.

Uniquely Identifying Members Through the Rules File

To ensure that duplicate member outline hierarchies are built correctly, use qualified member names in the data source or use the rules file to construct qualified member names from fields in the data source.

In most situations, the reference method and arrangement of fields provide enough information for Essbase to map data source columns to members in a duplicate member outline. The dimension build rules file for duplicate member is similar to the rules file for unique member outlines.

The following operations require more complex rules files for qualified member names:

- **Parent-child dimension builds:** The parent-child build method requires two fields, one for parent and one for child. For duplicate member situations the parent field must contain the qualified member name. Parent-child dimension builds on duplicate member outlines do not support attribute association. To ensure that attributes are associated with the correct members, perform the associations in a separate pass, using a generation-reference or level-reference rules file.

- **Association of attributes to existing members in the outline:** For duplicate member situations, the field to which the attribute is associated must contain the qualified member name.

Building Qualified Member Names Through the Rule File

If the source data does not contain the qualified member name as a field, you can use a rule to edit and join multiple fields, resulting in qualified member names.

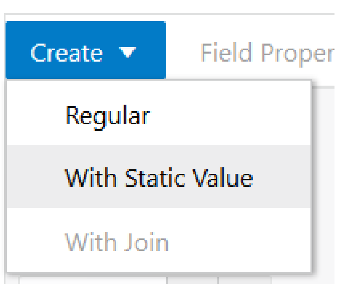
To create qualified member names, use static values to create brackets and periods, and moves/joins to build the names. For example, you can assign population attributes to existing city members in a duplicate member dimension. You can use join, move, and create operations to build the qualified name.

For example, assume the cities in the following source of data already exist in the outline. You want to associate with the cities the population attributes in the last column of this four-column source of data:

```
Central   "Kansas City"  Missouri    1070052
East      "New York"     Albany  8104079
```

Editing this source through the rule to build qualified names and sequence the field columns properly involves the following edits (make them in Preview mode so you can see the static values in the preview data):

1. Using **Create With Static Value**,



create one field each for the following text elements:

- `[`
- `].[`
- `].[`
- `]`
- Using Move operations, move the fields to the following sequence:

```
1   2          3     4          5     6             7    8
[   Central    ].[   Missouri   ].[   Kansas City   ]    706010
```

- Use Join operations to make the elements into two fields:

```
1                              2
 [Central].[Missouri].[Kansas City]  706010
```

16 Modeling Data in Private Scenarios

Using the Essbase scenario-management feature, you can test and model hypothetical data in a sandbox dimension without the storage overhead that would be required to replicate data from your working environment. Modeling scenarios applies only to hybrid mode databases.

This feature is documented at the following link: Model Data in Private Scenarios

17

Calculating Essbase Databases

An Essbase database (cube) contains input data, and values calculated from input data. Essbase offers two calculation methods: outline consolidation and calculation scripts.

- About Database Calculation
- About Multidimensional Calculation Concepts
- Setting the Default Calculation
- Calculating Databases
- Canceling Calculations
- Parallel and Serial Calculation
- Security Considerations

The information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases.

About Database Calculation

A database contains two types of values:

- Values that you enter, called *input data*
- Values that are calculated from input data

For example:

- You enter regional sales figures for a variety of products. You calculate the total sales for each.
- You enter the budget and actual values for the cost of goods sold for several products in several regions. You calculate the variance between budget and actual values for each product in each region.
- The database contains regional sales figures and prices for all products. You calculate what happens to total profit if you increase the price of one product in one region by 5%.

Small differences in the precision of cell values may occur between calculations run on different platforms, due to operating system math library differences.

> **Note:**
>
> Most computers represent numbers in binary, and therefore can only approximately represent real numbers. Because binary computers cannot hold an infinite number of bits after a decimal point, numeric fractions such as one-third (0.3333...) cannot be expressed as a decimal with a terminating point. Fractions with a denominator of the power of two (for example, 0.50) or ten (0.10) are the only real numbers that can be represented exactly. See IEEE Standard 754 for Floating-Point Representation (IEEE, 1985).

Essbase offers two methods for calculating a database:

- Outline calculation
- Calculation script calculation

The method that you choose depends on the type of calculation that you want to perform.

Outline Calculation

Outline calculation is the simplest calculation method. Essbase bases the calculation of the database on the relationships between members in the database outline and on any formulas that are associated with members in the outline.

For example, the image below shows the relationships between the members of the Market dimension in the Sample.Basic database. The values for New York, Massachusetts, Florida, Connecticut, and New Hampshire are added to calculate the value for East. The values for East, West, South, and Central are added to calculate the total value for Market.

Figure 17-1 Relationship Between Members of the Market Dimension

```
Market
    East (+) (UDAs: Major Market)
        New York (+) (UDAs: Major Market)
        Massachusetts (+) (UDAs: Major Market)
        Florida (+) (UDAs: Major Market)
        Connecticut (+) (UDAs: Small Market)
        New Hampshire (+) (UDAs: Small Market)
    West (+)
    South (+) (UDAs: Small Market)
    Central (+) (UDAs: Major Market)
```

The reason the child members of Market are *added* to get the parent values is that their member consolidation operators are set to (+).

The image below shows the Scenario dimension from the Sample.Basic database. The Variance and Variance % members are calculated by using the formulas attached to them.

Figure 17-2 Calculation of Variance and Variance %

```
Scenario (Label Only)
    Actual (+)
    Budget (~)
    Variance (~) (Dynamic Calc) (Two Pass Calc) @VAR(Actual, Budget);
    Variance % (~) (Dynamic Calc) (Two Pass Calc) @VARPER(Actual, Budget);
```

It may be more efficient to calculate some member combinations when you retrieve the data instead of calculating the member combinations during the regular database calculation. You can use dynamic calculations to calculate data at retrieval time. See Dynamically Calculating Data Values.

Calculation Script Calculation

Calculation script calculation is the second method of calculation. Using a calculation script, you can choose exactly how to calculate a database. For example, you can calculate part of a database or copy data values between members.

A calculation script contains a series of calculation commands, equations, and formulas. For example, the following calculation script increases the actual marketing expenses in the New York region by 5%.

```
FIX (Actual, “New York”)
   Marketing = Marketing *1.05;
ENDFIX;
```

See Develop Calculation Scripts for Block Storage Cubes.

About Multidimensional Calculation Concepts

The outline below, which is based on a simplified database, illustrates the nature of multidimensional calculations:

Figure 17-3 Calculating a Multidimensional Database

```
Accounts Accounts
     Margin (+)
          Sales (+)
          COGS (-)
     Margin% (~) (Two Pass Calc) Margin % Sales;
Time Time
     Qtr1 (+)
          Jan (+)
          Feb (+)
          Mar (+)
     Qtr2 (+)
     Qtr3 (+)
     Qtr4 (+)
Scenario (Label Only)
     Actual (+)
     Budget (+)
```

The database has three dimensions—Accounts, Time, and Scenario.

The Accounts dimension has four members:

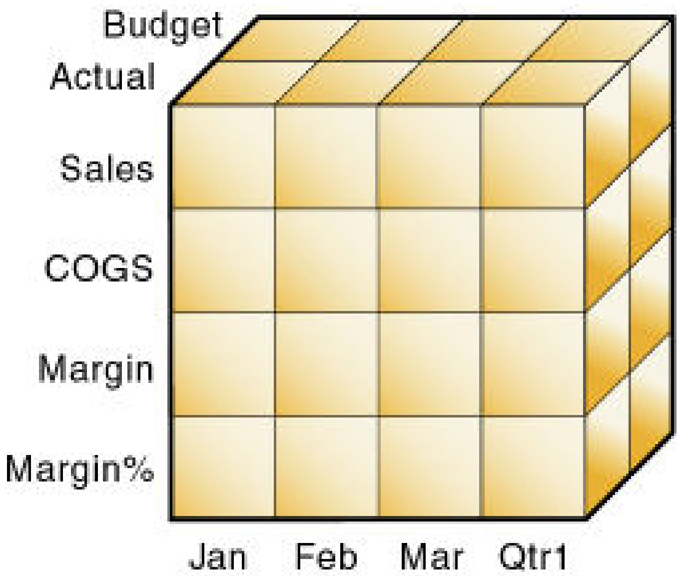
- Sales and COGS are input values
- Margin = Sales - COGS
- Margin% = Margin % Sales (Margin as a percentage of Sales)

The Time dimension has four quarters. The example displays only the members in Qtr1—Jan, Feb, and Mar.

The Scenario dimension has two child members—Budget for budget values and Actual for actual values.

The outline is illustrated as a three-dimensional cube:

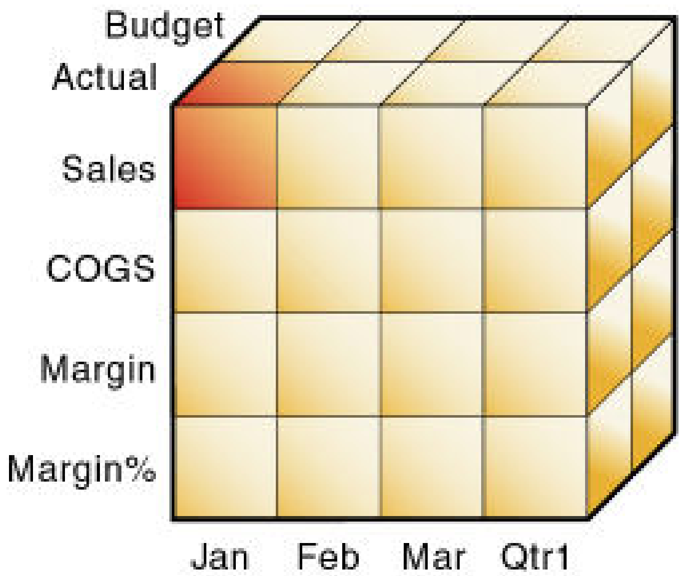
Figure 17-4   Illustration of a Three-Dimensional Database



An intersection of members (one member on each dimension) represents a data value; a data value is stored in one cell in the database. To refer to a specific data value in a multidimensional database, you must specify each member on each dimension. In Essbase, member combinations are denoted by a cross-dimensional operator (->). Create the cross-dimensional operator using a hyphen (-) and a greater-than symbol (>). Do not include a space between the cross-dimensional operator and members.

The single cell containing the data value for Sales, Jan, Actual, as shown in the image below, is written as Sales -> Jan -> Actual.

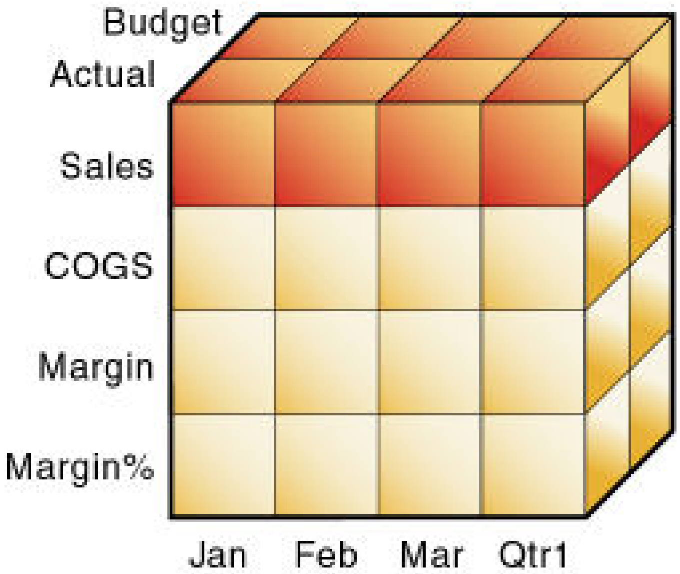
Figure 17-5 Sales, Jan, Actual Slice of the Database



When you refer to Sales, you are referring to a slice of the database containing eight values, as shown in the image above, which are:

- Sales -> Jan -> Actual
- Sales -> Feb -> Actual
- Sales -> Mar -> Actual
- Sales -> Qtr1 -> Actual
- Sales -> Jan -> Budget
- Sales -> Feb -> Budget
- Sales -> Mar -> Budget
- Sales -> Qtr1 -> Budget

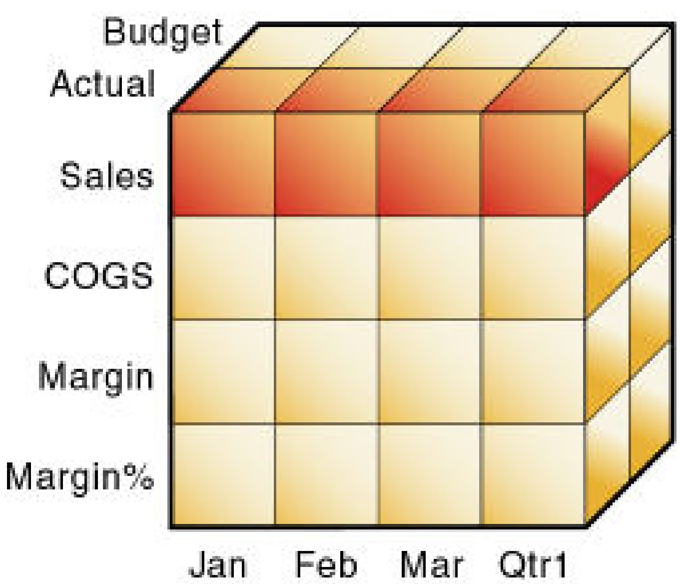
Figure 17-6 Sales, Actual, Budget Slice of the Database



When you refer to Actual Sales, you are referring to four values, as shown in the image below, which are:

- Sales -> Jan -> Actual
- Sales -> Feb -> Actual
- Sales -> Mar -> Actual
- Sales -> Qtr1 -> Actual

Figure 17-7 Actual, Sales Slice of the Database



When Essbase calculates the formula “Margin% = Margin % Sales,” it takes each Margin value and calculates it as a percentage of its corresponding Sales value.

Essbase cycles through the database and calculates Margin% as follows:

1. Margin -> Jan -> Actual as a percentage of Sales -> Jan -> Actual.

   The result is placed in Margin% -> Jan -> Actual.
2. Margin -> Feb -> Actual as a percentage of Sales -> Feb -> Actual.

   The result is placed in Margin% -> Feb -> Actual.
3. Margin -> Mar -> Actual as a percentage of Sales -> Mar -> Actual.

   The result is placed in Margin% -> Mar -> Actual.
4. Margin -> Qtr1 -> Actual as a percentage of Sales -> Qtr1 -> Actual.

   The result is placed in Margin% -> Qtr1 -> Actual.
5. Margin -> Jan -> Budget as a percentage of Sales -> Jan -> Budget.

   The result is placed in Margin% -> Jan -> Budget.
6. Essbase continues cycling through the database until it has calculated Margin% for every combination of members in the database.

See Defining Calculation Order.

Setting the Default Calculation

By default, the calculation for a database is a CALC ALL of the database outline. CALC ALL consolidates all dimensions and members and calculates all formulas in the outline.

You can, however, specify any calculation script as the default database calculation. Thus, you can assign a frequently used script to the database rather than loading the script each time you want to perform its calculation. If you want a calculation script to work with calculation settings defined at the database level, you must set the calculation script as the default calculation.

To set the default calculation, you can use the **alter database** MaxL statement.

Calculating Databases

Users with the Database Update role have access to run the default calculation on the cube, but no access to run any specific calculation scripts. Users with the Application Manager or Database Manager roles have calculation privileges and rights to execute all calculations. See About Calculating Data.

To calculate a database, you can use the **execute calculation** MaxL statement.

Canceling Calculations

To stop a calculation before Essbase completes it, click the **Cancel** button while the calculation is running.

When you cancel a calculation, Essbase performs one of the following operations:

- Reverts all values to their previous state
- Retains any values calculated before the cancellation

Parallel and Serial Calculation

Essbase supports parallel and serial calculations:

- *Serial calculation (default):* All steps in a calculation run on a single thread. Each task is completed before the next is started.
- *Parallel calculation:* The Essbase calculator can analyze a calculation, and, if appropriate, assign tasks to multiple CPUs (up to four).

See Using Parallel Calculation.

Security Considerations

To calculate a database, you must have the Database Update role for the database outline. You can calculate any value in the database, and you can calculate a value even if a security filter denies you read and update permissions. Carefully consider providing users with the Database Update role.

18
Develop Formulas for Block Storage Cubes

Formulas calculate relationships between Essbase members in a database (cube) outline. You can apply formulas to members in the outline, or include formulas in calculation scripts.

- Implement Essbase Formulas and Formula Calculations
- Essbase Formula Syntax
- Use Functions in Formulas
- Use Substitution Variables in Formulas
- Use Formulas on Partitions
- Display Formulas

The information in this chapter applies only to block storage and is not relevant to aggregate storage.

All of the examples in this chapter are based on the Sample.Basic application and database.

For more information about the functions referenced in this chapter, see Calculation Functions.

Implement Essbase Formulas and Formula Calculations

Essbase formulas calculate relationships between members. You can use them in an Essbase outline and in calculation scripts.

To implement formulas, you can:

- Apply formulas to members in the outline. Use this method if you do not need to control calculations carefully for accuracy or performance. This method limits formula size to less than 64 KB.

  See Use Functions in Formulas.

- Place formulas in a calculation script. Use this method if you need to control calculations carefully.

  See Formulas in Calculation Scripts.

The following image shows the Measures dimension from the Sample.Basic cube. The Margin %, Profit %, and Profit per Ounce members are calculated using the formulas applied to them.

Figure 18-1 Calculation of Margin %, Profit %, and Profit per Ounce

```
Ratios (~) (Label Only)
    Margin % (+) (Dynamic Calc) (Two Pass Calc) Margin % Sales;
    Profit % (~) (Dynamic Calc) (Two Pass Calc) Profit % Sales;
    Profit per Ounce (~) Profit/@ATTRIBUTEVAL(Ounces);
```

For formulas applied to members in the cube outline, Essbase calculates formulas when you perform the following actions:

- Run a default (CALC ALL) calculation.
- Run a calculation script that calculates the member containing the formula; for example, a CALC DIM of the dimension containing the member, or the member itself.

For a formula in a calculation script, Essbase calculates the formula when it occurs in the calculation script.

If a formula is associated with a dynamically calculated member, Essbase calculates the formula when the user requests the data values. In a calculation script, you cannot calculate a dynamically calculated member or make a dynamically calculated member the target of a formula calculation.

Using dynamically calculated members in a formula on the cube outline or in a calculation script can significantly affect calculation performance. Performance is affected because Essbase interrupts the regular calculation to perform the dynamic calculation.

You cannot use substitution variables in formulas that you apply to the outline.

Essbase Formula Syntax

You develop formulas on block storage cubes using Essbase calculation syntax, including member names, equations, values, operators, statement terminators, and quotation marks.

When you create member formulas, follow these rules:

- End each statement in the formula with a semicolon (;). For example:

  ```
  Margin % Sales;
  ```

- Use only saved outline member names. If a substitution variable is used for a member name, the substitution variable value must be a saved outline member name.
- Enclose a member name in double quotation marks (`""`) if the member name meets any of the following conditions:
  - Contains spaces. For example:

    ```
    "Opening Inventory" = "Ending Inventory" - Sales + Additions;
    ```

  - Is the same as an operator, function name, or keyword.

    See Naming Conventions in Calculation Scripts, Report Scripts, Formulas, Filters, and Substitution and Environment Variable Values.
  - Includes any nonalphanumeric character. For example, hyphens (-), asterisks (*), and slashes (/).
  - Is all numeric or starts with one or more numerals. For example, `"100"` or `"10Prod"`

  For a full list of member names that must be enclosed in quotation marks, see Naming Conventions in Calculation Scripts, Report Scripts, Formulas, Filters, and Substitution and Environment Variable Values.
- End each IF statement in a formula with an ENDIF statement.

For example, the following formula contains a simple IF...ENDIF statement. You can apply this formula to the Commission member in an outline:

```
IF(Sales < 100)
   Commission = 0;
ENDIF;
```

If you are using an IF statement nested within another IF statement, end each IF with an ENDIF. For example:

```
"Opening Inventory"
(IF (@ISMBR(Budget))
   IF (@ISMBR(Jan))
   "Opening Inventory" = Jan;
   ELSE
   "Opening Inventory" = @PRIOR("Ending Inventory");
   ENDIF;
ENDIF;)
```

- You do not need to end ELSE or ELSEIF statements with ENDIFs. For example:

```
IF (@ISMBR(@DESCENDANTS(West)) OR @ISMBR(@DESCENDANTS(East)
   Marketing = Marketing * 1.5;
ELSEIF(@ISMBR(@DESCENDANTS(South)))
   Marketing = Marketing * .9;
ELSE Marketing = Marketing * 1.1;
ENDIF;
```

> **Note:**
>
> If you use ELSE IF (with a space) rather than ELSEIF (one word) in a formula, you must supply an ENDIF for the IF statement.

- Ending ENDIF statements with a semicolon (;) is not required, but it is a good practice.

When writing formulas, you can check the syntax using the Formula Editor syntax checker. See Checking Formula Syntax.

See:

- Operators
- Dimension and Member Names
- Constant Values
- Nonconstant Values
- Basic Equations
- Checking Formula Syntax

Operators

You can use the following types of operators in formulas:

Table 18-1 List of Operator Types

Operator Type	Description
Mathematical	Perform common arithmetic operations. For example, you can add, subtract, multiply, or divide values. See Mathematical Operations.
Conditional	Control the flow of formula executions based on the results of conditional tests. For example, you can use an IF statement to test for a specified condition. See Conditional Tests.
Cross-dimensional	Point to the data values of specific member combinations. For example, you can point to the sales value for a specific product in a specific region. See Work with Member Combinations Across Dimensions.

For information about using operators with #MISSING, zero, and other values, see Operation Results on #MISSING Values and Zero (0) Values.

Dimension and Member Names

You can include dimension and member names in a formula. For example:

- `Scenario`
- `100-10`
- `Feb`

Constant Values

You can assign a constant value to a member. For example:

```
California = 120;
```

In this formula, California is a member in a sparse dimension and 120 is a constant value. Essbase automatically creates all possible data blocks for California and assigns the value 120 to all data cells. Many thousands of data blocks may be created.

To assign constants in a sparse dimension to only those intersections that require a value, use a FIX statement. See Constant Values Assigned to Members in a Sparse Dimension.

Nonconstant Values

If you assign anything other than a constant to a member in a sparse dimension, and no data block exists for that member, new blocks may not be created unless Essbase is enabled to create blocks on equations. By default, Create Blocks on Equations is disabled.

For example, to create blocks for West that did not exist before running the calculation, you must enable Create Blocks on Equations for this formula:

```
West = California + 120;
```

> **Note:**
>
> If Create Blocks on Equations is disabled for a database and data blocks exist for members on either the left- or right-side of the equation, the formula produces results.

You can enable Create Blocks on Equations at the database level, whereby blocks are always created, or you can control block creation within calculation scripts using the SET CREATEBLOCKONEQ ON | OFF calculation command.

Because unnecessary blocks can be created when Create Blocks on Equations is enabled at the application or database level, calculation performance can be affected. To control block creation within a calculation script, use the SET CREATEBLOCKONEQ ON | OFF calculation command. See Nonconstant Values Assigned to Members in a Sparse Dimension.

To enable the Create Blocks on Equations feature for all calculation scripts for a specific database, you can use the **alter database** MaxL statement.

Basic Equations

You can apply a mathematical operation to a formula to create a basic equation. The equation can be in the database outline or in a calculation script.

The syntax for an equation:

```
member = mathematical_operation;
```

member is a member name from the database outline and *mathematical_operation* is any valid mathematical operation.

In the following example, Essbase cycles through the database, subtracting the values in COGS from the values in Sales, and placing the results in Margin:

```
Margin = Sales - COGS;
```

The following example shows how to use an equation in the database outline and in a calculation script. In the outline, apply the following formula to a Markup member:

```
(Retail - Cost) % Retail;
```

Then, in a calculation script, use this formula:

```
Markup = (Retail - Cost) % Retail;
```

Essbase cycles through the database, subtracting the values in Cost from the values in Retail, calculating the resulting values as a percentage of the values in Retail, and placing the result in Markup.

Checking Formula Syntax

Essbase includes formula syntax checking that tells you about syntax errors in formulas. For example, Essbase tells you if you have mistyped a function name.

A syntax checker cannot tell you about semantic errors in a formula. Semantic errors occur when a formula does not work as you expect. To find semantic errors, run the calculation and check the results to ensure that they are as you expect.

Essbase displays the syntax checker results at the bottom of the Formula Editor. If Essbase finds no syntax errors, it displays the “No errors” message.

If Essbase finds one or more syntax errors, it displays the number of the line that includes the error and a brief description of the error. For example, if you do not include a semicolon end-of-line character at the end of a formula, Essbase displays a message similar to the following message:

```
Error: line 1: invalid statement; expected semicolon
```

If a formula passes validation in Formula Editor or Outline Editor, but Essbase Server detects semantic errors when the outline is saved, check the following:

- The incorrect formula is saved as part of the outline, even though it contains errors.
- Essbase Server writes a message in the application log that indicates what the error is and displays the incorrect formula.
- Essbase Server writes an error message to the comment field of the member associated with the incorrect formula. The message indicates that the incorrect formula was not loaded. You can view this comment in Outline Editor by closing and reopening the outline.
- If you do not correct the member formula, and a calculation that includes that member is run, the formula is ignored during the calculation.

After you have corrected the formula and saved the outline, the message in the member comment is deleted. You can view the updated comment when you reopen the outline.

Use Functions in Formulas

Essbase functions are predefined calc routines available in block storage cubes. They perform specialized calculations and return sets of members or data values.

You can use the following types of functions in outline formulas:

Table 18-2 List of Function Types

Function Type	Description
Boolean	Provide a conditional test by returning a TRUE (1) or FALSE (0) value. For example, you can use the @ISMBR function to determine whether the current member matches any members specified. See Conditional Tests.

Table 18-2 (Cont.) List of Function Types

Function Type	Description
Mathematical	Perform specialized mathematical calculations. For example, you can use the @AVG function to return the average value of a list of members. See Mathematical Operations.
Relationship	Look up data values within a database during a calculation. For example, you can use the @ANCESTVAL function to return the ancestor values of a specified member combination. See Member Relationship Functions.
Range	Declare a range of members as an argument to another function or command. For example, you can use the @SUMRANGE function to return the sum of all members within a specified range. See Range Functions.
Financial	Perform specialized financial calculations. For example, you can use the @INTEREST function to calculate simple interest or the @PTD function to calculate period-to-date values. See Financial Functions.
Specifying member lists and ranges	Specify multiple members or a range of members. For example, the @ISMBR function tests to see if a member that is currently being calculated matches any of a list or range of specified members. See Specify Member Lists and Ranges.
Generating member lists	Generate a list of members that is based on a specified member. For example, you can use the @ICHILDREN function to return a specified member and its children. See Generate Member Lists.
Character string manipulation	Manipulate character strings for member and dimension names. For example, you can generate member names by adding a character prefix to a name or removing a suffix from a name, or by passing the name as a string. See Manipulate Member Names.
Member combinations across dimensions	Point to data values of specific member combinations by using the cross-dimensional operator (->). See Work with Member Combinations Across Dimensions.
Interdependent values	For formulas that require values from members of the same dimension, but for which the required values have not yet been calculated. See Interdependent Values.

Table 18-2 (Cont.) List of Function Types

Function Type	Description
Variances and variance percentages	Calculate a variance or percentage variance between budget and actual values. See Variances Between Actual and Budget Values.
Allocation	Allocate values that are input at a parent level across child members. You can allocate values within the same dimension or across multiple dimensions. For example, you can use the @ALLOCATE function to allocate sales values that are input at a parent level to the children of the parent; the allocation of each child is determined by its share of the sales of the previous year. See Functions that Allocate Values.
Forecasting	Manipulate data for the purposes of smoothing or interpolating data, or calculating future values. For example, you can use the @TREND function to calculate future values that are based on curve-fitting to historical values. See Forecasting Functions.
Statistical	Calculate advanced statistics. For example, you can use the @RANK function to calculate the rank of a specified member or a specified value in a data set. See Statistical Functions.
Date and time	Use date and time characteristics in calculation formulas. For example, you can use the @TODATE function to convert date strings to numbers that can be used in calculation formulas. See Date and Time Function.
Calculation mode	Specify calculation modes that Essbase is to use to calculate a formula—cell, block, bottom-up, and top-down. See Calculation Mode Function.
Custom-defined	This type enables you to perform functions that you develop for calculation operations. These custom-developed functions are written in the Java programming language and are called by the calculator framework as external functions.

Note:

Abbreviations of functions are not supported. Some commands may work in an abbreviated form, but if another function has a similar name, Essbase may use the wrong function. Use the complete function name to ensure correct results.

Conditional Tests

You can define Essbase formulas that use a conditional test or a series of conditional tests to control the flow of calculation. In conjunction with the flow commands, you can use Boolean functions that return TRUE or FALSE (1 or 0, respectively) based on the result of the test.

The IF and ENDIF commands define a *conditional block*. The formulas between the IF and the ENDIF commands are executed only if the test returns TRUE (1). If the test returns FALSE (0), you can use the ELSE and ELSEIF commands to specify alternative actions. The formulas following each ELSE command are executed only if the previous test returns FALSE (0). Conditions following each ELSEIF command are tested only if the previous IF command returns FALSE (0).

When you use a conditional formula in a calculation script, enclose it in parentheses and associate it with a member in the database outline, as shown in the examples in this section.

In conjunction with an IF command, you can use functions that return TRUE or FALSE (1 or 0, respectively) based on the result of a conditional test. These functions are known as Boolean Functions.

Use Boolean functions to determine which formula to use. The decision is based on the characteristics of the current member combination. For example, to restrict a certain calculation to the members in the Product dimension that contain input data, preface the calculation with an IF test based on @ISLEV(Product,0).

If one of the function parameters is a cross-dimensional member, such as @ISMBR(Sales -> Budget), all of the parts of the cross-dimensional member must match the properties of the current cell to return a value of TRUE (1).

The following Boolean functions specify conditions:

Table 18-3 List of Boolean Functions That Test Conditions

Function	Condition
@ISACCTYPE	Current member has a specified accounts tag (for example, an Expense tag)
@ISANCEST	Current member is an ancestor of the specified member
@ISIANCEST	Current member is an ancestor of the specified member, or the specified member itself
@ISCHILD	Current member is a child of the specified member
@ISICHILD	Current member is a child of the specified member, or the specified member itself
@ISDESC	Current member is a descendant of the specified member
@ISIDESC	Current member is a descendant of the specified member, or the specified member itself
@ISGEN	Current member of the specified dimension is in the generation specified
@ISLEV	Current member of the specified dimension is in the level specified
@ISMBR	Current member matches any of the specified members
@ISPARENT	Current member is the parent of the specified member
@ISIPARENT	Current member is the parent of the specified member, or the specified member itself
@ISSAMEGEN	Current member (of the same dimension as the specified member) is in the same generation as the specified member
@ISSAMELEV	Current member (of the same dimension as the specified member) is in the same level as the specified member

Table 18-3 (Cont.) List of Boolean Functions That Test Conditions

Function	Condition
@ISSIBLING	Current member is a sibling of the specified member
@ISISIBLING	Current member is a sibling of the specified member, or the specified member itself
@ISUDA	A specified UDA exists for the current member of the specified dimension

When you place formulas on the database outline, you can use only the IF, ELSE, ELSEIF, and ENDIF commands and Boolean functions to control the flow of the calculations. You can use additional control commands in a calculation script.

For information about how to develop calculation scripts and how to use them to control how Essbase calculates a database, see Develop Calculation Scripts for Block Storage Cubes.

Examples of Conditional Tests

This example demonstrates how you can perform conditional tests in Essbase formulas.

You can apply the following formula to a Commission member in the cube outline to calculate commission at 1% of sales if the sales are greater than 500000:

```
IF(Sales > 500000)
   Commission = Sales * .01;
ENDIF;
```

If you place the formula in a calculation script, you must associate the formula with the Commission member as shown:

```
Commission (IF(Sales > 500000)
   Commission = Sales * .01;
ENDIF;)
```

Essbase cycles through the cube, performing these calculations:

1. The IF statement checks to see if the value of Sales for the current member combination is greater than 500000.
2. If Sales is greater than 500000, Essbase multiplies the value in Sales by 0.01 and places the result in Commission.

In the next example, the formula tests the ancestry of the current member and then applies the appropriate Payroll calculation formula:

```
IF(@ISIDESC(East) OR @ISIDESC(West))
   Payroll = Sales * .15;
ELSEIF(@ISIDESC(Central))
   Payroll = Sales * .11;
ELSE
   Payroll = Sales * .10;
ENDIF;
```

If you place the formula in a calculation script, you must associate the formula with the Payroll member as shown:

```
Payroll(IF(@ISIDESC(East) OR @ISIDESC(West))
   Payroll = Sales * .15;
ELSEIF(@ISIDESC(Central))
   Payroll = Sales * .11;
ELSE
   Payroll = Sales * .10;
ENDIF;)
```

Essbase cycles through the database, performing the following calculations:

1. The IF statement uses the @ISIDESC function to check whether the current member on the Market dimension is a descendant of either East or West.
2. If the current member on the Market dimension is a descendant of East or West, Essbase multiplies the value in Sales by 0.15 and moves on to the next member combination.
3. If the current member is not a descendant of East or West, the ELSEIF statement uses the @ISIDESC function to check whether the current member is a descendant of Central.
4. If the current member on the Market dimension is a descendant of Central, Essbase multiplies the value in Sales by 0.11 and moves to the next member combination.
5. If the current member is not a descendant of East, West, or Central, Essbase multiplies the value in Sales by 0.10 and moves to the next member combination.

See About Multidimensional Calculation Concepts.

Mathematical Operations

Essbase mathematical functions in the calculation language allow you to perform a variety of operations using numerical expression in formulas.

Table 18-4 List of Mathematical Functions

Function	Operation
@ABS	Return the absolute value of an expression
@AVG	Return the average value of the values in the specified member list
@EXP	Return the value of e (the base of natural logarithms) raised to power of the specified expression
@FACTORIAL	Return the factorial of an expression
@INT	Return the next-lowest integer value of a member or expression
@LN	Return the natural logarithm of a specified expression
@LOG	Return the logarithm to a specified base of a specified expression
@LOG10	Return the base-10 logarithm of a specified expression
@MAX	Return the maximum value among the expressions in the specified member list
@MAXS	Return the maximum value among the expressions in the specified member list, with the ability to skip zero and #MISSING values
@MIN	Return the minimum value among the expressions in the specified member list

Table 18-4 (Cont.) List of Mathematical Functions

Function	Operation
@MINS	Return the minimum value among the expressions in the specified member list, with the ability to skip zero and #MISSING values
@MOD	Return the modulus produced by the division of two specified members
@POWER	Return the value of the specified member raised to the specified power
@REMAINDER	Return the remainder value of an expression
@ROUND	Return the member or expression rounded to the specified number of decimal places
@SUM	Return the summation of values of all specified members
@TRUNCATE	Return the truncated value of an expression
@VAR	Return the variance (difference) between two specified members. See Variances Between Actual and Budget Values.
@VARPER	Return the percentage variance (difference) between two specified members. See Variances Between Actual and Budget Values.

Member Relationship Functions

Essbase member relationship functions allow you to use the member combination that Essbase is currently calculating to look up specific values.

Table 18-5 List of Member Relationship Functions

Function	Look-up Value
@ANCESTVAL	Ancestor values of the specified member combination
@ATTRIBUTEVAL	Numeric value of the attribute from the specified numeric or date attribute dimension associated with the current member
@ATTRIBUTESVAL	Text value of the attribute from the specified text attribute dimension associated with the current member
@ATTRIBUTEBVAL	Value (TRUE or FALSE) of the attribute from the specified Boolean attribute dimension associated with the current member
@CURGEN	Generation number of the current member combination for the specified dimension
@CURLEV	Level number of the current member combination for the specified dimension
@GEN	Generation number of the specified member
@LEV	Level number of the specified member
@MDANCESTVAL	Ancestor values of the specified member combination across multiple dimensions
@SANCESTVAL	Shared ancestor values of the specified member combination
@PARENTVAL	Parent values of the specified member combination
@MDPARENTVAL	Parent values of the specified member combination across multiple dimensions
@SPARENTVAL	Shared parent values of the specified member combination

Table 18-5 (Cont.) List of Member Relationship Functions

Function	Look-up Value
@XREF	Data value from another database to be used for calculation of a value from the current database
@XWRITE	Used to write values to another Essbase database, or to the same database

Range Functions

Range functions allow you to execute an Essbase function for a range of members, based on their position, value, or other factors.

Table 18-6 List of Range Functions

Function	Calculation
@AVGRANGE	The average value of a member across a range of members
@CURRMBRRANGE	A range of members that is based on the relative position of the member combination Essbase is currently calculating
@MAXRANGE	The maximum value of a member across a range of members
@MAXSRANGE	The maximum value of a member across a range of members, with the ability to skip zero and #MISSING values
@MDSHIFT	The next or *n*th member in a range of members, retaining all other members identical to the current member across multiple dimensions
@MINRANGE	The minimum value of a member across a range of members
@MINSRANGE	The minimum value of a member across a range of members, with the ability to skip zero and #MISSING values
@NEXT	The next or *n*th member in a range of members
@NEXTS	The next or *n*th member in a range of members, with the option to skip #MISSING, zero, or both values
@PRIOR	The previous or *n*th previous member in a range of members
@PRIORS	The previous or *n*th previous member in a range of members, with the option to skip #MISSING, zero, or both values
@SHIFT In some cases, @SHIFTPLUS or @SHIFTMINUS	The next or *n*th member in a range of members, retaining all other members identical to the current member and in the specified dimension
@SUMRANGE	The summation of values of all specified members across a range of members

Financial Functions

Financial functions allow you to include financial calculations in Essbase formulas, including accumulation, interest, growth, depreciation, and more.

Table 18-7 List of Financial Functions

Function	Calculation
@ACCUM	An accumulation of values up to the specified member
@COMPOUND	The proceeds of a compound interest calculation
@COMPOUNDGROWTH	A series of values that represent the compound growth of the specified member across a range of members
@DECLINE	Depreciation for a specific period, calculated using the declining balance method
@DISCOUNT	A value discounted by the specified rate, from the first period of the range to the period in which the amount to discount is found
@GROWTH	A series of values that represents the linear growth of the specified value
@INTEREST	The simple interest for a specified member at a specified rate
@IRR	The Internal Rate of Return on a cash flow that is calculated across the time dimension or a specified range of members and must contain at least one investment (negative) and one income (positive). Includes an initial guess of 0.07 (the initial guess cannot be configured).
@IRREX	The Internal Rate of Return on a cash flow that is calculated across the time dimension or a specified range of members and must contain at least one investment (negative) and one income (positive). Includes functionality to configure the initial guess and the number of iterations the algorithm can make.
@NPV	The Net Present Value of an investment (based on a series of payments and incomes)
@PTD	The period-to-date values of members in the dimension tagged as time
@SLN	The amount per period that an asset in the current period may be depreciated (calculated across a range of periods). The depreciation method used is straight-line depreciation.
@SYD	The amount per period that an asset in the current period may be depreciated (calculated across a range of periods). The depreciation method used is sum of the year's digits.

Note:

One member formula cannot contain multiple financial functions (for example, @NPV and @SLN, or multiple instances of @NPV). A member formula that requires multiple financial functions must be broken into separate formulas so that each formula contains only one financial function (for example, *MemberName*(@NPV(...));*Membername*(@NPV(...))).

Member-Related Functions

Use Essbase member functions in formulas to specify member lists and ranges, generate member lists, manipulate member names, and work with member combinations across dimensions.

This section discusses creating formulas that refer to members.

- Specify Member Lists and Ranges
- Generate Member Lists
- Manipulate Member Names
- Work with Member Combinations Across Dimensions

Specify Member Lists and Ranges

When using some Essbase calc functions, you may need to specify multiple members (a *mbrlist*), or you may need to specify a range of members.

For example, the @ISMBR function tests to see if a member that is currently being calculated matches any of a list or range of specified members.

The following table lists the syntax for specifying members:

Table 18-8 Syntax for Specifying Member Lists and Ranges

Member List or Range	Syntax
One member	The member name. For example: `Mar2001`
A list of members	A comma-delimited (,) list of member names. For example: `Mar2001, Apr2001, May2001`
A range of all members at the same level, between and including the two defining members	The two defining member names separated by a colon (:). For example: `Jan2000:Dec2000`
A range of all members in the same generation, between and including the two defining members	The two defining member names separated by two colons (::). For example: `Q1_2000::Q4_2000`
A function-generated list of members or a range of members	For a list of member list contents and corresponding functions, see Generate Member Lists.

Table 18-8 (Cont.) Syntax for Specifying Member Lists and Ranges

Member List or Range	Syntax

A combination of ranges and list	Separate each range, list, and function with a comma (,).
	For example:
	`Q1_97::Q4_98, FY99, FY2000`
	or
	`@SIBLINGS(Dept01), Dept65:Dept73, Total_Dept`

If you do not specify a list of members or a range of members in a function that requires either, Essbase uses the level 0 members of the dimension tagged as time. If no dimension is tagged as time, Essbase displays an error message.

Generate Member Lists

Essbase member set functions allow you to generate member lists that are based on a specified member or member list.

Table 18-9 List of Member Set Functions

Function	Contents of Member List
@ALLANCESTORS	All ancestors of the specified member, including ancestors of the specified member as a shared member. This function does not include the specified member.
@IALLANCESTORS	All ancestors of the specified member, including ancestors of the specified member as a shared member. This function includes the specified member.
@ANCEST	The ancestor of the specified member at the specified generation or level
@ANCESTORS	All ancestors of the specified member (optionally, up to the specified generation or level), but not the specified member
@IANCESTORS	All ancestors of the specified member (optionally, up to the specified generation or level), including the specified member
@LANCESTORS	All ancestors of the specified list of members (optionally, up to the specified generation or level), but not including the specified members
@ILANCESTORS	All ancestors of the specified list of members (optionally, up to the specified generation or level), including the specified members
@ATTRIBUTE	All base-dimension members that are associated with the specified attribute-dimension member

Table 18-9 (Cont.) List of Member Set Functions

Function	Contents of Member List
@WITHATTR	All base members that are associated with attributes that satisfy the specified conditions
@BETWEEN	All members whose name string value fall between, and are inclusive of, two specified string tokens
@CHILDREN	All children of the specified member, but not including the specified member
@ICHILDREN	All children of the specified member, including the specified member
@CURRMBR	The current member being calculated for the specified dimension
@DESCENDANTS	All descendants of the specified member (optionally, up to the specified generation or level), but not the specified member nor descendants of shared members
@IDESCENDANTS	All descendants of the specified member (optionally, up to the specified generation or level), including the specified member, but not descendants of shared members
@LDESCENDANTS	All descendants of the specified list of members (optionally, down to the specified generation or level), but not including the specified members
@ILDESCENDANTS	All descendants of the specified list of members (optionally, down to the specified generation or level), including the specified members
@RDESCENDANTS	All descendants of the specified member (optionally, up to the specified generation or level), including descendants of shared members, but not the specified member
@IRDESCENDANTS	All descendants of the specified member (optionally, up to the specified generation or level), including the specified member and descendants of shared members
@EQUAL	Member names that match the specified token name
@NOTEQUAL	Member names that do not match the specified token name
@EXPAND	Expands a member search by calling a member set function for each member in a member list
@GENMBRS	All members of the specified generation in the specified dimension
@LEVMBRS	All members of the specified level in the specified dimension
@LIKE	Member names that match the specified pattern.
@LIST	Separate lists of members to be processed by functions that require multiple list arguments
@MATCH	All members that match the specified wildcard selection
@MBRCOMPARE	Member names that match the comparison criteria

Table 18-9 (Cont.) List of Member Set Functions

Function	Contents of Member List
@MBRCOMPARE	Parent of the specified member
@MEMBER	The member with the name that is provided as a character string
@MERGE	A merged list of two member lists to be processed by another function
@PARENT	The parent of the current member being calculated in the specified dimension
@RANGE	A member list that crosses the specified member from one dimension with the specified member range from another dimension
@REMOVE	A list of members from which some members have been removed
@RELATIVE	All members of the specified generation or level that are above or below the specified member
@SHARE	A member list that identifies all shared members among the specified members
@SIBLINGS	All siblings of the specified member, but not the specified member
@ISIBLINGS	All siblings of the specified member, including the specified member
@LSIBLINGS	All siblings that precede the specified member in the database outline, but not the specified member
@RSIBLINGS	All siblings that follow the specified member in the database outline, but not the specified member
@ILSIBLINGS	All siblings that precede the specified member in the database outline, including the specified member
@IRSIBLINGS	All siblings that follow the specified member in the database outline, including the specified member
@SHIFTSIBLING	The sibling at the specified distance from the member
@NEXTSIBLING	The next, or right-most, sibling of the member
@PREVSIBLING	The previous, or left-most, sibling of the member
@UDA	All members that have a common UDA defined on Essbase Server
@XRANGE	A member list that identifies the range of members between (and inclusive of) two specified single or cross-dimensional members at the same level

Manipulate Member Names

You can work with Essbase member names as character strings in your block storage formulas and calc scripts.

The following table lists character string manipulation functions:

Table 18-10 List of Character String Manipulation Functions

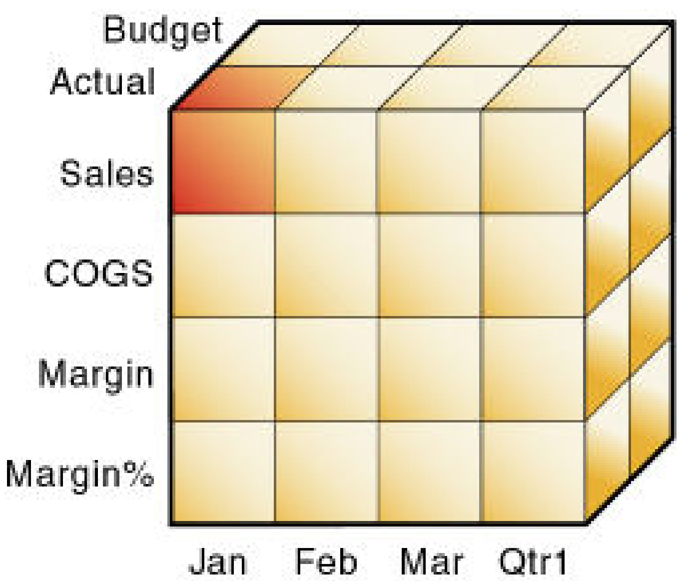
Function	Character String Manipulation
@CONCATENATE	Create a character string that is the result of appending a member name or specified character string to another member name or character string
@NAME	Return a member name as a string
@SUBSTRING	Return a substring of characters from another character string or from a member name

Work with Member Combinations Across Dimensions

Use the cross-dimensional operator to point to data values of specific member combinations. Create the cross-dimensional operator using a hyphen (-) and a greater-than symbol (>). Do not include a space between the cross-dimensional operator and members.

Below is a simplified illustration of a multidimensional cube, in which Jan is the first column on the X axis, Sales is the fourth and top-most row on the Y axis, and Actual is the first row on the Z axis. In this example, Sales -> Jan -> Actual is the intersection of a single data value.

Figure 18-2 Specifying a Single Data Value



The following example, which allocates miscellaneous expenses to each product in each market, illustrates how to use the cross-dimensional operator. The value of Misc_Expenses for all products in all markets is known. The formula allocates a percentage of the total Misc_Expenses value to each Product -> Market combination. The allocation is based on the value of Sales for each product in each market.

```
Misc_Expenses = Misc_Expenses -> Market -> Product *


   (Sales / ( Sales -> Market -> Product));
```

Essbase cycles through the database, performing these calculations:

1. Divides the Sales value for the current member combination by the total Sales value for all markets and all products (Sales -> Market -> Product).
2. Multiplies the value calculated in step 1 by the Misc_Expenses value for all markets and all products (Misc_Expenses -> Market -> Product).
3. Allocates the result to Misc_Expenses for the current member combination.

Using the cross-dimensional operator can have significant performance implications. For optimization guidelines, see Cross-Dimensional Operators in Member Formulas.

Value-Related Functions

You can use Essbase value functions in your outline formulas to calculate equations on interdependent metrics such as inventory over time periods, variances used in expense reporting, and allocations.

The following topics discuss use of formulas related to values.

- Interdependent Values
- Variances Between Actual and Budget Values
- Functions that Allocate Values

Interdependent Values

Sometimes, to calculate the required results, you need an Essbase formula to apply interdependent, multiple equations to one member in the outline.

Essbase optimizes calculation performance by calculating formulas for a range of members in the same dimension simultaneously. Some formulas, however, require values from members of the same dimension, and Essbase may not yet have calculated the required values.

A good example is that of cash flow, in which the opening inventory is dependent on the ending inventory from the previous month.

The values for Opening Inventory and Ending Inventory must be calculated on a month-by-month basis. Assume you want to achieve the results shown:

Table 18-11 Data Values for Cash Flow Example

	Jan	Feb	Mar
Opening Inventory	100	120	110
Sales	50	70	100
Addition	70	60	150
Ending Inventory	120	110	160

Assuming that the Opening Inventory value for January is loaded into the cube, the following calculations are required to get the results in the grid above:

```
1. January Ending   = January Opening - Sales + Additions
2. February Opening = January Ending
3. February Ending  = February Opening - Sales + Additions
```

```
4. March Opening     = February Ending
5. March Ending      = March Opening - Sales + Additions
```

You can calculate the required results by applying interdependent, multiple equations to one member in the outline.

The following formula, applied to the Opening Inventory member in the outline, calculates the correct values:

```
IF(NOT @ISMBR (Jan))
   "Opening Inventory" = @PRIOR("Ending Inventory");
ENDIF;
"Ending Inventory" = "Opening Inventory" - Sales + Additions;
```

If you place the formula in a calculation script, you must associate the formula with the Opening Inventory member as shown:

```
"Opening Inventory"
(IF(NOT @ISMBR (Jan))
   "Opening Inventory" = @PRIOR("Ending Inventory");
ENDIF;)
"Ending Inventory" = "Opening Inventory" - Sales + Additions;
```

Essbase cycles through the months, performing the following calculations:

1. The IF statement and @ISMBR function check that the current member on the Year dimension is not Jan. This step is necessary because the Opening Inventory value for Jan is an input value.
2. If the current month is not Jan, the @PRIOR function obtains the value for the Ending Inventory for the previous month. This value is then allocated to the Opening Inventory of the current month.
3. The Ending Inventory is calculated for the current month.

> **Note:**
>
> To calculate the correct results, you must place the above formula on one member, Opening Inventory. If you place the formulas for Opening Inventory and Ending Inventory on their separate members, Essbase calculates Opening Inventory for all months and then Ending Inventory for all months. This organization means that the value of the Ending Inventory of the previous month is not available when Opening Inventory is calculated.

Variances Between Actual and Budget Values

The @VAR and @VARPER calculation functions enable you to calculate a variance or percentage variance between budget and actual values.

You may want variances to be positive or negative, depending on whether you are calculating expense or nonexpense items in the Accounts dimension. By default, Essbase assumes that members are nonexpense items.

Consider the impact of expense vs nonexpense accounting on variance calculations:

- Expense items. You want Essbase to show a positive variance if the actual values are less than the budget values (for example, if actual costs are less than budgeted costs).
- Nonexpense items. You want Essbase to show a negative variance if the actual values are less than the budget values (for example, if actual sales are less than budgeted sales).

By default, without using functions, Essbase assumes that members are nonexpense items and calculates the variance accordingly.

When you use the @VAR or @VARPER functions, Essbase shows a positive variance if the actual values are less than the budget values. For example, in Sample.Basic, the children of Total Expenses are expense items. The Variance and Variance % members of the Scenario dimension calculate the variance between the Actual and Budget values.

Figure 18-3 Variance Example

```
Database: Basic (Current Alias Table: Default)
    Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Measures Accounts (Label Only)
        Profit (+) (Dynamic Calc)
            Margin (+) (Dynamic Calc)
            Total Expenses (-) (Dynamic Calc) (Expense Reporting)
                Marketing (+) (Expense Reporting)
                Payroll (+) (Expense Reporting)
                Misc (+) (Expense Reporting)
        Inventory (~) (Label Only)
        Ratios (~) (Label Only)
    Product
    Market
    Scenario (Label Only)
        Actual (+)
        Budget (~)
        Variance (~) (Dynamic Calc) (Two Pass Calc) @VAR(Actual, Budget);
        Variance % (~) (Dynamic Calc) (Two Pass Calc) @VARPER(Actual, Budget);
    Caffeinated Attribute
    Ounces Attribute
    Pkg Type Attribute
    Population Attribute
    Intro Date Attribute
```

See Setting Variance Reporting Properties.

Functions that Allocate Values

Essbase allocation functions allow you to allocate values that are input at the parent level across child members in the same dimension, or in different dimensions. For example, you can allocate costs across products, or you can perform cross dimensional and range allocations.

The allocation is based on a variety of criteria you specify when using the calculation functions.

Table 18-12 List of Allocation Functions

Function	Allocated Values
@ALLOCATE	Values from a member, cross-dimensional member, or value across a member list within the same dimension.
@MDALLOCATE	Values from a member, cross-dimensional member, or value across multiple dimensions.

For examples of calculation scripts using @ALLOCATE, see also Allocate Costs Across Products; using @MDALLOCATE, see Allocate Values Across Multiple Dimensions.

Forecasting Functions

Essbase forecasting functions allow you to manipulate data for the purposes of interpolating data or calculating future values.

Table 18-13 List of Forecasting Functions

Function	Data Manipulation
@MOVAVG	Apply a moving average to a data set and replace each term in the list with a trailing average. This function modifies the data set for smoothing purposes.
@MOVMAX	Apply a moving maximum to a data set and replace each term in the list with a trailing maximum. This function modifies the data set for smoothing purposes.
@MOVMED	Apply a moving median to a data set and replace each term in the list with a trailing median. This function modifies the data set for smoothing purposes.
@MOVMIN	Apply a moving minimum to a data set and replace each term in the list with a trailing minimum. This function modifies the data set for smoothing purposes.
@MOVSUM	Apply a moving sum to a data set and replace each term with a trailing sum. This function modifies the data set for smoothing purposes.
@MOVSUMX	Apply a moving sum to a data set and replace each term with a trailing sum. Specify how to assign values to members before you reach the number to sum. This function modifies the data set for smoothing purposes.
@SPLINE	Apply a smoothing spline to a set of data points. A spline is a mathematical curve that is used to smooth or interpolate data.
@TREND	Calculate future values and base the calculation on curve-fitting to historical values.

See Also

Forecast Future Values

Statistical Functions

Statistical functions allow you to calculate advanced statistics in Essbase.

Table 18-14 List of Statistical Functions

Function	Calculated Value
@CORRELATION	The correlation coefficient between two parallel data sets
@COUNT	The number of values in the specified data set
@MEDIAN	The median, or middle number, in the specified data set
@MODE	The mode, or the most frequently occurring value, in the specified data set
@RANK	The rank of the specified member or value in the specified data set
@STDEV	The standard deviation, based upon a sample, of the specified members
@STDEVP	The standard deviation, based upon the entire population, of the specified members
@STDEVRANGE	The standard deviation, crossed with a range of members, of the specified members
@VARIANCE	The variance, based upon a sample, of the specified data set
@VARIANCEP	The variance, based upon the entire population, of the specified data set

Date and Time Function

The Essbase date function allows you to use dates with other calculation functions.

The @TODATE function converts date strings to numbers that can be used in calculation formulas.

If your cube uses date measures, you can also use more functions to process them. Refer to Date & Time Functions.

Calculation Mode Function

The calculation mode function allows you to specify which calculation mode Essbase uses to calculate a formula.

@CALCMODE: Specify the calculation mode (cell, block, bottom-up, or top-down) that Essbase uses to calculate a formula

> **Note:**
>
> You can also use the configuration setting CALCMODE to set calculation modes to BLOCK or BOTTOMUP at the database, application, or server level.

Use Substitution Variables in Formulas

Substitution variables act as placeholders for information that changes regularly; for example, time-period information. You can use substitution variables in Essbase formulas that you apply to the cube outline.

When the outline is calculated, Essbase replaces the substitution variable with the value that you have assigned to it. You can create and assign values to substitution variables using the Essbase web interface or MaxL. To use the Essbase web interface, see Use Substitution Variables. For MaxL, use the statement alter database **add variable**.

You can set substitution variables at the server, application, and database (cube) levels. Essbase must be able to access the substitution variable from the application and database on which you are running the calculation scripts. See also Using Substitution Variables.

To use a substitution variable in a formula, enter an ampersand (`&`), followed by the substitution variable name.

Essbase treats any text string preceded by `&` as a substitution variable.

For example, assume that the substitution variable UpToCurr is defined as Jan:Jun. You can use the following @ISMBR function as part of a conditional test:

```
@ISMBR(&UpToCurr)
```

At the time Essbase calculates the outline, it replaces the substitution variable, as shown:

```
@ISMBR(Jan:Jun)
```

> **Note:**
>
> Substitution variables used in formulas for new outline members do not pass verification unless the outline is saved.

Use Formulas on Partitions

A partitioned Essbase application can span multiple service instances. You can use formulas in partitioning, just as you use formulas on your local database. If a formula you use in one cube references a value from another, Essbase must retrieve the data during calculation.

Ensure that the referenced values are up-to-date, and carefully consider the performance impact on the overall database calculation.

With transparent partitions, carefully consider how you use formulas on the data target.

See Also

Use Calculation Scripts on Partitions

Transparent Partitions and Member Formulas

Performance Considerations for Transparent Partitions

Display Formulas

To display a formula in the Essbase web interface, see Creating Member Formulas.

You can also use the query database MaxL statement.

19 Formula Examples for Block Storage Databases

These examples of Essbase formulas can help you develop your own. Examples include period-to-date calculations, rolling values, monthly asset movements, testing for missing/empty values, and attribute calculations.

The information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases.

Calculating Period-to-Date Values in an Accounts Dimension

If the outline includes a dimension tagged as accounts, you can use the @PTD function to calculate period-to-date values.

This example uses the Inventory branch of the Measures dimension from the Sample.Basic database, as shown:

```
Inventory (~) (Label Only)
   Opening Inventory (+) (TB First) (Expense Reporting)
      IF(NOT @ISMBR(Jan))
   Additions (~) (Expense Reporting)
   Ending Inventory (~) (TB Last) (Expense Reporting)
```

To calculate period-to-date values for the year and for the current quarter, add two members to the Year dimension: QTD for quarter-to-date and YTD for year-to-date. For example:

```
QTD (~) @PTD(Apr:May)
YTD (~) @PTD(Jan:May);
```

Assuming that the current month is May, add this formula to the QTD member:

```
@PTD(Apr:May);
```

And add this formula on the YTD member:

```
@PTD(Jan:May);
```

Essbase sums the values for the range of months, as appropriate. Opening Inventory, however, has a time balance tag, First, and Ending Inventory has a time balance tag, Last. Essbase takes these values and treats them accordingly. See Calculating First, Last, and Average Values.

The following is example of the calculation results for the members in the Inventory branch and for the Sales member:

Table 19-1 Results: Example Calculation Script for Calculating Period-to-Date Values

Measures->Time	Jan	Feb	Mar	Apr	May	QTD	YTD
Opening Inventory	100	110	120	110	140	**110**	**100**
Additions	110	120	100	160	180	**340**	**670**
Sales	100	110	110	130	190	**320**	**640**
Ending Inventory	110	120	110	140	130	**130**	**130**

The values for Sales and Additions have been summed.

Opening Inventory has a First tag. For QTD, Essbase takes the first value in the current quarter, which is Apr. For YTD, Essbase takes the first value in the year, which is Jan.

Ending Inventory has a Last tag. For QTD, Essbase takes the last value in the current quarter, which is May. For YTD, Essbase takes the last value in the year, which is also May.

Note:

You can also use Dynamic Time Series members to calculate period-to-date values.

Calculating Rolling Values

You can use the @AVGRANGE function to calculate rolling averages and the @ACCUM function to calculate rolling year-to-date values.

For example, assume that a database contains monthly Sales data values and that the database outline includes the members AVG_Sales and YTD_Sales.

You would add this formula to the AVG_Sales member:

```
@AVGRANGE(SKIPNONE, Sales, @CURRMBRRANGE(Year, LEV, 0, , 0));
```

And you would add this formula on the YTD_Sales member:

```
@ACCUM(Sales);
```

Essbase calculates the average Sales values across the months in the dimension tagged as time. The SKIPNONE parameter means that all values are included, even #MISSING values. Essbase places the results in AVG_Sales.

The following table shows the results when Essbase calculates the cumulative Sales values and places the results in YTD_Sales:

Table 19-2 Results: Example Calculation Script for Calculating Rolling Values

Measures -> Time	Jan	Feb	Mar	Qtr1
Sales	100	200	300	**600**
AVG_Sales	100	150	200	#MISSING
YTD_Sales	100	300	600	#MISSING

The values for AVG_Sales are averages of the months-to-date. For example, AVG_Sales -> Mar is an average of Sales for Jan, Feb, and Mar.

The values for YTD_Sales are the cumulative values up to the current month. So YTD_Sales -> Feb is the sum of Sales -> Jan and Sales -> Feb.

Calculating Monthly Asset Movements

You can use the @PRIOR function to calculate values based on a previous month's value.

For example, assume that a database contains assets data values that are stored on a month-by-month basis. You can calculate the difference between the assets values of successive months (the asset movement) by subtracting the previous month's value from the present month's value.

Assume these three members manage the asset values for the database:

- Assets for the monthly asset values
- Asset_MVNT for the asset movement values
- Opening_Balance for the asset value at the beginning of the year

For Jan, the Asset_MVNT value is calculated by subtracting the Opening_Balance value from the Jan value.

You would add this formula on the Asset_MVNT member:

```
IF(@ISMBR(Jan)) Asset_MVNT = Assets - Opening_Balance;
   ELSE Asset_MVNT = Assets - @PRIOR(Assets);
ENDIF;
```

The following table shows the results when Essbase calculates the difference between the values of assets in successive months:

Table 19-3 Results: Example Calculation Script for Calculating Monthly Asset Movements

Assets -> Time	Opening_Balance	Jan	Feb	Mar
Assets	1200	1400	1300	1800
Asset_MVNT		**200**	**-100**	**500**

Essbase cycles through the months, performing these calculations:

1. The IF statement and @ISMBR function check whether the current member on the Year dimension is Jan. This check is necessary because the Asset_MVNT value for Jan cannot be calculated by subtracting the previous month's value.
2. If the current member on the Year dimension is Jan, Essbase subtracts the Opening_Balance from the Jan -> Assets value and places the result in Jan -> Asset_MVNT.
3. If the current member on the Year dimension is not Jan, the @PRIOR function obtains the value for the previous month's assets. Essbase subtracts the previous month's assets from the current month's assets. It places the result in the current month's Asset_MVNT value.

Testing for #MISSING Values

You can test for #MISSING values in a database.

Assume that a database outline contains a member called Commission. Commission is paid at 10% of sales when the Sales value for the current member combination is not #MISSING. When applied to a Commission member in the database outline, the following formula calculates Commission:

```
IF(Sales <> #MISSING) Commission = Sales * .1;
   ELSE Commission = #MISSING;
ENDIF;
```

If you place the formula in a calculation script, you must associate it with the Commission member as shown:

```
Commission(IF(Sales <> #MISSING) Commission = Sales * .1;
   ELSE Commission = #MISSING;
ENDIF;);
```

Essbase cycles through the database, performing the following calculations:

1. The IF statement checks the value of the Sales member for the current member combination.
2. If Sales is not #MISSING, Essbase multiplies the value in the Sales member by 0.1 and places the result in the Commission member.
3. If Sales is #MISSING, Essbase places #MISSING in the Commission member.

Calculating an Attribute Formula

You can perform specific calculations on attribute-dimension members in a database.

For example, to calculate profitability by ounce for products sized in ounces, you can use the @ATTRIBUTEVAL function in a calculation formula. In the Sample.Basic database, the Ratios branch of the Measures dimension contains a member called Profit per Ounce. The formula on this member:

```
Profit/@ATTRIBUTEVAL(@NAME(Ounces));
```

Essbase cycles through the Products dimension, performing the following calculations:

1. For each base member that is associated with a member from the Ounces attribute dimension, the @ATTRIBUTEVAL function returns the numeric attribute value (for example, 12 for the member 12 under Ounces). The @NAME function is required to process the string “Ounces” before passing it to the @ATTRIBUTEVAL function.
2. Essbase then divides Profit by the result of @ATTRIBUTEVAL to yield Profit per Ounce.

See also Using Attributes in Calculation Formulas.

20 Defining Calculation Order

Essbase calculates the database block by block, in an order determined by the hierarchy defined in the outline.

Understanding the calculation order of Essbase block storage cubes can help you refine your application's calculation process. An Essbase database is calculated at the data block level. The Essbase calculation processor brings one or more blocks into memory, and calculates the required data values within each block. Data blocks are calculated in order, according to their block numbers. The database outline determines the order of blocks. Likewise, within each block, the values are calculated in order according to the hierarchy defined in the database outline.

- Data Storage in Data Blocks
- Member Calculation Order
- Block Calculation Order
- Data Block Renumbering
- Cell Calculation Order
- Calculation Passes
- Calculation of Shared Members

The information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases.

Data Storage in Data Blocks

Essbase stores data values in data blocks. Essbase creates a stored data block for each unique, non-empty combination of sparse dimension members. Non empty means that that at least one data value exists for the member combination. Each data block contains all the dense dimension member values for its unique combination of sparse dimension members. In the Sample.Basic database, the Year, Measures, and Scenario dimensions are dense; the Product and Market dimensions are sparse. The following image shows an outline of the dimensions in the Sample.Basic database:

Figure 20-1 Dimensions from the Sample.Basic Database

```
Database: Basic (Current Alias Table: Default)
    Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Measures Accounts (Label Only)
    Product
    Market
    Scenario (Label Only)
```

> **Note:**
>
> Sample.Basic also contains five attribute dimensions. These dimensions are sparse, Dynamic Calc, meaning that attribute data is not stored in the database.

Essbase creates a data block for each unique combination of members in the Product and Market dimensions (providing that at least one data value exists for the combination). For example, it creates one data block for the combination of 100-10, New York. This data block contains all the Year, Measures, and Scenario values for 100-10, New York. The following image shows an outline of the Product and Market dimensions in the Sample.Basic database:

Figure 20-2 Product and Market Dimensions from the Sample.Basic Database

```
Product
    100 (+) (Alias: Colas)
        100-10 (+) (Alias: Cola)
        100-20 (+) (Alias: Diet Cola)
        100-30 (+) (Alias: Caffeine Free Cola)
Market
    East (+) (UDAs: Major Market)
        New York (+) (UDAs: Major Market)
        Massachusetts (+) (UDAs: Major Market)
```

In Essbase, member combinations are denoted by the cross-dimensional operator. The symbol for the cross-dimensional operator is -> (a hyphen followed by a greater-than symbol). So 100-10, New York is written as 100-10 -> New York.

You can categorize data blocks in the following ways:

- Input

  These blocks are created by loading data to cells in a block. Input blocks can be created for (1) sparse, level 0 member combinations or (2) sparse, upper-level member combinations, when at least one of the sparse members is a parent-level member. Input blocks can be level 0 or upper-level blocks.

- Noninput

  These blocks are created through calculations. For example, in Sample.Basic, the East -> Cola block is created during a sparse calculation process (that is, the block did not exist before calculation).

- Level 0

  These blocks are created for sparse member combinations when all of the sparse members are level 0 members. For example, in Sample.Basic, New York -> Cola is a level 0 block because New York and Cola are level 0 members of their respective sparse dimensions. Level 0 blocks can be input or noninput blocks; for example, a level 0 noninput block is created during an allocation process, where data is loaded at a parent level and then allocated down to level 0.

- Upper level

  These blocks are created for sparse member combinations when at least one of the sparse members is a parent-level member. Upper-level blocks can be input or noninput blocks.

See Generations and Levels and Data Blocks and the Index System.

Member Calculation Order

The default order in which Essbase calculates outline members is, in general: Accounts, Time, dense, and sparse, taking into account the order of dimensions in the outline, and whether formulas are applied.

When you perform a default calculation (CALC ALL) on a database, Essbase calculates the dimensions in this order:

- If both a dimension tagged as accounts and a dimension tagged as time exist, and if formulas are applied to members on the accounts dimension, Essbase calculates in this order:
    1. Dimension tagged as accounts
    2. Dimension tagged as time
    3. Other dense dimensions (in the order in which they are displayed in the database outline)
    4. Other sparse dimensions (in the order in which they are displayed in the database outline)
- Otherwise, Essbase calculates in this order:
    1. Dense dimensions (in the order in which they are displayed in the database outline)
    2. Sparse dimensions (in the order in which they are displayed in the database outline)

> **Note:**
>
> Attribute dimensions, which are not included in the database consolidation, do not affect calculation order. See Working with Attributes.

In the Sample.Basic database, the dimensions are calculated in this order: Measures, Year, Scenario, Product, and Market.

You can override the default order by using a calculation script. See Develop Calculation Scripts for Block Storage Cubes.

Understanding the Effects of Member Relationships

The order of calculation within each dimension depends on the relationships between members in the database outline. Within each branch of a dimension, level 0 values are calculated first followed by their level 1, parent value. Then the level 0 values of the next branch are calculated, followed by their level 1, parent value. The calculation continues in this way until all levels are calculated.

The following image shows the Year dimension from the Sample.Basic database. The calculation order is shown on the left. This example assumes that the parent members are not tagged as Dynamic Calc.

Figure 20-3   Year Dimension from the Sample.Basic Database

```
17  Year Time
4       Qtr1 (+)
1           Jan (+)
2           Feb (+)
3           Mar (+)
8       Qtr2 (+)
5           Apr (+)
6           May (+)
7           Jun (+)
12      Qtr3 (+)
9           Jul (+)
10          Aug (+)
11          Sep (+)
16      Qtr4 (+)
13          Oct (+)
14          Nov (+)
15          Dec (+)
```

Jan is the first member in the first branch. Jan has no formula, so it is not calculated. The same applies to Feb and Mar, the other two members in the branch.

Essbase calculates Qtr1 by consolidating Jan, Feb, and Mar. In this example, these members are added.

Essbase then calculates the Qtr2 through Qtr4 branches in the same way.

Finally, Essbase calculates the Year member by consolidating the values of Qtr1 through Qtr4. These members are added.

Determining Member Consolidation

You can choose how Essbase consolidates members by applying any calculation operator (+, -, /, *, %, ~, ^) to the members in the database outline.

If an accounts member has a time balance tag (First, Last, or Average), Essbase consolidates it accordingly. See Calculating First, Last, and Average Values.

If a parent member has a label only operator, Essbase does not calculate the parent from its children.

If a member has a ~ operator, Essbase does not consolidate the member up to its parent.

If a member has a ^ operator, Essbase does not consolidate the member in any dimension.

> **Note:**
>
> If you use dynamic calculations, Essbase may use a different calculation order. See Calculation Order for Dynamic Calculation.

Ordering Dimensions in the Database Outline

To ensure the required calculation results, consider the calculation order of the dimensions in the database outline if you do either of these tasks:

- Use calculation operators to divide (/), multiply (*), or calculate percentages (%) for members in the database outline.
- Place formulas on members in the database outline.

You need not consider calculation order if you use only calculation operators to add (+) and subtract (–) members in the database outline and you do not use formulas in the outline.

See:

- Placing Formulas on Members in the Database Outline
- Using the Calculation Operators *, /, and %

Placing Formulas on Members in the Database Outline

If you place formulas on members in the database outline, consider the calculation order of the dimensions. A formula that is attached to a member on one dimension may be overwritten by a subsequent calculation on another dimension.

For example, the Sample.Basic database has a Measures dimension, tagged as accounts, and a Year dimension, tagged as time. Measures is calculated first and Year second. If you attach a formula to Margin on the Measures dimension, Essbase calculates the formula when it calculates the Measures dimension. Essbase then overwrites the formula when it consolidates the Year dimension. See Cell Calculation Order.

Using the Calculation Operators *, /, and %

If you use calculation operators to multiply (*), divide (/), and calculate percentages (%) for members in the database outline, consider the calculation order of the dimensions. The required calculated values may be overwritten by a subsequent calculation on another dimension.

For example, the Sample.Basic database has a Measures dimension, tagged as accounts, and a Year dimension, tagged as time. Measures is calculated first and Year second. If you multiply members on the Measures dimension, the calculated results may be overwritten when Essbase consolidates values on the Year dimension. See Cell Calculation Order.

When you use a multiplication (*), division (/), or percentage (%) operator to consolidate members, carefully order the members in the branch to achieve the required result.

Figure 20-4 shows calculations operators as they appear in an outline. Assume that the user wants to divide the total of Child 2 and Child 3 by Child 1. However, if Child 1 is the first member, Essbase starts with Child 1, starting with the value #MISSING, and dividing it by Child 1. The result is #MISSING. Essbase then adds Child 2 and Child 3. Obviously, this result is not the required one.

Figure 20-4   Calculation Operators in the Database Outline

```
Parent 1
    Child 1 (/)
    Child 2 (+)
    Child 3 (+)
```

To calculate the correct result, make Child 1 the last member in the branch.

You can apply a formula to a member on the database outline to achieve the same result. However, it is far more efficient to use these calculation operators on members as shown in Figure 20-4.

Avoiding Forward Calculation References

To obtain the calculation results you expect, ensure that the outline does not contain forward calculation references. *Forward calculation references* occur when the value of a calculating member is dependent on a member that Essbase has not yet calculated. In these cases, Essbase may not produce the required calculation results.

For example, consider the Product dimension shown below, which has three forward calculation references: two shared members (P100–20 and P300–20) and one nonshared member (P500–20):

Figure 20-5 Product Dimension with Forward Calculation References

```
Product
    Diet (~)
        P100-20 (+) (Shared Member)
        P200-20 (+) (Shared Member)
        P300-20 (+) (Shared Member)
        P400-20 (+) "P200-10"*2;
        P500-20 (+) ("P200-20"+"P300-20");
    Regular (+)
        P100 (+)
            P100-10 (+)
            P100-20 (+)
                P100-20-01 (+)
                P100-20-02 (+)
        P200 (+)
            P200-10 (+)
            P200-20 (+)
        P300 (+)
            P300-10 (+)
            P300-20 (+) "P100-20"+"P300-20";
```

In Outline Editor, when you verify the outline, Essbase identifies shared members with forward calculation references. Verifying the outline does not identify nonshared members that have forward calculation references. You can save and use an outline containing forward calculation references.

Consider the five members under Diet. The members P100-20, P300-20, and P500-20 have forward calculation references:

- P100-20 (+) (Shared Member): Essbase calculates the shared member P100-20 before it calculates the prototype member P100-20. Because the prototype member P100-20 has children, Essbase must calculate the prototype member by adding its children before it can accurately calculate the shared member P100-20.
- P300-20 (+) (Shared Member): Essbase calculates the shared member P300-20 before it calculates the prototype member P300-20. Because the prototype member P300-20 has a formula, Essbase must calculate the prototype member before it can accurately calculate the shared member P300-20.

- P500-20 (+) (“P200-20” + “P300-20”): The formula applied to P500-20 refers to members that Essbase has not yet calculated. One such prototype member, P300-20, has its own formula, and Essbase must calculate P300-20 before it can accurately calculate P500-20. The members P200-20 and P400-20 calculate correctly, because they do not have forward calculation references.
- P200-20 (+) (Shared Member): P200-20 is not a forward calculation reference, although Essbase calculates the shared member P200-20 before it calculates the prototype member P200-20. The prototype member P200-20 has no calculation dependencies (no children and no formula). Therefore, Essbase does not need to calculate the prototype member before the shared member. Essbase simply takes the value of the prototype member.
- P400-20 (+) (“P200-10” * 2): P400-20 is not a forward calculation reference, although the formula that is applied to P400-20 references a member that Essbase has not yet calculated. The member prototype in the formula does not itself have calculation dependencies. P200-10 is the only member in the formula, and P200-10 does not itself have children or a formula. Essbase accurately calculates P400-20.

To get accurate calculation results for P100-20, P300-20, and P500-20, change the order of members in the outline. By placing the Diet shared members after the Regular members, as shown below, you ensure that Essbase calculates the members in the required order.

Figure 20-6   Changed Product Dimension Without Forward Calculation References

```
Product
    Regular (+)
        P100 (+)
            P100-10 (+)
            P100-20 (+)
                P100-20-01 (+)
                P100-20-02 (+)
        P200 (+)
            P200-10 (+)
            P200-20 (+)
        P300 (+)
            P300-10 (+)
            P300-20 (+) "P100-20"+"P300-20"
    Diet (~)
        P100-20 (+) (Shared Member)
        P200-20 (+) (Shared Member)
        P300-20 (+) (Shared Member)
        P400-20 (+) "P200-10"*2;
        P500-20 (+) ("P200-20"+"P300-20");
```

Now Essbase calculates:

- The prototype member P100-20 before it calculates the shared member P100-20. So, P100-20 no longer has a forward calculation reference.
- The prototype member P300-20 before the shared member P300-20. So, P300-20 no longer has a forward calculation reference.
- The member with a formula, P300-20, before the member P500-20. So, P500-20 no longer has a forward calculation reference.

Block Calculation Order

Essbase calculates blocks in numerical order, starting with the first sparse dimension in the database outline.

Essbase defines sparse member combinations starting from the first sparse dimension in the outline.

In the Sample.Basic database, Product is the first sparse dimension in the database outline.

Figure 20-7 Dimensions in the Sample.Basic Database

```
Database: Basic (Current Alias Table: Default)
    Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Measures Accounts (Label Only)
    Product
    Market
    Scenario (Label Only)
```

> **Note:**
>
> The attribute dimensions in the Sample.Basic outline (not shown in the figure above), are not included in the database consolidation and do not affect block calculation order. See Working with Attributes..

As shown below, Product has 19 members (excluding the shared members, for which Essbase does not create data blocks). Therefore, the first 19 data blocks in the database are numbered according to the calculation order of members in the Product dimension.

Figure 20-8 Product Dimension from the Sample.Basic Database

```
Product
    100 (+) (Alias: Colas)
        100-10 (+) (Alias: Cola)
        100-20 (+) (Alias: Diet Cola)
        100-30 (+) (Alias: Caffeine Free Cola)
    200 (+) (Alias: Root Beer)
        200-10 (+) (Alias: Old Fashioned)
        200-20 (+) (Alias: Diet Root Beer)
        200-30 (+) (Alias: Sasparilla)
        200-40 (+) (Alias: Birch Beer)
    300 (+) (Alias: Cream Soda)
        300-10 (+) (Alias: Dark Cream)
        300-20 (+) (Alias: Vanilla Cream)
        300-30 (+) (Alias: Diet Cream)
    400 (+) (Alias: Fruit Soda)
        400-10 (+) (Alias: Grape)
        400-20 (+) (Alias: Orange)
        400-30 (+) (Alias: Strawberry)
    Diet (~) (Alias: Diet Drinks)
        100-20 (+) (Shared Member)
        200-20 (+) (Shared Member)
        300-30 (+) (Shared Member)
```

The other sparse dimension is Market. The first 19 data blocks contain the first member to be calculated in the Market dimension, which is New York. The table below shows the sparse member combinations of each Product member and New York, for the first five of these 19 data blocks:

Table 20-1 Sparse Member Combinations: Data Blocks 0 Through 4

Block Number	Product Member	Market Member
0	Cola (100-10)	New York
1	Diet Cola (100-20)	New York
2	Caffeine Free Cola (100-30)	New York
3	Colas (100)	New York
4	Old Fashioned (200-10)	New York

The next member in the Market dimension is Massachusetts. Essbase creates the next 19 data blocks for sparse combinations of each Product member and Massachusetts. The table below shows the sparse member combinations for the block numbers 19 through 23:

Table 20-2 Sparse Member Combinations: Data Blocks 19 Through 23

Block Number	Product Member	Market Member
19	Cola (100-10)	Massachusetts
20	Diet Cola (100-20)	Massachusetts
21	Caffeine Free Cola (100-30)	Massachusetts
22	Colas (100)	Massachusetts

Table 20-2 (Cont.) Sparse Member Combinations: Data Blocks 19 Through 23

Block Number	Product Member	Market Member
23	Old Fashioned (200-10)	Massachusetts

Essbase continues until blocks have been created for all combinations of sparse dimension members for which at least one data value exists.

Essbase creates a data block only if at least one value exists for the block. For example, if no data values exist for Old Fashioned Root Beer (200-10) in Massachusetts, then Essbase does not create a data block for 200-10 -> Massachusetts. However, Essbase does reserve the appropriate block number for 200-10 -> Massachusetts in case data is loaded for that member combination in the future.

When you run a default calculation (CALC ALL) on a database, each block is processed in order, according to its block number. If you have Intelligent Calculation turned on, and if the block does not need to be calculated, then Essbase skips the block and moves to the next block.

Data Block Renumbering

Essbase renumbers data blocks in the database when you add or make changes to sparse dimensions.

Essbase renumbers the data blocks when you make any of these changes:

- Move a sparse dimension
- Add a sparse dimension
- Change a dense dimension to a sparse dimension
- Move any member in a sparse dimension
- Delete any member in a sparse dimension
- Add a member to a sparse dimension

Cell Calculation Order

The order in which Essbase calculates the cells within each data block depends on how you have configured the database.

Each data block contains all the dense dimension member values for its unique combination of sparse dimension members. Each data value is contained in a cell of the data block.

How you have configured the database determines the calculation order of dense dimension members *within each block*, as well as the calculation order of blocks that represent sparse dimension members.

See the following examples:

- Cell Calculation Order: Example 1
- Cell Calculation Order: Example 2
- Cell Calculation Order: Example 3
- Cell Calculation Order: Example 4

- Cell Calculation Order for Formulas on a Dense Dimension

Cell Calculation Order: Example 1

In this example, which is the simplest case, these conditions are true:

- No dimensions have time or accounts tags.
- The setting for consolidating #MISSING values is turned on.
- Market and Year are dense dimensions.

  Essbase calculates dense dimensions in the order in which they are defined in the database outline. Assume that the Year dimension is positioned in the database outline before the Market dimension and is calculated first.

The following table shows a subset of the cells in a data block:

Table 20-3 Calculation Order Example 1: Input Cells and Calculated Cells

Year-Market	New York	Massachusetts	East
Jan	112345	68754	**3**
Feb	135788	75643	**4**
Mar	112234	93456	**5**
Qtr1	**1**	**2**	**6**

Data values have been loaded into the following input cells:

- Jan -> New York
- Feb -> New York
- Mar -> New York
- Jan -> Massachusetts
- Feb -> Massachusetts
- Mar -> Massachusetts

Essbase calculates the following cells. In the example below, the calculation order for these cells is represented by the numbers 1 through 6 that appear in the cells:

1. Qtr1 -> New York
2. Qtr1 -> Massachusetts
3. Jan -> East
4. Feb -> East
5. Mar -> East
6. Qtr1 -> East

Qtr1 -> East has multiple consolidation paths; it can be consolidated on Market or on Year. When consolidated on Market, it is a consolidation of Qtr1 -> New York and Qtr1 -> Massachusetts. When consolidated on Year, it is a consolidation of Jan -> East, Feb -> East, and Mar -> East.

Essbase knows that Qtr1 -> East has multiple consolidation paths. Therefore, it calculates Qtr1 -> East only once by consolidating the values for Qtr1 and uses the consolidation path of the dimension calculated last (in this example, the Market dimension), as shown below.

Table 20-4 Calculation Order Example 1: Results

Year-Market	New York	Massachusetts	East
Jan	112345	68754	**181099**
Feb	135788	75643	**211431**
Mar	112234	93456	**205690**
Qtr1	**360367**	**237853**	**598220**

Based on the calculation order, if you place a member formula on Qtr1 in the database outline, Essbase ignores it when calculating Qtr1 -> East. If you place a member formula on East in the database outline, the formula is calculated when Essbase consolidates Qtr1 -> East on the Market consolidation path.

If required, you can use a calculation script to calculate the dimensions in the order you choose.

Cell Calculation Order: Example 2

In this example, these conditions are true:

- No dimensions have time or accounts tags.
- The setting for consolidating #MISSING values is turned off (the default).
- Market and Year are dense dimensions.

  Essbase calculates dense dimensions in the order in which they are defined in the database outline. Assume that the Year dimension is positioned in the database outline before the Market dimension and is calculated first.

The following example shows a subset of the cells in a data block:

Table 20-5 Calculation Order Example 2: Input Cells and Calculated Cells

Year-Market	New York	Massachusetts	East
Jan	112345	68754	**4**
Feb	135788	75643	**5**
Mar	112234	93456	**6**
Qtr1	**1**	**2**	**3/7**

Data values have been loaded into the following input cells:

- Jan -> New York
- Feb -> New York
- Mar -> New York
- Jan -> Massachusetts
- Feb -> Massachusetts
- Mar -> Massachusetts

Essbase calculates the Qtr1 cells for New York, Massachusetts, and East and the East cells for Jan, Feb, and March. In the example below, the calculation order for these cells is represented by the numbers 1 through 7 that appear in the cells:

1. Qtr1 -> New York
2. Qtr1 -> Massachusetts
3. Qtr1 -> East
4. Jan -> East
5. Feb -> East
6. Mar -> East
7. Qtr1 -> East

Qtr1 -> East is calculated on both the Year and Market consolidation paths. First, Qtr1 -> East is calculated as a consolidation of Qtr1 -> New York and Qtr1 -> Massachusetts. Second, Qtr1 -> East is calculated as a consolidation of Jan -> East, Feb -> East, and Mar -> East.

The results are identical to the results for example 1. However, Qtr1 -> East has been calculated twice. This fact is significant when you need to load data at parent levels.

Table 20-6 Calculation Order Example 2: Results

Year-Market	New York	Massachusetts	East
Jan	112345	68754	**181099**
Feb	135788	75643	**211431**
Mar	112234	93456	**205690**
Qtr1	**360367**	**237853**	**598220**

Based on the calculation order, if you place a member formula on Qtr1 in the database outline, its result is overwritten when Essbase consolidates Qtr1 -> East on the Market consolidation path. If you place a member formula on East in the database outline, the result is retained, because the Market consolidation path is calculated last.

Cell Calculation Order: Example 3

In this example, these conditions are true:

- No dimensions have time or accounts tags.
- The setting for consolidating #MISSING values is turned off (the default).
- Data values have been loaded at parent levels.
- Market and Year are dense dimensions.

  Essbase calculates dense dimensions in the order in which they are defined in the database outline. Assume that the Year dimension is positioned in the database outline before the Market dimension and is calculated first.

The following example shows a subset of the cells in a data block:

Table 20-7 Calculation Order Example 3: Input Cells and #MISSING Values

Year-Market	New York	Massachusetts	East
Jan	#MISSING	#MISSING	**181099**
Feb	#MISSING	#MISSING	**211431**
Mar	#MISSING	#MISSING	**205690**
Qtr1	#MISSING	#MISSING	

The cells are calculated in the same order as in Cell Calculation Order: Example 2. Qtr1 -> East is calculated on both the Year and Market consolidation paths.

Because the setting for consolidating #MISSING values is turned off, Essbase does not consolidate the #MISSING values. Thus, the data that is loaded at parent levels is not overwritten by the #MISSING values below it.

However, if any of the child data values are not #MISSING, these values are consolidated and overwrite the parent values. For example, if Jan -> New York contains 50000.00, this value overwrites the values loaded at parent levels.

The results show that Essbase first correctly calculates the Qtr1 -> East cell by consolidating Jan -> East, Feb -> East, and Mar -> East, and then calculates on the Market consolidation path. However, it does not consolidate the #MISSING values in Qtr1 -> New York and Qtr1 -> Massachusetts; therefore, the value in Qtr1 -> East is not overwritten.

Table 20-8 Calculation Order Example 3: Results

Year-Market	New York	Massachusetts	East
Jan	#MISSING	#MISSING	**181099**
Feb	#MISSING	#MISSING	**211431**
Mar	#MISSING	#MISSING	**205690**
Qtr1	#MISSING	#MISSING	**598220**

Essbase must calculate the Qtr1 -> East cell twice to ensure that a value is calculated for the cell. If Qtr1 -> East is calculated according to only the last consolidation path, the result is #MISSING, which is not the required result.

Cell Calculation Order: Example 4

In this example, these conditions are true:

- The Year dimension is tagged as time.
- The Measures dimension is tagged as accounts.

  Essbase calculates a dimension tagged as accounts first, followed by a dimension tagged as time. Therefore, in this example, Measures is calculated before Year.
- The setting for consolidating #MISSING values is turned off (the default).
- The Marketing, Payroll, and Misc Expenses values have been loaded at the Qtr1, parent level.

The image below shows the Profit branch of the Measures dimension in the Sample.Basic database. This example assumes that Total Expenses is not a Dynamic Calc member.

Figure 20-9 Profit Branch of the Measures Dimension

```
Profit (+)
    Margin (+)
        Sales (+)
        COGS (-) (Expense Reporting)
    Total Expenses (-) (Expense Reporting)
        Marketing (+) (Expense Reporting)
        Payroll (+) (Expense Reporting)
        Misc (+) (Expense Reporting)
```

The following table shows a subset of the cells in a data block:

Table 20-9 Calculation Order Example 4: Input Cells, #MISSING Values, and Calculated Cells

Measures/Year	Jan	Feb	Mar	Qtr1
Sales	31538	32069	32213	**13**
COGS	14160	14307	14410	**14**
Margin	**1**	**4**	**7**	**10/15**
Marketing	#MISSING	#MISSING	#MISSING	15839
Payroll	#MISSING	#MISSING	#MISSING	12168
Misc	#MISSING	#MISSING	#MISSING	233
Total Expenses	**2**	**5**	**8**	**11/16**
Profit	**3**	**6**	**9**	**12/17**

The following cells have multiple consolidation paths:

- Margin -> Qtr1
- Total Expenses -> Qtr1
- Profit -> Qtr1

Because the setting for consolidating #MISSING values is turned off, Essbase does not consolidate the #MISSING values. Thus, any data that is loaded at parent levels is not overwritten by the #MISSING values and Essbase calculates the cells with multiple consolidation paths twice.

The results are shown below:

Table 20-10 Calculation Order Example 4: Results

Measures/Year	Jan	Feb	Mar	Qtr1
Sales	31538	32069	32213	**95820**
COGS	14160	14307	14410	**42877**
Margin	**17378**	**17762**	**17803**	**52943**
Marketing	#MISSING	#MISSING	#MISSING	15839
Payroll	#MISSING	#MISSING	#MISSING	12168
Misc	#MISSING	#MISSING	#MISSING	233
Total Expenses				**28240**
Profit	**17378**	**17762**	**17803**	**12/17**

Based on the calculation order, if you place a member formula on, for example, Margin in the database outline, its result is overwritten by the consolidation on Qtr1.

Cell Calculation Order for Formulas on a Dense Dimension

The cell calculation order within a data block is not affected by formulas on members. When Essbase encounters a formula in a data block, it locks any other required data blocks, calculates the formula, and proceeds with the data block calculation.

When placing a formula on a dense dimension member, carefully consider the cell calculation order. As described in the examples above, the dimension calculated last overwrites previous cell calculations for cells with multiple consolidation paths. If required, you can use a calculation script to change the order in which the dimensions are calculated. See Develop Calculation Scripts for Block Storage Cubes and Develop Formulas for Block Storage Cubes.

Calculation Passes

Whenever possible, Essbase calculates a database in one calculation pass through the database. In some situations, Essbase must perform multiple calculation passes.

Usually, Essbase calculates a database in one calculation pass through the database. Thus, it reads each of the required data blocks into memory only once, performing all relevant calculations on the data block and saving it. However, in some situations, Essbase may perform multiple calculation passes through a database. On subsequent calculation passes, Essbase brings data blocks back into memory, performs further calculations on them, and saves them again.

When you perform a default, full calculation of a database (CALC ALL), Essbase attempts to calculate the database in one calculation pass. If you have dimensions that are tagged as accounts or time, Essbase may have to do multiple calculation passes through the database.

The following table shows the number of calculation passes Essbase performs if you have dimensions that are tagged as time or accounts, and you have at least one formula on the accounts dimension:

Table 20-11   Calculation Passes For Accounts and Time Dimension

Dimension Tagged As Accounts	Dimension Tagged As Time	Calculation Passes	During each calculation pass, Essbase calculates based on:
Dense or Sparse	None	1	All dimensions
Dense	Dense	1	All dimensions
Dense	Sparse	2	Pass 1: Accounts and time dimensions Pass 2: Other dimensions
Sparse	Sparse	2	Pass 1: Accounts and time dimensions Pass 2: Other dimensions
Sparse	Dense	2	Pass 1: Accounts dimension Pass 2: Other dimensions

If you are using formulas that are tagged as Two-Pass, Essbase may need to do an *extra* calculation pass to calculate these formulas.

When you use a calculation script to calculate a database, the number of calculation passes Essbase needs to perform depends upon the calculation script.

If the isolation level is set for committed access, and multiple passes are required, Essbase writes data values at the end of each pass. Data retrievals that occur between passes can pick up intermediate values.

When you calculate a database, Essbase automatically displays the calculation order of the dimensions for each pass through the database and tells you how many times Essbase has cycled through the database during the calculation. Essbase displays this information in the ESSCMD shell window and in the application log.

For each data block, Essbase decides whether to do a dense or a sparse calculation. The type of calculation it chooses depends on the type of values within the data block. When you run a default calculation (CALC ALL) on a database, each block is processed in order, according to its block number.

Essbase calculates the blocks using this procedure:

- If you have Intelligent Calculation turned on, and if the block does not need to be calculated (if it is marked as *clean*), Essbase skips the block and moves to the next block.
- If the block needs recalculating, Essbase checks to see if the block is a level 0, an input, or an upper-level block.
- If the block is a level 0 block or an input block, Essbase performs a dense calculation on the block. Each cell in the block is calculated.
- If the block is an upper-level block, Essbase either consolidates the values or performs a sparse calculation on the data block.

  The sparse member combination of each upper-level block contains at least one parent member. Essbase consolidates or calculates the block based on the parent member's dimension. For example, if the upper-level block is for Product -> Florida from the Sample.Basic database, then Essbase chooses the Product dimension.

  If the sparse member combination for the block has multiple parent members, Essbase chooses the last dimension in the calculation order that includes a parent member. For example, if the block is for Product -> East, and you perform a default calculation on the Sample.Basic database, Essbase chooses the Market dimension, which contains East. The Market dimension is last in the default calculation order because it is placed after the Product dimension in the database outline. See Member Calculation Order.

  Based on the chosen sparse dimension, Essbase either consolidates the values or performs a sparse calculation on the data block:

  – If a formula is applied to the data block member on the chosen sparse dimension, Essbase performs a formula calculation on the sparse dimension. Essbase evaluates each cell in the data block. The formula affects only the member on the sparse dimension, so overall calculation performance is not significantly affected.

  – If the chosen sparse dimension is a default consolidation, Essbase consolidates the values, taking the values of the previously calculated child data blocks.

Calculation of Shared Members

A calculation on a shared member in Essbase is a calculation on its prototype member.

If you use the FIX command to calculate a subset of a database and the subset includes a shared member, Essbase calculates the prototype member.

Shared members are those that share data values with other members. For example, in the Sample.Basic database, Diet Cola, Diet Root Beer, and Diet Cream are consolidated under two parents: under Diet and under their product types—Colas, Root Beer, and Cream Soda. The members under the Diet parent are shared members, as shown below. See Shared Members.

Figure 20-10   Calculating Shared Members

- Product
  - 100 (+) (Alias: Colas)
    - 100-10 (+) (Alias: Cola)
    - 100-20 (+) (Alias: Diet Cola)
    - 100-30 (+) (Alias: Caffeine Free Cola)
  - 200 (+) (Alias: Root Beer)
    - 200-10 (+) (Alias: Old Fashioned)
    - 200-20 (+) (Alias: Diet Root Beer)
    - 200-30 (+) (Alias: Sasparilla)
    - 200-40 (+) (Alias: Birch Beer)
  - 300 (+) (Alias: Cream Soda)
    - 300-10 (+) (Alias: Dark Cream)
    - 300-20 (+) (Alias: Vanilla Cream)
    - 300-30 (+) (Alias: Diet Cream)
  - 400 (+) (Alias: Fruit Soda)
  - Diet (~) (Alias: Diet Drinks)
    - 100-20 (+) (Shared Member)
    - 200-20 (+) (Shared Member)
    - 300-30 (+) (Shared Member)

21 Intelligent Calculation for Block Storage Cubes

When Essbase calculates a block storage cube, it tracks which data blocks it calculates using markers. If you then load a subset of data, on subsequent calculations, Essbase calculates only the blocks that have not been calculated, and any blocks that require recalculation.

Intelligent Calculation is based on Essbase marking data blocks as clean or dirty. Intelligent Calculation applies only to non-federated, block storage cubes.

You can turn Intelligent Calculation on or off. See Turn Intelligent Calculation On or Off.

By default, Intelligent Calculation is turned on.

Benefits of Intelligent Calculation

Intelligent Calculation is designed to provide significant calculation performance benefits for these types of calculations:

- A full calculation of a database (CALC ALL), with some exceptions.

  See Limitations of Intelligent Calculation.
- A calculation script that calculates all members in one CALC DIM statement.
- For database calculations that cannot use Intelligent Calculation for the full calculation, you may be able to use Intelligent Calculation for part of the calculation.

  For example, to significantly improve calculation performance for a case in which you calculate a database by doing a default consolidation and then an allocation of data, enable Intelligent Calculation for the default consolidation and then disable Intelligent Calculation for the allocation.

  Assuming that Intelligent Calculation is turned on (the default), create a calculation script to perform these steps for a partial Intelligent Calculation:
  - Enable Intelligent Calculation, if it is disabled
  - Use CALC ALL to calculate the database
  - Use the SET UPDATECALC command to disable Intelligent Calculation
  - Allocate data
  - Optionally, enable Intelligent Calculation again

Ways to Use Intelligent Calculation

Intelligent Calculation is enabled by default for block storage cubes. When you run the first full calculation, Essbase calculates every block. When you recalculate the whole cube, Essbase checks each block to see whether it is marked as clean or dirty. Avoid unnecessary recalculations during partial cube calculations.

This section provides information on turning Intelligent Calculation on and off, and using it with different types of calculations.

- Turn Intelligent Calculation On or Off
- Intelligent Calculation for a Default, Full Calculation
- Intelligent Calculation for a Calculation Script, Partial Calculation

Turn Intelligent Calculation On or Off

By default, Intelligent Calculation is turned on for an Essbase block storage cube. You can also turn it on or off using the UPDATECALC configuration setting, or the SET UPDATECALC command in a calc script.

By default, Intelligent Calculation is turned on. To change the default, use the UPDATECALC configuration setting.

To turn Intelligent Calculation on and off for the duration of a calculation script, use the SET UPDATECALC command in a calculation script.

Intelligent Calculation for a Default, Full Calculation

Intelligent Calculation provides significant performance benefits when you do a full calculation (CALC ALL) of an Essbase block storage cube. If you do a full calculation, leave Intelligent Calculation turned on (the default) to take advantage of its performance benefits.

> **Caution:**
>
> When using Intelligent Calculation, note the information in Limitations of Intelligent Calculation.

Calculating for the First Time

When you do the first full calculation of the cube, Essbase calculates every block. The performance is the same whether Intelligent Calculation is on or off.

Recalculating

When you do a full recalculation of a cube with Intelligent Calculation turned on, Essbase checks each block to see whether it is marked as clean or dirty. See Intelligent Calculation and Data Block Status.

Checking data blocks has a 5% to 10% performance overhead, which is insignificant when compared to the performance gained by enabling Intelligent Calculation.

If, however, you recalculate a database in which more than approximately 80% of the values have changed, the overhead of Intelligent Calculation may outweigh the benefits. In this case, turn off Intelligent Calculation.

Intelligent Calculation for a Calculation Script, Partial Calculation

If your calc script calculates a block storage cube only partially, Essbase does not mark the calculated blocks as clean unless you use the SET CLEARUPDATESTATUS command.

Use SET CLEARUPDATESTATUS to avoid unnecessary recalculations.

Essbase marks a data block as clean when it calculates the data block on a full calculation (CALC ALL) or when it calculates all dimensions in one CALC DIM command. See Marking Blocks as Clean.

In any other calculations, Essbase does not mark calculated data blocks as clean, unless you use the SET CLEARUPDATESTATUS command in a calculation script. For example, if you calculate a subset of a database or calculate a database in two calculation passes, Essbase does not mark the calculated blocks as clean unless you call SET CLEARUPDATESTATUS.

The following calculation scripts do not cause Essbase to mark the calculated data blocks as clean:

```
FIX("New York")
   CALC DIM(Product, Measures);
ENDFIX
```

```
CALC DIM(Measures, Product);
CALC DIM(Market, Year, Scenario);
```

Intelligent Calculation and Data Block Status

To provide Intelligent Calculation, Essbase checks the status of the data blocks in a database. Data blocks have a calculation status of clean or dirty. Essbase marks a data block as clean after certain calculations.

When Intelligent Calculation is enabled, Essbase calculates only dirty blocks and their dependent parents. When disabled, Essbase calculates all data blocks, regardless of whether they are marked as clean or dirty.

Marking Blocks as Clean

Essbase marks data blocks as clean in these types of calculations:

- A full calculation (CALC ALL) of a database (the default calculation).
- A calculation script that calculates all the dimensions in one CALC DIM statement.

  For example, the following calculation script calculates all members in the Sample.Basic database:

  ```
  CALC DIM(Measures, Product, Market, Year, Scenario);
  ```

Compare this calculation script to a calculation script that calculates all the members with two CALC DIM statements:

```
CALC DIM(Measures, Product);
CALC DIM(Market, Year, Scenario);
```

Using two CALC DIM statements causes Essbase to do at least two calculation passes through the database. In this calculation, Essbase does not, by default, mark the data blocks as clean. Because Intelligent Calculation depends on accurate clean and dirty status, you must manage these markers carefully. See Maintaining Clean and Dirty Status.

Essbase marks calculated data blocks as clean only in the situations described above, unless you use the SET CLEARUPDATESTATUS command in a calculation script. See Using the SET CLEARUPDATESTATUS Command.

Marking Blocks as Dirty

Essbase marks a data block as dirty in these situations:

- Calculating the data block for a partial calculation of the database only if SET CLEARUPDATESTATUS AFTER is not part of the partial calculation statement in the calculation script
- Loading data into the data block
- Restructuring the database (for example, by adding a member to a dense dimension)
- Copying data to the data block; for example, using DATACOPY

Maintaining Clean and Dirty Status

To use Intelligent Calculation when calculating a subset of a database or when performing multiple calculation passes through a database, consider carefully the implications of how Essbase marks data blocks as clean. When using Intelligent Calculation, you must accurately maintain the clean and dirty status of the data blocks to ensure that Essbase recalculates the database as efficiently as possible.

For example, when you calculate a subset of a database, the newly calculated data blocks are not marked as clean by default. You can ensure that the newly calculated blocks are marked as clean by using the SET CLEARUPDATESTATUS AFTER command in a calculation script. Before creating the calculation script, see Using the SET CLEARUPDATESTATUS Command.

Limitations of Intelligent Calculation

Consider the following limitations and situations when using Intelligent Calculation:

- Intelligent Calculation works on a data block level and not on a cell level. For example, if you load a data value into one cell of a data block, the whole data block is marked as dirty.
- A CALC ALL that requires two passes through the database may calculate incorrectly. The problem occurs because blocks that are marked clean during the first pass are skipped during the second pass. To avoid this problem, turn Intelligent Calculation off or perform a CALC DIM for each dimension (rather than a CALC ALL for the database). A CALC ALL requires two passes through the database in either of these situations:
  - When the accounts dimension is sparse
  - When the accounts dimension is dense, the time dimension is sparse, and there is at least one more dense dimension in the outline

- Changing a formula on the database outline or changing an accounts property on the database outline does not cause Essbase to restructure the database. Therefore, Essbase does not mark the affected blocks as dirty. You must recalculate the appropriate data blocks. See Changing Formulas and Accounts Properties: Impact on Block Status.
- Whenever possible, Essbase calculates formulas that are tagged as two-pass and in the dimension tagged as accounts as part of the main calculation of a database. You may, however, need to use a calculation script to calculate some formulas twice. When you use a calculation script, disable Intelligent Calculation before recalculating formulas.
- When SET CREATENONMISSINGBLK is set to ON in a calculation script, Intelligent Calculation is turned off, and affected blocks are calculated whether they are marked clean or dirty.

Considerations for Essbase Intelligent Calculation on Oracle Exalytics In-Memory Machine

The following functionality applies only to Essbase running on an Oracle Exalytics In-Memory machine.

Status bits for Intelligent Calculation on a block storage database are not persisted. Therefore, Intelligent Calculation works as intended only if both of the following conditions are true:

- The whole index fits in the index cache
- Related calculation scripts that use Intelligent Calculation complete within the lifetime of the Essbase Server

For Intelligent Calculation, set the index cache size large enough to fit the whole index and to account for future index growth due to data load or calculation.

Using the SET CLEARUPDATESTATUS Command

In some cases, Essbase does not mark calculated blocks as clean; for example, if you calculate a subset of a database or calculate a database in two calculation passes. To manually mark data blocks as clean for purposes of Intelligent Calculation, use the SET CLEARUPDATESTATUS command in a calculation script. Read this section, and also see Intelligent Calculation and Data Block Status.

Understanding SET CLEARUPDATESTATUS

The SET CLEARUPDATESTATUS command has three parameters—AFTER, ONLY, and OFF.

- SET CLEARUPDATESTATUS AFTER;

  Essbase marks calculated data blocks as clean, even if it is calculating a subset of a database.
- SET CLEARUPDATESTATUS ONLY;

  Essbase marks the specified data blocks as clean but does not calculate the data blocks. This parameter provides the same result as AFTER, but without calculation.
- SET CLEARUPDATESTATUS OFF;

  Essbase calculates the data blocks but does not mark the calculated data blocks as clean. Data blocks are not marked as clean, even on a full calculation (CALC ALL) of a database. The existing clean or dirty status of the calculated data blocks remains unchanged.

Choosing a SET CLEARUPDATESTATUS Setting

When you use the SET CLEARUPDATESTATUS command to mark calculated data blocks as clean, be aware of these recommendations before selecting the parameter (AFTER, ONLY, OFF):

- Only calculated data blocks are marked as clean.
- Do not use the SET CLEARUPDATESTATUS AFTER command with concurrent calculations unless you are certain that the concurrent calculations do not need to calculate the same data block or blocks. If concurrent calculations attempt to calculate the same data blocks, with Intelligent Calculation enabled, Essbase does not recalculate the data blocks if the data blocks are already marked clean by the other concurrent calculation. See Handling Concurrent Calculations.
- When Essbase calculates data blocks on a first calculation pass through a database, it marks the data blocks as clean. If you try to calculate the same data blocks on a subsequent pass with Intelligent Calculation enabled, Essbase does not recalculate the data blocks, because they are already marked as clean.

Reviewing Examples That Use SET CLEARUPDATESTATUS

Assume a scenario using the Sample.Basic database:

- Sparse dimensions are Market and Product.
- New York is a member on the sparse Market dimension.
- Intelligent Calculation is turned on (the default).

These examples show different ways of using SET CLEARUPDATESTATUS:

Example 1: CLEARUPDATESTATUS AFTER

```
SET CLEARUPDATESTATUS AFTER;
FIX(“New York”)
   CALC DIM(Product);
ENDFIX
```

In this example, Essbase searches for dirty parent data blocks for New York (for example New York -> Colas, in which Colas is a parent member on the Product dimension). It calculates these dirty blocks and marks them as clean. Essbase does not mark the level 0 data blocks as clean, because they are not calculated. For information on level 0 blocks, see Defining Calculation Order.

Example 2: CLEARUPDATESTATUS ONLY

```
SET CLEARUPDATESTATUS ONLY;
FIX(“New York”)
   CALC DIM(Product);
ENDFIX
```

Essbase searches for dirty parent data blocks for New York (for example New York -> Colas, in which Colas is a parent member on the Product dimension). Essbase marks the dirty parent data blocks as clean but does not calculate the data blocks. Essbase does not mark the level 0 data blocks as clean because they are not calculated. For example, if New York -> 100-10 (a level 0 block) is dirty, it remains dirty.

Example 3: CLEARUPDATESTATUS OFF

```
SET CLEARUPDATESTATUS OFF;
CALC ALL;
CALC TWOPASS;
SET CLEARUPDATESTATUS ONLY;
CALC ALL;
```

In this example, Essbase first calculates all the dirty data blocks in the database. The calculated data blocks remain dirty. Essbase does not mark them as clean.

Essbase then calculates the members tagged as two-pass that are in the dimension tagged as accounts. Because the data blocks are still marked as dirty, Essbase recalculates them. Again, it does not mark the calculated data blocks as clean.

Essbase then searches for all the dirty blocks in the database and marks them as clean. It does not calculate the blocks, although a CALC ALL command is used.

Calculating Data Blocks

Essbase creates a data block for each unique combination of sparse dimension members, provided that at least one data value exists for the combination. Each data block represents all dense dimension member values for that unique combination of sparse dimension members.

For example, in the Sample.Basic database, the Market and Product dimensions are sparse. Therefore, the data block New York -> Colas represents all the member values on the Year, Measures, and Scenario dimensions for the sparse combination New York -> Colas.

These sections assume that you are familiar with the concepts of upper-level, level 0, and input data blocks. See Data Storage in Data Blocks.

Calculating Dense Dimensions

When you calculate a dense dimension and do not use a FIX command, Essbase calculates at least some of the data values in every data block in the database.

For example, the following calculation script is based on the Sample.Basic database:

```
SET CLEARUPDATESTATUS AFTER;
CALC DIM(Year);
```

This script calculates the Year dimension, which is a dense dimension. Because Year is dense, every data block in the database includes members of the Year dimension. Therefore, Essbase calculates data values in every data block. Because the script uses the SET CLEARUPDATESTATUS AFTER command, Essbase marks all data blocks as clean.

Calculating Sparse Dimensions

When you calculate a sparse dimension, Essbase may not need to calculate every data block in the database.

For example, the following calculation script is based on the Sample.Basic database:

```
SET CLEARUPDATESTATUS AFTER;
CALC DIM(Product);
```

This script calculates the Product dimension, which is a sparse dimension. Because Product is sparse, a data block exists for each member on the Product dimension. For example, one data block exists for New York -> Colas and another for New York -> 100-10.

Level 0 Effects

The data block New York -> 100-10 is a level 0 block; it does not represent a parent member on either sparse dimension (Market or Product). The data values for New York -> 100-10 are input values; they are loaded into the database. Therefore, Essbase does not need to calculate this data block. Nor does Essbase mark the data block for New York -> 100-10 as clean, even though the script uses the SET CLEARUPDATESTATUS AFTER command.

> **Note:**
>
> Essbase calculates level 0 data blocks if a corresponding sparse, level 0 member has a formula applied to it.

If you load data into a database, the level 0 data blocks into which you load data are marked as dirty. If you subsequently calculate only a sparse dimension or dimensions, the level 0 blocks remain dirty, because Essbase does not calculate them. Therefore, when you recalculate only a sparse dimension or dimensions, Essbase recalculates all upper-level data blocks, because the upper-level blocks are marked as dirty if their child blocks are dirty, although the upper-level blocks were originally clean.

Upper-Level Effects

Colas is a parent-level member on the Product dimension. Essbase must calculate values for Colas, so Essbase calculates this data block. Because the script uses the SET CLEARUPDATESTATUS AFTER command, Essbase marks the data block as clean.

When Essbase calculates a sparse dimension, it recalculates an upper-level data block if the block is dependent on one or more dirty child blocks.

Unnecessary Calculation

You can avoid unnecessary calculation by calculating at least one dense dimension. When you calculate a dense dimension and do not use the FIX command, data values are calculated in every data block, including the level 0 blocks. So the level 0 blocks are marked as clean.

Handling Concurrent Calculations

If concurrent calculations attempt to calculate the same data blocks, and Intelligent Calculation is turned on, Essbase may not recalculate the data blocks, because they are already marked as clean.

In the following example, based on the Sample.Basic database, Actual and Budget are members of the dense Scenario dimension. Because Scenario is dense, each data block in the database contains Actual and Budget values. If User 1 runs the following calculation script, Essbase calculates the Actual values for all data blocks that represent New York. Essbase marks the calculated data blocks as clean, although not all the data values in each calculated block have been calculated. For example, the Budget values have not been calculated.

```
SET CLEARUPDATESTATUS AFTER;
FIX("New York", Actual)
   CALC DIM(Product, Year);
ENDFIX
```

If User 2 runs the following calculation script to calculate the Budget values for New York, Essbase does not recalculate the specified data blocks, because they are already marked as clean. The calculation results for Budget are not correct.

```
SET CLEARUPDATESTATUS AFTER;
FIX("New York", Budget)
   CALC DIM(Product, Year);
ENDFIX
```

One way to solve this problem is to make the Scenario dimension sparse. Then the Actual and Budget values are in different data blocks; for example, New York -> Colas -> Actual and New York -> Colas -> Budget. In this case, the second calculation script correctly calculates Budget data block.

Running concurrent calculations might require an increase in the data cache.

Understanding Multiple-Pass Calculations

Whenever possible, Essbase calculates a database in one calculation pass through the database. See Calculation Passes.

When you use a calculation script to calculate a database, the number of calculation passes that Essbase performs depends upon the calculation script. See Intelligent Calculation and Data Block Status and Group Formulas and Dimensions in the Script.

For example, assume that Essbase calculates data blocks on a first calculation pass through a database and marks them as clean. If you attempt to calculate the same data blocks on a subsequent pass and Intelligent Calculation is enabled, Essbase does not recalculate the data blocks, because they are already marked as clean.

Reviewing Examples and Solutions for Multiple-Pass Calculations

These examples describe situations that produce incorrect calculation results and provide a solution to obtain correct results. They are based on the Sample.Basic database and assume that Intelligent Calculation is turned on.

Example 1: Intelligent Calculation and Two-Pass

This calculation script does a default calculation and then a two-pass calculation:

```
CALC ALL;
CALC TWOPASS;
```

Error

Essbase calculates the dirty data blocks in the database and marks all the data blocks as clean. Essbase then needs to recalculate the members tagged as two-pass in the dimension tagged as accounts. However, Essbase does not recalculate the specified data blocks because they are already marked as clean. The calculation results are not correct.

Solution

You can calculate the correct results by disabling Intelligent Calculation for the two-pass calculation.

Example 2: SET CLEARUPDATESTATUS and FIX

This calculation script calculates data values for New York. The calculation is based on the Product dimension:

```
SET CLEARUPDATESTATUS AFTER;
FIX("New York”)
   CALC DIM(Product);
ENDFIX
CALC TWOPASS;
```

Error

Essbase performs the following processes:

1. Essbase cycles through the database calculating the dirty data blocks that represent New York. The calculation is based on the Product dimension. Thus, Essbase calculates only the blocks that represent a parent member on the Product dimension (for example, New York -> Colas, New York -> Root Beer, and New York -> Fruit Soda), and then only calculates the aggregations and formulas for the Product dimension.
2. Because the SET CLEARUPDATESTATUS AFTER command is used, Essbase marks the calculated data blocks as clean, although not all data values in each calculated block have been calculated.
3. Essbase should recalculate the members tagged as two-pass in the dimension tagged as accounts; however, some of these data blocks are already marked as clean from the calculation in the previous step. Essbase does not recalculate the data blocks that are marked as clean. The calculation results are not correct.

Solution

You can calculate the correct results by disabling Intelligent Calculation for the two-pass calculation.

Example 3: SET CLEARUPDATESTATUS and Two CALC DIM Commands

This calculation script bases the database calculation on the Product and Year dimensions. Because two CALC DIM commands are used, Essbase does two calculation passes through the database:

```
SET CLEARUPDATESTATUS AFTER;
CALC DIM(Product);
CALC DIM(Year);
```

Error

Essbase performs the following processes:

1. Essbase cycles through the database calculating the dirty data blocks. The calculation is based on the Product dimension, as in Example 2: SET CLEARUPDATESTATUS and FIX.
2. Because the SET CLEARUPDATESTATUS AFTER command is used, Essbase marks the calculated data blocks as clean, although not all data values in each calculated block have been calculated.
3. Essbase should recalculate the data blocks. The recalculation is based on the Year dimension. However, as a result of the calculation in the previous step, some data blocks are already marked as clean, and Essbase does not recalculate them. The calculation results are not correct.

Solution

You can calculate the correct results by using one CALC DIM command to calculate the Product and Year dimensions. Essbase calculates both dimensions in one calculation pass through the database.

The following calculation script calculates the correct results:

```
SET CLEARUPDATESTATUS AFTER;
CALC DIM(Product, Year);
```

> **Note:**
>
> When you calculate several dimensions in one CALC DIM command, Essbase calculates the dimensions in the default calculation order and not in the order in which you list them in the command. See Member Calculation Order.

Example 4: Two Calculation Scripts

This example calculates data values for New York but calculates based on two dimensions using two calculation scripts. The first calculation script calculates the Product dimension:

```
SET CLEARUPDATESTATUS AFTER;
FIX("New York")
   CALC DIM(Product);
ENDFIX
```

Essbase calculates the data blocks that include New York. Because the calculation is based on the Product dimension, Essbase calculates only the dirty blocks that include a parent member on the Product dimension (for example, New York -> Colas, New York -> Root Beer, and New York -> Fruit Soda), and calculates only the aggregations and formulas for the Product dimension.

Because of the CLEARUPDATESTATUS AFTER command, Essbase marks the calculated data blocks as clean, although not all data values in each calculated block have been calculated.

The second calculation script calculates the Year dimension:

```
SET CLEARUPDATESTATUS AFTER;
FIX("New York")
   CALC DIM(Year);
ENDFIX
```

Essbase calculates the data blocks that represent New York. Because the calculation is based on the Year dimension, which is a dense dimension, Essbase should calculate all data blocks that include New York, although within each block Essbase calculates only the aggregations and formulas for the Year dimension.

Error

As a result of the first calculation, some data blocks for New York are already marked as clean. Essbase does not recalculate these data blocks with the second calculation script because the data blocks are marked as clean. The calculation results are not correct.

Solution

You can calculate the correct results by telling Essbase not to mark the calculated data blocks as clean. The following calculation script calculates the correct results:

```
SET CLEARUPDATESTATUS OFF;
FIX("New York")
   CALC DIM(Product);
ENDFIX
SET CLEARUPDATESTATUS AFTER;
FIX("New York")
   CALC DIM(Year);
ENDFIX
```

With the SET CLEARUPDATESTATUS OFF command, Essbase calculates dirty data blocks but does not to mark them as clean, unlike the SET CLEARUPDATESTATUS AFTER command.

This solution assumes that the data blocks are not marked as clean from a previous partial calculation of the database.

You can ensure that all data blocks are calculated, regardless of their status, by disabling Intelligent Calculation. The following calculation script calculates all specified data blocks, regardless of their clean or dirty status:

```
SET UPDATECALC OFF;
FIX("New York")
   CALC DIM(Year, Product);
ENDFIX
```

Because you have not used the SET CLEARUPDATESTATUS AFTER command, Essbase does not mark calculated data blocks as clean.

Effects of Intelligent Calculation

When you use Intelligent Calculation in Essbase, the clean / dirty status of data blocks may not always be as you expect if you update formulas, change an Accounts tag in the outline, use relationship functions in a sparse dimension, copy or clear data, convert currency, or restructure.

Using Intelligent Calculation may have implications for how you administer the cube. Review the topics that follow to learn the implications of each action.

- Changing Formulas and Accounts Properties: Impact on Block Status
- Using Relationship and Financial Functions: Impact on Block Status
- Restructuring Databases: Impact on Block Status
- Copying and Clearing Data: Impact on Block Status
- Copying and Clearing Data: Impact on Block Status

Changing Formulas and Accounts Properties: Impact on Block Status

When you use Intelligent Calculation in Essbase, data block status does not change if you edit the outline to make changes to a member formula or to change an Accounts tag.

Because neither changing a formula nor changing an accounts property requires Essbase to restructure the outline, data blocks affected by such a change are not marked as dirty. For example, if you change a time balance tag in the accounts dimension, Essbase does not restructure the cube, and does not mark the affected blocks as dirty.

When you subsequently run a default calculation with Intelligent Calculation turned on, the changes are not calculated. To recalculate the appropriate data blocks, use a calculation script to perform any of the following tasks:

- Disable Intelligent Calculation and calculate the member formula that has changed.
- Disable Intelligent Calculation and use the FIX…ENDFIX command to calculate the appropriate subset of the cube.
- Disable Intelligent Calculation and perform a default (CALC ALL) calculation on the cube.

Using Relationship and Financial Functions: Impact on Block Status

Formulas on sparse dimension members that contain relationship or financial functions always cause recalculation, even if Essbase Intelligent Calculation marks the blocks as clean.

If you use relationship functions (for example, @PRIOR or @NEXT) or financial functions (for example, @ACCUM, @NPV, or @INTEREST) in a formula on a sparse dimension or a dense dimension, Essbase always recalculates the data block that contains the formula.

For more information, see Calculation Script Formulas and Intelligent Calculation Status.

Restructuring Databases: Impact on Block Status

When you use Intelligent Calculation in Essbase, all data blocks are marked as dirty after the cube is restructured.

When you restructure a database (for example, by adding a member to a dense dimension), all data blocks potentially need recalculating. Therefore, Essbase marks all data blocks as dirty. When you calculate the restructured database, all blocks are calculated.

Note:

Changing a formula in the database outline or changing an accounts property in the database outline does not cause Essbase to restructure the database. You must recalculate the appropriate data blocks. See Changing Formulas and Accounts Properties: Impact on Block Status.

Copying and Clearing Data: Impact on Block Status

When you use Intelligent Calculation in Essbase, the clean / dirty status of data blocks may not always be as you expect if you copy values to a block or clear data from blocks.

When you copy values to a data block by using the DATACOPY command, the resulting data block is marked as dirty. Essbase calculates the block when you recalculate a database.

When you clear data values by using the CLEARDATA and CLEARBLOCK commands, Essbase clears all the blocks regardless of how they are marked.

Converting Currencies: Impact on Block Status

When you use Intelligent Calculation in Essbase, the clean / dirty status of data blocks may not always be as you expect if you have a currency conversion cube.

If you convert currencies using the CCONV command, the resulting data blocks are marked as dirty. Essbase calculates all currency-converted data blocks when you recalculate.

22

Dynamically Calculating Data Values

To improve calculation performance, you can apply Dynamic Calc for select values in an Essbase block storage cube. Dynamic Calc adds efficiency by calculating some members at retrieval time.

When you design the way your cube is calculated, you may find it to be more efficient to calculate some member combinations when data is retrieved, instead of precalculating the member combinations during batch database calculations.

The information in this chapter applies only to block storage databases, and is not relevant to aggregate storage databases.

- Understanding Dynamic Calculation
- Benefitting from Dynamic Calculation
- Using Dynamic Calculation
- Choosing Values to Calculate Dynamically
- How Dynamic Calculation Changes Calculation Order
- Reducing the Impact on Retrieval Time
- Using Dynamic Calculations with Standard Procedures
- Creating Dynamic Calc Members
- Restructuring Databases
- Dynamically Calculating Data in Partitions

Understanding Dynamic Calculation

When you design the overall database calculation, it may be more efficient to calculate some member combinations when you retrieve their data, instead of precalculating the member combinations during a batch database calculation. Dynamically calculating some values in a database can significantly improve the performance of an overall database calculation.

In Essbase, you can define a member to have a *dynamic calculation*. This definition tells Essbase to calculate a data value for the member as users request it. Dynamic calculation shortens batch database calculation time, but may increase retrieval time for the dynamically calculated data values. See Reducing the Impact on Retrieval Time.

In Essbase you specify dynamic calculations on a per-member basis. See Understanding Dynamic Calc Members

Understanding Dynamic Calc Members

For a member tagged as Dynamic Calc, Essbase does not calculate its data value during a batch database calculation (for example, during a CALC ALL). Instead, Essbase calculates the data value upon retrieval (for example, when you retrieve the data into Smart View.)

Specifically, Essbase calculates a data value dynamically when you request the data value in either of two ways:

- By retrieving the data value into Smart View
- By running a report script that displays the data value

Essbase does not store the calculated value; it recalculates the value for each subsequent retrieval.

Retrieving the Parent Value of Dynamically Calculated Child Values

If you retrieve a parent value that is calculated from Dynamic Calc child members, Essbase must dynamically calculate the child member combinations before calculating the parent value.

Benefitting from Dynamic Calculation

Dynamically calculating some database values can significantly improve the performance of an overall database calculation.

By calculating some data values dynamically, you reduce:

- Batch calculation time of the database, because Essbase has fewer member combinations to calculate.
- Disk usage, because Essbase stores fewer calculated data values. Database size and index size are also reduced.
- Database restructure time. For example, adding or deleting a Dynamic Calc member in a dense dimension does not change the data block size, so Essbase does not need to restructure the database. See Restructuring Databases.
- Time required to back up the database. Because database size is reduced, Essbase takes less time to perform a backup.

Data values that Essbase calculates dynamically can take longer to retrieve. You can estimate the retrieval time for dynamically calculated members. See Reducing the Impact on Retrieval Time.

Using Dynamic Calculation

You can tag any member as Dynamic Calc, except the following members:

- Level 0 members that do not have a formula
- Label-only members
- Shared members

Which members you choose to calculate dynamically depends on the database structure and on the balance between (1) the need for reduced calculation time and disk usage and (2) the need for speedy data retrieval for users. See Choosing Values to Calculate Dynamically.

In Outline Editor, you can see which members are Dynamic Calc. The following figure shows Dynamic Calc members.

Figure 22-1   Sample.Basic Outline Showing Dynamic Calc Members

```
Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Qtr1 (+) (Dynamic Calc)
    Qtr2 (+) (Dynamic Calc)
    Qtr3 (+) (Dynamic Calc)
    Qtr4 (+) (Dynamic Calc)
```

In Smart View, users can display visual cues to distinguish dynamically calculated values.

When developing spreadsheets that include dynamically calculated values, spreadsheet designers may want to use the spreadsheet Navigate Without Data option, so that Essbase does not dynamically calculate and store values while test spreadsheets are built.

Choosing Values to Calculate Dynamically

Dynamically calculating some data values decreases calculation time and disk usage and reduces database restructure time but increases retrieval time for dynamically calculated data values.

Use the guidelines described in the following sections when deciding which members to calculate dynamically.

Dense Members and Dynamic Calculation

Consider making the following changes to members of dense dimensions:

- Tag upper-level members of dense dimensions as Dynamic Calc.
- Try tagging level 0 members of dense dimensions with simple formulas as Dynamic Calc, and assess the increase in retrieval time.

  Simple formulas do not require Essbase to perform an expensive calculation. Formulas containing financial functions or cross-dimensional operators (->) are complex formulas.

Sparse Members and Dynamic Calculation

Consider making the following changes to members of sparse dimensions:

- Tag some upper-level members of sparse dimensions that have six or fewer children as Dynamic Calc.
- Tag sparse-dimension members with complex formulas as Dynamic Calc.

  A complex formula requires Essbase to perform an expensive calculation. For example, any formula that contains a financial function is a complex formula. See Complex Essbase Member Formulas
- Tag upper-level members in a dimension that you frequently restructure as Dynamic Calc.
- Do not tag upper-level, sparse-dimension members that have 20 or more descendants as Dynamic Calc.

Two-Pass Members and Dynamic Calculation

To reduce the time needed to perform batch calculations, tag two-pass members as Dynamic Calc. You can tag any Dynamic Calc member as two-pass, even if it is not on an accounts dimension. See Two-Pass Calculation.

For information about the interaction of members tagged as two-pass and attribute members, see Comparing Attribute and Standard Dimensions.

For information about how querying on a two-pass member in a dense dimension impacts the dynamic calculator cache, see Two-Pass Members and Dynamic Calculation.

Parent-Child Relationships and Dynamic Calculation

If a parent member has one child member, and you tag the child as Dynamic Calc, you must also tag the parent as Dynamic Calc.

However, if a parent member has one child member, and the parent is a Dynamic Calc member, you do not have to tag the child as Dynamic Calc.

Calculation Scripts and Dynamic Calculation

When Essbase calculates a CALC ALL or CALC DIM statement in a calculation script, it bypasses the calculation of Dynamic Calc members.

Similarly, if a member set function (for example, @CHILDREN or @SIBLINGS) is used to specify the list of members to calculate, Essbase bypasses the calculation of any Dynamic Calc members in the resulting list.

If you specify a Dynamic Calc member explicitly in a calculation script, the calculation script fails. You cannot do a calculation script calculation of a Dynamic Calc member. To use a calculation script to calculate a member explicitly, do not tag the member as Dynamic Calc.

For example, the following calculation script is valid only if Qtr1 is not a Dynamic Calc member:

```
FIX (East, Colas)
   Qtr1;
ENDFIX
```

Formulas and Dynamically Calculated Members

You can include a dynamically calculated member in a formula when you apply the formula to the database outline. For example, if Qtr1 is a Dynamic Calc member, you can place the following formula on Qtr1 in the database outline:

```
Qtr1 = Jan + Feb;
```

You cannot make a dynamically calculated member the target of a formula calculation in a calculation script; Essbase does not reserve memory for a dynamically calculated value and, therefore, cannot assign a value to it. For example, if Qtr1 is a Dynamic Calc member, Essbase displays a syntax error if you include the following formula in a calculation script:

```
Qtr1 = Jan + Feb;
```

If Qtr1 is a Dynamic Calc member and Year is not, you can use the following formula in a calculation script:

```
Year = Qtr1 + Qtr2;
```

This formula is valid because Essbase does not assign a value to the dynamically calculated member.

> **Note:**
>
> When you reference a dynamically calculated member in a formula in the database outline or in a calculation script, Essbase interrupts the regular calculation to do the dynamic calculation. This interruption can significantly reduce calculation performance.

Scripts and Dynamically Calculated Members

The preprocessing phase of a calculation script cannot determine whether an outline contains dense Dynamic Calc members. If a script contains runtime-dependent formulas, Essbase must calculate all dense Dynamic Calc members when the script is executed. Using the SET FRMLRTDYNAMIC OFF calculation command improves performance by stopping calculation of these Dynamic Calc members.

Dynamically Calculated Children

If the calculation of a member depends on the calculation of Dynamic Calc child members, Essbase must calculate the child members first during the batch database calculation in order to calculate the parent. Therefore, regular calculation time is not reduced. This requirement applies to members of sparse dimensions and members of dense dimensions.

For example, in the figure below, Qtr1 is a Dynamic Calc member. Its children, Jan, Feb, and Mar, are not dynamic members. Its parent, Year, is not a dynamic member. When Essbase calculates Year during a batch database calculation, it must consolidate the values of its children, including Qtr1. Therefore, it must take the additional time to calculate Qtr1, although Qtr1 is a Dynamic Calc member.

Figure 22-2 Sample.Basic Outline, Showing Qtr1 as a Dynamic Calc Member

```
Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D)
    Qtr1 (+) (Dynamic Calc)
        Jan (+)
        Feb (+)
        Mar (+)
```

How Dynamic Calculation Changes Calculation Order

Dynamic Calc optimizes batch calculation performance, and changes the order in which Essbase calculates values. Sparse dimensions are calculated first. Take precautions when using Dynamic Calc in asymmetric data sets or with Two Pass members.

Using dynamically calculated data values changes the order in which Essbase calculates the values, and can have implications for how you administer the cube.

- Calculation Order for Dynamic Calculation
- Calculation Order for Dynamically Calculating Two-Pass Members
- Calculation Order for Asymmetric Data
- Solve Order in Hybrid Mode

Calculation Order for Dynamic Calculation

When you use Dynamic Calc, Essbase calculates values of sparse dimensions first, and then dense. Dense values are calculated in the order of accounts, time, time series, other dense dimensions, and finally, attributes.

When Essbase dynamically calculates data values, it calculates the data in an order different from the batch database calculation order.

During batch calculations, Essbase calculates the database in the following order:

1. Dimension tagged as accounts
2. Dimension tagged as time
3. Other dense dimensions (in the order in which they appear in the database outline)
4. Other sparse dimensions (in the order in which they appear in the database outline)
5. Two-pass calculations

See Defining Calculation Order.

For dynamically calculated values, on retrieval, Essbase calculates the values by calculating the database in the following order:

1. Sparse dimensions
   - If the dimension tagged as time is sparse and the database outline uses time series data, Essbase bases the sparse calculation on the time dimension.
   - Otherwise, Essbase bases the calculation on the dimension that it normally uses for a batch calculation.
2. Dense dimensions
   a. Dimension tagged as accounts, if dense
   b. Dimension tagged as time, if dense
   c. Time series calculations
   d. Remaining dense dimensions
   e. Two-pass calculations
   f. Attributes

If your data retrieval uses attribute members, the last step in the calculation order is the summation of the attributes. Attribute calculation performs on-the-fly aggregation on data blocks that match the attribute members specified in the query. When the query contains two-pass calculation members, attribute calculation applies the two-pass calculation member formula after all the aggregated values are collected. This two-pass calculation uses the data values from the attribute calculation, not the values in a real data block.

The use of attribute members in your query causes Essbase to disregard the value of the Time Balance member in the dynamic calculations. During retrievals that do not use attributes, the value of the Time Balance member is applied to the calculations. The difference in calculation procedure between the use and nonuse of attribute members generates different results for any upper-level time members that are dynamically calculated.

During retrievals that do not use attributes, these dynamically calculated members are calculated in the last step and, therefore, apply the time balance functionality properly. However, during retrievals that do use attributes, the summation of the attribute is the last step applied. The difference in calculation order produces two different, predictable results for upper-level time members that are dynamically calculated.

Calculation Order for Dynamically Calculating Two-Pass Members

Dynamic Calc changes the order in which Essbase calculates values. Take precautions when using Dynamic Calc with Two Pass members.

Consider the following information to ensure that Essbase produces the required calculation result when it dynamically calculates data values for members tagged as two-pass (see Two-Pass Calculation).

If multiple Dynamic Calc dense dimension member are tagged as two-pass, Essbase performs the dynamic calculation in the first pass, and then calculates the two-pass members in this order:

1. Two-pass members in the accounts dimension, if any exist
2. Two-pass members in the time dimension, if any exist
3. Two-pass members in the remaining dense dimensions in the order in which the dimensions appear in the outline

For example, in the Sample.Basic database, assume the following:

- Margin% in the dense Measures dimension (the dimension tagged as accounts) is tagged as Dynamic Calc and two-pass.
- Variance in the dense Scenario dimension is tagged as Dynamic Calc and two-pass.

Essbase calculates the accounts dimension member first. So, Essbase calculates Margin% (from the Measures dimension) and then calculates Variance (from the Scenario dimension).

If Scenario is a sparse dimension, Essbase calculates Variance first, following the regular calculation order for dynamic calculations. Essbase then calculates Margin%. See Calculation Order for Dynamic Calculation.

This calculation order does not produce the required result, because Essbase needs to calculate Margin % -> Variance using the formula on Margin %, and not the formula on Variance. You can avoid this problem by making Scenario a dense dimension. This problem does not occur if the Measures dimension (the accounts dimension) is sparse, because Essbase still calculates Margin% first.

Calculation Order for Asymmetric Data

Essbase Dynamic Calc may not be suitable for use with asymmetric data sets. Review the examples in this topic against your use case.

Because the calculation order of Dynamic Calc differs from that of batch calculations, in some cube outlines, you may get different calculation results if you tag certain members as Dynamic Calc. These differences happen when Essbase dynamically calculates asymmetric data.

Symmetric data calculations produce the same results no matter which dimension is calculated.

Using the data set in the symmetric example below, the calculation for Qtr1-> Profit produces the same result whether you calculate along the dimension tagged as time or the dimension tagged as accounts. Calculating along the time dimension, add the values for Jan, Feb, and Mar:

```
50+100+150=300
```

Calculating along the accounts dimension, subtract Qtr1 -> COGS from Qtr1 -> Sales:

```
600–300=300
```

Table 22-1 Example of a Symmetric Calculation

Time -> Accounts	Jan	Feb	Mar	Qtr1
Sales	100	200	300	600
COGS	50	100	150	300
Profit (Sales – COGS)	50	100	150	300

Asymmetric data calculations calculate differently along different dimensions.

Using the data set in the asymmetric example below, the calculation for East -> Sales produces the correct result when you calculate along the Market dimension, but produces an incorrect result when you calculate along the accounts dimension. Calculating along the Market dimension, adding the values for New York, Florida, and Connecticut produces the correct results:

```
50 + 100 + 100 = 250
```

Calculating along the accounts dimension, multiplying the value East -> Price by the value East -> UnitsSold produces incorrect results:

```
15 * 50 = 750
```

Table 22-2 Example of an Asymmetric Calculation

Market -> Accounts	New York	Florida	Connecticut	East
UnitsSold	10	20	20	50
Price	5	5	5	15
Sales (Price * UnitsSold)	50	100	100	250

In the following outline, East is a sparse dimension, and Accounts is a dense dimension:

```
East
     New York (+)
     Florida (+)
     Connecticut (+)
Accounts
     UnitsSold (~)
     Price (~)
     Sales (~) UnitsSold*Price;
```

If East and Sales are tagged as Dynamic Calc, Essbase calculates a different result than it does if East and Sales are not tagged as Dynamic Calc.

If East and Sales are not Dynamic Calc members, Essbase produces the correct result by calculating these dimensions:

1. Dense Accounts dimension—calculating the values for UnitsSold, Price, and Sales for New York, Florida, and Connecticut
2. Sparse East dimension—aggregating the calculated values for UnitsSold, Price, and Sales for New York, Florida, and Connecticut to obtain the Sales values for East

If East and Sales are Dynamic Calc members, Essbase produces an incorrect result by calculating these dimensions:

1. Sparse East dimension—aggregating the values for UnitsSold, Price, and Sales for New York, Florida, and Connecticut to obtain the values for East
2. Values for East -> Sales—taking the aggregated values in the East data blocks and performing a formula calculation with these values to obtain the value for Sales

To avoid this problem and ensure that you obtain the required results, do not tag the Sales member as Dynamic Calc.

Solve Order in Hybrid Mode

Solve order in Essbase determines the order in which dynamic calculation executes in hybrid mode. You can customize the solve order or accept the default, which is optimized for high performance and dependency analysis.

The concept of solve order applies to dynamic calculation execution, whether initiated by a dynamic member formula or a dynamic dependency in a calculation script. When a cell is evaluated in a multidimensional query, the order in which the calculations should be resolved may be ambiguous, unless solve order is specified to indicate the required calculation priority.

You can set solve order for dimensions or members, or you can use the default Essbase solve order. The minimum solve order you can set is 0, and the maximum is 127. A higher solve order means the member is calculated later; for example, a member with a solve order of 1 is solved before a member with a solve order of 2.

When hybrid mode is enabled, the default solve order (also known as calculation order) closely matches that of block storage databases:

Dimension/Member Type	Default Solve Order Value
Stored members	0
Sparse Dimension Members	10
Dense Account dimension members	30
Dense Time dimension members	40
Dense regular dimension members	50
Attribute dimension members	90
Two pass dynamic members	100
MDX calculated members or named sets (defined in MDX With)	120

In summary, the default solve order in hybrid mode dictates that stored members are calculated before dynamic calc members, and sparse dimensions are calculated before dense dimensions, in the order in which they appear in the outline (top to bottom).

Dynamic members (with or without formulas) that do not have a specified solve order inherit the solve order of their dimension, unless they are tagged as two pass.

Two-pass calculation is a setting you can apply, in block storage mode, to members with formulas that must be calculated twice to produce the correct value.

> **Note:**
>
> Do not use two-pass calculation with hybrid mode cubes. Only use solve order.

Two pass is not applicable in hybrid mode, and any members tagged as two pass are calculated last, after attributes. In hybrid mode, you should implement a custom solve order, instead of two pass, if the default solve order does not meet your requirements.

The default solve order in hybrid mode is optimized for these scenarios:

- Forward references, in which a dynamic member formula references a member that comes later in the outline order. There is no outline order dependency in hybrid mode.
- Aggregation of child values based on outline order more closely matches aggregation using equivalent formulas.
- Dynamic dense members as dependencies inside sparse formulas. In hybrid mode, if a sparse formula references a dense dynamic member, the reference is ignored, because sparse dimensions are calculated first. To change this, assign a solve order to the sparse dimension that is higher than (calculated later than) the dense dimension’s solve order.

Customizing the Solve Order

If you need to adjust the behavior of dynamic calculations in hybrid mode, customizing the solve order of dimensions and members helps you achieve it without making major changes to the outline.

If you implement a custom solve order, it overrides the default solve order. If members or dimensions have equal solve order, the order in which they appear in the outline (top to bottom) resolves the conflict.

Unless you customize a solve order for certain members, the top dimension member's solve order applies for all dynamic members in the dimension.

To change the solve order, use the outline editor in the Essbase web interface, or use Smart View (see Changing the Solve Order of a Selected POV).

The minimum solve order you can set is 0, and the maximum is 127. A higher solve order means the member is calculated later.

To explore use cases for solve order, see the Solve Order templates in the Technical section of the gallery of application workbooks, which you can find in the files catalog in Essbase.

Notes on Solve Order in Non-hybrid Mode

In aggregate storage cubes,

- Solve order is set to 0 for all dimensions.

- Aggregation executes in outline order, except:
  - Stored hierarchy members are processed first.
  - Dynamic hierarchy members are processed next.

In non-hybrid block storage cubes, the default solve order is

- sparse before dense
- accounts before time
- attributes last

> **Note:**
>
> If Accounts members' solve order are set manually to be greater than Time members' solve order, the Accounts will be evaluated after Dynamic Time Series members.

Reducing the Impact on Retrieval Time

The increase in retrieval time when you dynamically calculate a member of a dense dimension is not significant unless the member contains a complex formula. The increase in retrieval time may be significant when you tag members of sparse dimensions as Dynamic Calc.

The following sections discuss ways you can analyze and manage the effect of Dynamic Calc members on a database.

> **Note:**
>
> For a list of functions that have the most significant effect on query retrieval, see Member Set Functions and Performance.

Displaying a Retrieval Factor

To help you estimate any increase in retrieval time, Essbase calculates a retrieval factor for a database outline when you save the outline. Essbase calculates this retrieval factor based on the dynamically calculated data block that is the most expensive for Essbase to calculate. The retrieval factor takes into account only aggregations. It does not consider the retrieval impact of formulas.

The retrieval factor is the number of data blocks that Essbase must retrieve from the disk or from the database to calculate the most expensive block. If the database has Dynamic Calc members in dense dimensions only (no Dynamic Calc members in sparse dimensions), the retrieval factor is 1.

An outline with a high retrieval factor (for example, greater than 2000) can cause long delays when users retrieve data. However, the actual impact on retrieval time also depends on how many dynamically calculated data values a user retrieves. The retrieval factor is only an indicator. In some applications, using Dynamic Calc members may reduce retrieval time because the database size and index size are reduced.

Essbase displays the retrieval factor value in the application log.

A message similar to this sample indicates a retrieval factor:

```
[Wed Sep 20 20:04:13 2000] Local/Sample///Info (1012710)
Essbase needs to retrieve [1] Essbase kernel blocks in order
to calculate the top dynamically-calculated block.
```

This message tells you that Essbase needs to retrieve one block to calculate the most expensive dynamically calculated data block.

Displaying a Summary of Dynamically Calculated Members

When you add Dynamic Calc members to a database outline and save the outline, Essbase provides a summary of how many members are tagged as Dynamic Calc. Essbase displays the summary in the application log.

Increasing Retrieval Buffer Size

By default, the retrieval buffer size is 10 KB. However, you may speed retrieval time if you set the retrieval buffer size greater than 10 KB.

Using Dynamic Calculator Caches

By default, when Essbase calculates a Dynamic Calc member in a dense dimension (for example, for a query), it writes all blocks needed for the calculation into an area in memory called the dynamic calculator cache. When Essbase writes these blocks into the dynamic calculator cache, it expands them to include all Dynamic Calc members in the dense dimensions.

If a query includes a two-pass calculation member in a dense dimension, the query needs one dynamic calculator cache for each block retrieved.

Using the Essbase dynamic calculator cache enables centralized control of memory usage for dynamic calculations. Managing data blocks in the dynamic calculator cache also reduces the overall memory space requirement and can improve performance by reducing the number of calls to the operating system to do memory allocations.

> **Note:**
>
> The dynamic calculator cache and the calculator cache use different approaches to optimizing calculation performance.

Reviewing Dynamic Calculator Cache Usage

Essbase writes two messages to the application log for each data retrieval. In the following example, the first message describes the total time required for the retrieval:

```
[Thu Aug 03 14:33:00 2005]Local/Sample/Basic/aspen/Info(1001065)
Regular Extractor Elapsed Time : [0.531] seconds

[Thu Aug 03 14:33:00 2005]Local/Sample/Basic/aspen/Info(1001401)
```

```
Regular Extractor Big Blocks Allocs -- Dyn.Calc.Cache : [30] non-
Dyn.Calc.Cache : [0]
```

If a dynamic calculator cache is used, a second message displays the number of blocks calculated within the data calculator cache (Dyn.Calc.Cache: [*n*]) and the number of blocks calculated in memory outside dynamic calculator cache (non-Dyn.Calc.Cache: [*n*]).

To determine whether the dynamic calculator cache is being used effectively, review both messages and consider your configuration settings. For example, if the message indicates that blocks were calculated outside and in a dynamic calculator cache, you may increase the DYNCALCCACHEMAXSIZE setting.

You can use the **query database** MaxL statement with the **performance statistics** grammar to view a summary of dynamic calculator cache activity.

Using Dynamic Calculations with Standard Procedures

Using dynamic calculations with standard Essbase procedures affects these processes:

- Clearing data and data blocks

  The CLEARDATA command has no effect on data values for Dynamic Calc members.

- Copying data

  You cannot copy data to a dynamically calculated data value. You cannot specify a Dynamic Calc member as the target for the DATACOPY calculation command.

- Loading data

  When you load data, Essbase does not load data into member combinations that contain a Dynamic Calc member. Essbase skips these members during data load and does not display an error message.

- Exporting data

  Essbase does not calculate dynamically calculated values before exporting data. Essbase does not export values for Dynamic Calc members.

- Including dynamic members in calculation scripts

  When calculating a database, Essbase skips the calculation of any Dynamic Calc members. Essbase displays an error message if you attempt to do a member calculation of a Dynamic Calc member in a calculation script. See Calculation Scripts and Dynamic Calculation.

Creating Dynamic Calc Members

You can create Dynamic Calc members in these ways:

- In an application workbook, enter the appropriate property value in the **Storage** field.

  See Understanding Dimension Worksheets.

- In the outline, select the **Data Storage Type**.

  See Setting Information Properties.

- In a dimension build data file, enter the appropriate property value.

  See Use the Source Data to Work with Member Properties.

Restructuring Databases

When you add a Dynamic Calc member to a dense dimension, Essbase does not reserve space in the data block for the member's values. Therefore, Essbase does not need to restructure the database.

When you add a Dynamic Calc member to a sparse dimension, Essbase updates the index but does not change the relevant data blocks.

Essbase can save changes to the database outline significantly faster if it does not have to restructure the database.

In the following cases, Essbase does not restructure the database or change the index (Essbase saves only the database outline, which is very fast):

- Add, delete, or move a dense dimension Dynamic Calc member.
- Change the storage property of a sparse dimension Dynamic Calc member to a nondynamic storage property.
- Rename any Dynamic Calc member.

In the following cases, Essbase does not restructure the database but does restructure the database index, which is significantly faster:

- Add, delete, or move sparse dimension Dynamic Calc members.

In the following cases, Essbase restructures the database:

- Change the storage property of a nondynamic member in a dense dimension to Dynamic Calc.
- Change the storage property of a dense dimension from Dynamic Calc member to a nondynamic value.
- Change the storage property of a nondynamic member in a sparse dimension to Dynamic Calc.

See Implicit Restructures and Explicit Restructures.

Dynamically Calculating Data in Partitions

You can define Dynamic Calc members in transparent or replicated regions of the partitions.

In a transparent partition, the definition on the remote database takes precedence over any definition on the local database. For example, if a member is tagged as Dynamic Calc in the local database but not in the remote database, Essbase retrieves the value from the remote database and does not do the local calculation.

> **Note:**
>
> When Essbase replicates data, it checks the time stamp on each source data block and each corresponding target data block. If the source data block is more recent, Essbase replicates the data in the data block. However, for dynamically calculated data, data blocks and time stamps do not exist. Therefore, Essbase always replicates dynamically calculated data.

23
Calculating Time Series Data

Time series calculations assume that you have Dynamic Time Series members defined in the outline. Calculating time series data is helpful in tracking inventory by calculating the first and last values for a time period, and in calculating period-to-date values.

- Calculating First, Last, and Average Values
- Calculating Period-to-Date Values Using Dynamic Time Series Members
- Using Dynamic Time Series Members in Transparent Partitions

The information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases.

Calculating First, Last, and Average Values

Using time balance and variance reporting tags on the dimension tagged as accounts, you can tell Essbase how to perform time balance calculations on accounts data.

Essbase usually calculates a dimension tagged as time by consolidating or calculating the formulas on the parent's children. However, you can use accounts tags, such as time balance and variance reporting tags, to consolidate a different kind of value. For example, if you tag a parent member in the accounts dimension with a time balance property of First, Essbase calculates the member by consolidating the value of the member's first child. For example, in the Sample.Basic database, the Opening Inventory member in the Measures dimension (the accounts dimension) has a time balance property of First. This member represents the inventory at the beginning of the time period. If the time period is Qtr1, Opening Inventory represents the inventory available at the beginning of Jan (the first member in the Qtr1 branch).

To use accounts tags, you must have a dimension tagged as accounts and a dimension tagged as time. You use the First, Last, and Average tags (time balance properties) and the Expense tag (variance reporting property) only on members of a dimension tagged as accounts. The dimensions you tag as time and accounts can be either dense or sparse dimensions.

For cells of time balance account members, a member in any dimension other than the time dimension that is set with the ^ consolidation operator is excluded from the Average calculation; the member is, however, included in First and Last calculations.

Formulas override time balance properties. If a member with a formula uses time balance properites, the time balance properties are ignored, and the formula is used to calculate it.

> **Note:**
>
> If you are using Intelligent Calculation, changing accounts tags in the database outline does not cause Essbase to restructure the database. You may have to tell Essbase explicitly to recalculate the required data values. See Changing Formulas and Accounts Properties: Impact on Block Status.

Specifying Accounts and Time Dimensions

When you tag a dimension as accounts, Essbase knows that the dimension contains members with accounts tags. When you tag a dimension as time, Essbase knows that this dimension is the one on which to base the time periods for the accounts tags.

As shown in the illustration, the Measures dimension is tagged as accounts, and the Year dimension is tagged as time.

Figure 23-1 Sample.Basic Outline Showing Accounts and Time Tags

```
Database: Basic (Current Alias Table: Default)
    Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Measures Accounts (Label Only)
    Product
    Market
    Scenario (Label Only)
```

See Creating a Time Dimension and Creating an Accounts Dimension.

Reporting the Last Value for Each Time Period

For an accounts dimension member, you can tell Essbase to move the last value for each time period up to the next level. To report the last value for each time period, set the member's time balance property as Last. (The tag displays as TB Last in the database outline.)

As shown in the illustration, the accounts member Ending Inventory is tagged as TB Last. Ending Inventory consolidates the value for the last month in each quarter and uses that value for that month's parent. For example, the value for Qtr1 is the same as the value for Mar.

Figure 23-2 Sample.Basic Outline Showing Last Tag

```
Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Qtr1 (+) (Dynamic Calc)
        Jan (+)
        Feb (+)
        Mar (+)
    Qtr2 (+) (Dynamic Calc)
    Qtr3 (+) (Dynamic Calc)
    Qtr4 (+) (Dynamic Calc)
Measures Accounts (Label Only)
    Profit (+) (Dynamic Calc)
    Inventory (~) (Label Only)
        Opening Inventory (+) (TB First) (Expense Reporting)
        Additions (~) (Expense Reporting)
        Ending Inventory (~) (TB Last) (Expense Reporting)
```

For information on tagging an accounts member as Last, see Setting Time Balance Properties.

By default, Essbase does not skip #MISSING or zero (0) values when calculating a parent value. You can choose to skip these values. For a discussion of how and why to skip #MISSING values, see Skipping #MISSING and Zero Values.

Reporting the First Value for Each Time Period

For an accounts dimension member, you can tell Essbase to move the first value for each time period up to the next level. To report the first value for each time period, set the member's time balance property as First. (The tag displays as TB First in the database outline.)

As shown in the illustration, the accounts member Opening Inventory is tagged as TB First. Opening Inventory consolidates the value of the first month in each quarter and uses that value for that month's parent. For example, the value for Qtr1 is the same as the value for Jan.

Figure 23-3   Sample.Basic Outline Showing First Tag

```
Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Qtr1 (+) (Dynamic Calc)
        Jan (+)
        Feb (+)
        Mar (+)
    Qtr2 (+) (Dynamic Calc)
    Qtr3 (+) (Dynamic Calc)
    Qtr4 (+) (Dynamic Calc)
Measures Accounts (Label Only)
    Profit (+) (Dynamic Calc)
    Inventory (~) (Label Only)
        Opening Inventory (+) (TB First) (Expense Reporting)
        Additions (~) (Expense Reporting)
        Ending Inventory (~) (TB Last) (Expense Reporting)
```

For information on tagging an accounts member as First, see Setting Time Balance Properties.

By default, Essbase does not skip #MISSING or zero (0) values when calculating a parent value. You can choose to skip these values. See Skipping #MISSING and Zero Values.

Reporting the Average Value for Each Time Period

For an accounts dimension member, you can tell Essbase to average values across time periods and consolidate the average up to the next level. For example, you can tell Essbase to average the values for Jan, Feb, and Mar and then use that value for the Qtr1 value. To report the average value for each time period, set the member's time balance property as Average.

For information on tagging an accounts member as Average, see Setting Time Balance Properties.

By default, Essbase does not skip #MISSING or zero (0) values when it calculates a parent value. Thus, when it calculates the average, Essbase aggregates the child values and divides by the number of children, regardless of whether the children have #MISSING or zero values. You can tell Essbase to skip #MISSING and zero values. See Skipping #MISSING and Zero Values.

Skipping #MISSING and Zero Values

You can tell Essbase how to treat #MISSING and zero (0) values when doing time balance calculations. A #MISSING value is a marker in Essbase that indicates that the data in this location does not exist, does not contain any meaningful value, or was never entered.

By default, Essbase does not skip #MISSING or 0 (zero) values when calculating a parent value.

You can override this default by setting a skip property. See Setting Skip Properties.

For example, if you tag an accounts dimension member as Last and Skip Missing, then Essbase consolidates the last nonmissing child to the parent. Consider the example below:

Table 23-1 Example of the Effects of the Skip Missing

Accounts -> Time	Jan	Feb	Mar	Qtr1
Accounts Member (**Last, Skip Missing**)	60	70	#MI	70

Tagging an account as Average and Skip Missing may produce different results from tagging that account as Average and Skip None. A calculation performed with Average and Skip None produces correct results because no data is skipped. But because grandparents with children are consolidated by summing the averages, results of a calculation on an account with Average and Skip Missing is incorrect unless you use Dynamic Calc or Two-Pass tags.

Considering the Effects of First, Last, and Average Tags

The following example shows how Essbase consolidates the time dimension based on the time balance (TB) First, Last, and Average tags on accounts dimension members.

Table 23-2 Example of the Effects of (TB) First, Last and Average

Accounts -> Time	Jan	Feb	Mar	Qtr1	Consolidation
Accounts Member1	11	12	13	36	Value of Jan + Feb + Mar
Accounts Member2 (**TB First**)	20	25	21	20	Value of Jan
Accounts Member3 (**TB Last**)	25	21	30	30	Value of Mar
Accounts Member4 (**TB Average**)	20	30	28	26	Average of Jan, Feb, Mar

Calculating Period-to-Date Values Using Dynamic Time Series Members

In Essbase, you can calculate period-to-date values. To do this, you define Dynamic Time Series members in the outline. For example, to calculate quarter-to-date values, you enable the Q-T-D member and associate it with a particular generation.

A sample use case of dynamic time series is when you want to calculate the sales values for the current quarter up to the current month. If the current month is May, using a standard calendar quarter, the quarter total is the total of the values for April and May.

In Essbase, you can calculate period-to-date values in the following ways:

- During a batch calculation, using the @PTD function
- Dynamically, when a user requests the values, using Dynamic Time Series members
- As part of an MDX query, using the DTS function

This section explains how to use Dynamic Time Series members to dynamically calculate period-to-date values. Using Dynamic Time Series members is almost always the most efficient method. For a detailed example, see Calculating Period-to-Date Values in an Accounts Dimension.

Using Dynamic Time Series Members

To calculate period-to-date values dynamically, you must use a Dynamic Time Series member for a period on the dimension tagged as time. See Specifying Accounts and Time Dimensions.

You do not create the Dynamic Time Series member directly in the database outline. Instead, you enable a predefined Dynamic Time Series member and associate it with an appropriate generation number.

For example, to calculate quarter-to-date values, you enable the Q-T-D member and associate it with the generation to which you want to apply the Dynamic Time Series member. In Sample.Basic, the generation containing quarters is generation number 2, which contains the Qtr1, Qtr2, Qtr3, and Qtr4 members. Essbase creates a Dynamic Time Series member called Q-T-D and associates it with generation 2. The Q-T-D member calculates monthly values up to the current month in the quarter.

Dynamic Time Series members are not displayed as members in the database outline. Instead, Essbase lists the currently active Dynamic Time Series members in a comment on the time dimension. In the outline below, H-T-D (history-to-date) and Q-T-D (quarter-to-date) are active. H-T-D is associated with generation 1; Q-T-D is associated with generation 2.

Figure 23-4 Sample.Basic Outline Showing Dynamic Time Series

```
Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
    Qtr1 (+) (Dynamic Calc)
    Qtr2 (+) (Dynamic Calc)
        Apr (+)
        May (+)
        Jun (+)
    Qtr3 (+) (Dynamic Calc)
    Qtr4 (+) (Dynamic Calc)
```

Essbase provides eight predefined Dynamic Time Series members:

- HTD (history-to-date)
- Y-T-D (year-to-date)
- S-T-D (season-to-date)
- P-T-D (period-to-date)
- Q-T-D (quarter-to-date)
- M-T-D (month-to-date)
- W-T-D (week-to-date)
- D-T-D (day-to-date)

These members provide up to eight levels of period-to-date reporting. How many and which members you use depends on the data and the database outline.

For example, if the database contains hourly, daily, weekly, monthly, quarterly, and yearly data, you can report day-to date (D-T-D), week-to-date (W-T-D), month-to-date (M-T-D), quarter-to-date (Q-T-D), and year-to-date (Y-T-D) information.

If the database contains monthly data for the last five years, you can report year-to-date (Y-T-D) and history-to-date (H-T-D) information, up to a specific year.

If the database tracks data for seasonal time periods, you can report period-to-date (P-T-D) or season-to-date (S-T-D) information.

You can associate a Dynamic Time Series member with any generation in the time dimension except the highest generation number, regardless of the data. For example, you can use the P-T-D member to report quarter-to-date information. You cannot associate Dynamic Time Series members with level 0 members of the time dimension.

> **Note:**
>
> Oracle recommends that you avoid assigning time balance properties (First, Last, Average, Skip Missing) to members set for dynamic calculations if you plan to use these members in Dynamic Time Series calculations. Doing so may retrieve incorrect values for the parent members in your accounts dimension.

Enabling and Disabling Dynamic Time Series Members

To use a predefined Dynamic Time Series member, you must enable the member and associate it with an appropriate generation number. See Understanding the Cube.Settings Worksheet: Dynamic Time Series.

> **Note:**
>
> The number of generations displayed depends on the number of generations in the time dimension. You cannot associate Dynamic Time Series members with the highest generation (level 0 members).

After you enable Dynamic Time Series members in the database outline, Essbase adds a comment to the dimension tagged as time; for example, the Year dimension from Sample.Basic showing H-T-D and Q-T-D defined:

```
Year Time (Active Dynamic Time Series Members: H-T-D, Q-T-D) (Dynamic Calc)
```

If required, you can specify aliases for Dynamic Time Series members. See Specifying Alias Names for Dynamic Time Series Members.

To disable Dynamic Time Series members, tell Essbase not to use the predefined member.

Specifying Alias Names for Dynamic Time Series Members

You can specify alias names for predefined Dynamic Time Series members, such as QtrToDate, for the Q-T-D Dynamic Time Series member. You can then use the alias names to retrieve the Dynamic Time Series members in Smart View or in a report.

You can create up to eight alias names for each Dynamic Time Series member. Essbase saves each alias name in the Dynamic Time Series alias table that you specify.

For information on specifying and displaying alias names, see Setting Aliases.

Applying Predefined Generation Names to Dynamic Time Series Members

When you enable a Dynamic Time Series member and associate it with a generation number, Essbase creates a predefined generation name for that generation number.

The following table lists the Dynamic Time Series members and their corresponding generation names:

Table 23-3 Dynamic Time Series Members and Corresponding Generation Names

Member	Generation Name
D-T-D	Day
H-T-D	History
M-T-D	Month
P-T-D	Period
Q-T-D	Quarter
S-T-D	Season
W-T-D	Week
Y-T-D	Year

These member and generation names are reserved for use by Essbase.

For example, in Sample.Basic, you can enable Dynamic Time Series for a generation called Quarter. Quarter contains quarterly data in the members Qtr1, Qtr2, and so on. If you enable Dynamic Time Series for Quarter, Essbase creates a Dynamic Time Series member called Q-T-D.

Retrieving Period-to-Date Values

When you retrieve a Dynamic Time Series member, you must tell Essbase the time period up to which you want to calculate the period-to-date value. This time period, known as the *latest time period*, must be a level 0 member on the time dimension.

Use the following methods to specify the latest time period:

- For a specific member, in Smart View, specify the latest period member name. Place that name after the Dynamic Time Series member or alias name. For example, Q-T-D(May) returns the quarter-to-date value by adding values for April and May.
- For a retrieval, specify the Select DTS Member option in Smart View.

In the example below, Q-T-D(May) displays the period-to-date value for May that is obtained by adding the values for Apr and May (8644 + 8929 = 17573).

Figure 23-5 Spreadsheet Showing Period-To-Date Value for May

	Measures	Product	Market	Scenario
Qtr1	24703			
Apr	8644			
May	8929			
Jun	9534			
Qtr2	27107			
Qtr3	27912			
Qtr4	25800			
Year	105522			
Q-T-D(May)	17573			

Using Dynamic Time Series Members in Transparent Partitions

To optimize query time across transparent partitions for outlines containing Dynamic Time Series members, use the configuration setting TARGETTIMESERIESOPT.

24

Develop Calculation Scripts for Block Storage Cubes

You can write Essbase calculation scripts to perform calculations other than those defined by the cube outline. You can clear and copy data, fix on subsets of data, implement variables, control flow, regulate block creation, and more. Calculation scripts can contain commands, equations, and formulas.

- Calculate a Cube with Calculation Scripts
- Learn Basic Calculation Script Syntax
- Use Calculation Commands
- Formulas in Calculation Scripts
- Substitution Variables in Calculation Scripts
- Runtime Substitution Variables in Calculation Scripts Run in Essbase
- Runtime Substitution Variables in Calculation Scripts Run in Smart View
- Clear and Copy Data
- Calculate a Subset of Data in the Cube
- Enable Calculations on Potential Blocks
- Use Calculation Scripts on Partitions
- Save, Execute, and Copy Calculation Scripts
- Check Calculation Results

The information in this chapter applies only to block storage cubes, and is not relevant to aggregate storage cubes.

Calculate a Cube with Calculation Scripts

Calculation scripts override Essbase block storage outline formulas and give you more control of the cube's calculation flow. You can control the logical calculation flow, prevent overcalculation, and clear data. Calculation scripts also enable you to to copy, export, and calculate data subsets, and to implement variables for information that changes frequently.

A calculation script contains a series of calculation commands, equations, and functions which you can use to control calculations more specifically than the flow determined by the database outline.

In a calculation script, you can perform a default calculation (CALC ALL) or a calculation of your choosing (for example, you can calculate part of a database or copy data values between members).

You must write a calculation script to perform any of the following tasks:

- Calculate a subset of a database
- Change the calculation order of the dense and sparse dimensions in a database

- Perform a complex calculation in a specific order or perform a calculation that requires multiple iterations through the data (for example, some two-pass calculations require a calculation script)
- Perform any two-pass calculation on a dimension without an accounts tag
- Calculate member formulas that differ from formulas in the database outline (formulas in a calculation script override formulas in the database outline)
- Use an API interface to create a custom calculation dynamically
- Use control of flow logic in a calculation (for example, to use the IF…ELSE…ENDIF or the LOOP…ENDLOOP commands)
- Clear or copy data from specific members.
- Define temporary variables for use in a database calculation
- Force a recalculation of data blocks after you have changed a formula or an accounts property on the database outline
- Control how Essbase uses Intelligent Calculation when calculating a database

The following calculation script calculates the Actual values from the Year, Measures, Market, and Product dimensions:

```
FIX (Actual)
   CALC DIM(Year, Measures, Market, Product);
ENDFIX
```

You can create calculation scripts by using the script editor. See Creating Calculation Scripts.

If you run a calculation script from Smart View, the file must have a `.csc` extension. However, because a calculation script is a text file, you can use MaxL to run any text file as a calculation script.

Learn Basic Calculation Script Syntax

When designing calculation scripts for Essbase block storage cubes, there are a few syntax rules to learn. Commands are generally terminated by a semicolon. Quotation marks must be used around certain member names. Essbase can help you validate the syntax.

Overall Syntax Rules

Essbase block storage cubes include a flexible set of commands that you can use to control how the cube is calculated. You can construct calculation scripts from commands and formulas.

When you create a calculation script, you must apply the following rules:

- End each formula or calculation script command with a semicolon (;). For example:

  Example 1

  ```
  CALC DIM(Product, Measures);
  ```

  Example 2

  ```
  DATACOPY Plan TO Revised_Plan;
  ```

Example 3

```
"Market Share" = Sales % Sales -> Market;
```

Example 4

```
IF (Sales <> #MISSING)
   Commission = Sales * .9;
   ELSE
      Commission = #MISSING;
ENDIF;
```

You do not need to end the following conditional or control flow commands with semicolons:

– IF
– ENDIF
– ELSE
– ELSIF
– FIX
– ENDFIX
– EXCLUDE
– ENDEXCLUDE
– LOOP
– ENDLOOP

> **Note:**
>
> Although not required, Oracle recommends ending each ENDIF statement in a formula with a semicolon.

- Enclose a member name in double quotation marks (" "), if that member name matches any of the following conditions:
  – Contains spaces.

    For example, in the following formula, Opening Inventory and Ending Inventory are enclosed in double quotation marks:

    ```
    "Opening Inventory" = "Ending Inventory" - Sales + Additions;
    ```

  – Is the same as an operator, function name, or keyword.

    See Naming Conventions in Calculation Scripts, Report Scripts, Formulas, Filters, and Substitution and Environment Variable Values.

  – Includes any nonalphanumeric character, such as a hyphen (-), asterisk (*), or slash (/).

  – Contains only numerals or starts with a numeral.

    For example: "100" or "10Prod"

  – Begins with an ampersand (&). The leading ampersand (&) is reserved for substitution variables. If a member name begins with &, enclose the name in quotation marks.

> **Note:**
>
> Do not enclose substitution variables in quotation marks in a calculation script.

  – Contains a dot (.).

    For example: 1999.Jan or .100

- If you are using an IF statement or an interdependent formula, enclose the formula in parentheses to associate it with the specified member.

  For example, the following formula is associated with the Commission member in the database outline:

```
Commission
(IF(Sales < 100)
   Commission = 0;
ENDIF;)
```

- End each IF statement in a formula with an ENDIF statement.

  For example, the previous formula contains a simple IF...ENDIF statement.

- If you are using an IF statement that is nested within another IF statement, end each IF with an ENDIF statement.

  For example:

```
"Opening Inventory"
(IF (@ISMBR(Budget))
   IF (@ISMBR(Jan))
      "Opening Inventory" = Jan;
   ELSE
      "Opening Inventory" = @PRIOR("Ending Inventory");
   ENDIF;
ENDIF;)
```

- You do not need to end ELSE or ELSEIF statements with ENDIF statements.

  For example:

```
Marketing
(IF (@ISMBR(@DESCENDANTS(West)) OR @ISMBR(@DESCENDANTS(East)))
   Marketing = Marketing * 1.5;
ELSEIF(@ISMBR(@DESCENDANTS(South)))
   Marketing = Marketing * .9;
ELSE
   Marketing = Marketing * 1.1;
ENDIF;)
```

> **Note:**
>
> If you use ELSE IF (with a space) rather than ELSEIF (one word) in a formula, you must supply an ENDIF for the IF statement.

- End each FIX statement with an ENDFIX statement.

  For example:

  ```
  FIX(Budget,@DESCENDANTS(East))
     CALC DIM(Year, Measures, Product);
  ENDFIX
  ```

- End each EXCLUDE statement with an ENDEXCLUDE statement.

When you write a calculation script using the Essbase web interface, you can use the syntax checker to validate it.

Use Comments in Calculation Scripts

You can include comments to annotate calculation scripts. Comments are ignored when the calculation script runs.

To include a comment, start the comment with /* and end the comment with */. For example:

```
/* This calculation script comment
   spans two lines. */
```

Check the Syntax

Essbase includes a syntax checker that flags syntax errors (such as a mistyped function name) in a calculation script.

If syntax errors are not found, Essbase indicates the syntax check succeeded.

If syntax errors are found, Essbase indicates the syntax check failed and displays one error at a time. Typically, an error message includes the line number in which the error occurred and a brief description.

Refere to Create Calculation Scripts

> **Note:**
>
> The syntax checker cannot determine semantic errors, which occur when a calculation script does not work as you expect. To find semantic errors, run the calculation and check the results to ensure they are as you expect. Refer to Check Calculation Results.

Use Calculation Commands

Essbase calculation commands help you flexibly calculate the data in your block storage cube. You can design scripts to calculate data based on the outline, calculate iteratively or in parallel, restrict to subsets, and declare variables. You can customize the behavior of different features from within the calc script.

The topics in this section discuss calculation commands, grouped by functionality.

- Calculate the Essbase Outline
- Control the Flow of Calculations
- Declare Data Variables During Calculation
- Specify Global Settings for a Calculation

Calculate the Essbase Outline

The following Essbase calculation commands calculate a block storage cube based on the structure and formulas in its outline.

Table 24-1 List of Commands for Calculating the Cube Outline

Command	Calculation
CALC ALL	The entire cube, based on the outline
CALC DIM	A specified dimension or dimensions
CALC TWOPASS	All members tagged as two-pass on the dimension tagged as accounts **Note:** Do not use two-pass calculation with hybrid mode cubes. Only use solve order.
membername	The formula applied to a member in the cube outline, where *membername* is the name of the member to which the formula is applied
CALC AVERAGE	All members tagged as Average on the dimension tagged as accounts
CALC FIRST	All members tagged as First on the dimension tagged as accounts
CALC LAST	All members tagged as Last on the dimension tagged as accounts
CCONV	Currency conversions

Control the Flow of Calculations

Control flow commands in block storage Essbase calculation scripts help you iterate calculations, calculate in parallel, or restrict calculations to a subset of the cube.

The following commands manipulate the flow of calculations:

Table 24-2 List of Commands to Control the Flow of Calculations

Command	Calculation
FIX...ENDFIX	Calculate a subset of a database by inclusion
EXCLUDE...ENDEXCLUDE	Calculate a subset of a database by exclusion
LOOP...ENDLOOP	Specify the number of times that commands are iterated
FIXPARALLEL...ENDFIXPARALLEL	Enable parallel calculation on a block of commands

You can also use the IF and ENDIF commands to specify conditional calculations.

> **Note:**
>
> Essbase does not allow branching from one calculation script to another calculation script.

See Also

Control Flow Commands

Declare Data Variables During Calculation

Use data declaration commands in Essbase block storage calculation scripts to declare and set the initial values of temporary variables, which store the results of intermediate calculations.

You can also use substitution variables in a calculation script. See Substitution Variables in Calculation Scripts.

Table 24-3 List of Commands for Declaring Data Variables

Command	Calculation
ARRAY	Declare one-dimensional array variables
VAR	Declare a temporary variable that contains a single value
THREADVAR	Declare a temporary variable that can be used within a FIXPARALLEL...ENDFIXPARALLEL block.

Values stored in temporary variables exist only while the calculation script is running. You cannot report on the values of temporary variables.

Variable and array names are character strings that contain any of the following characters:

- Letters a–z
- Numerals 0–9
- Special characters: $ (dollar sign), # (pound sign), and _ (underscore)

Typically, arrays are used to store variables as part of a member formula. The size of the array variable is determined by the number of members in the corresponding dimension. For example, if the Scenario dimension has four members, the following command creates an array called `Discount` with four entries:

```
ARRAY Discount[Scenario];
```

You can use multiple arrays at a time.

Specify Global Settings for a Calculation

SET commands for Essbase calculation scripts help define behavior of different functional aspects of block storage calculations. The first occurrence of a SET command stays in effect until the next occurrence of the same SET command in the script.

The following commands define calculation behavior:

Table 24-4 List of Commands for Defining Calculation Behavior

Command	Calculation
SET AGGMISSG	Specify how Essbase treats #MISSING values during a calculation.
SET CACHE	Adjust the default calculator cache size. Refer to Essbase Caches and Calc Performance.
SET CALCPARALLEL	Enable parallel calculation. Refer to Enabling CALCPARALLEL Parallel Calculation.
SET CALCTASKDIMS	Increase the number of dimensions used to identify tasks for parallel calculation. Refer to Identifying Additional Tasks for Parallel Calculation.
SET CLEARUPDATESTATUS	Control how Essbase marks data blocks for Intelligent Calculation. Refer to Using the SET CLEARUPDATESTATUS Command.
SET CREATEBLOCKONEQ	Turn on and turn off the Create Blocks on Equation setting, which controls the creation of blocks when you assign nonconstant values to members of a sparse dimension. Refer to Nonconstant Values Assigned to Members in a Sparse Dimension.
SET CREATENONMISSINGBLK	Enable calculations on potential data blocks and save these blocks when the result is not #MISSING.
SET FRMLBOTTOMUP	Optimize the calculation of sparse dimension formulas in large database outlines. Refer to Formulas on Sparse Dimensions in Large Outlines.
SET MSG SET NOTICE SET TRACE	Display messages to monitor and trace a calculation.
SET RUNTIMESUBVARS	Declare runtime substitution variables that are used in a calculation script. Refer to Runtime Substitution Variables in Calculation Scripts Run in Essbase and Runtime Substitution Variables in Calculation Scripts Run in Smart View
SET UPDATECALC	Turn Intelligent Calculation On or Off.

In the following calculation script, Essbase displays messages at the detail level (`SET MSG DETAIL;`) when calculating the Year dimension and displays messages at the summary level (`SET MSG SUMMARY;`) when calculating the Measures dimension:

```
SET MSG DETAIL;
CALC DIM(Year);
```

SET MSG SUMMARY;

CALC DIM(Measures);

Some SET calculation commands trigger additional passes through the database.

In the following calculation script, Essbase calculates member combinations for Qtr1 with SET AGGMISSG turned on, and then does a second calculation pass through the database and calculates member combinations for East with SET AGGMISSG turned off:

```
SET AGGMISSG ON;
Qtr1;
SET AGGMISSG OFF;
East;
```

See Also

SET Commands

Formulas in Calculation Scripts

Essbase formulas calculate mathematical and positional relationships between members in a database outline. You can apply them to members in the outline, and/or use them in calculation scripts for strict control of your calculations.

When you define Essbase formulas in calculation scripts, they take precedence over any formulas applied to members in the cube outline. Using formulas in this way gives your batch calculations additional flexibility beyond the relationships encoded into the cube outline.

You can write formulas to calculate using basic equations, conditional equations, and interdependent formulas.

In an Essbase calculation script, you can perform both of these operations:

- Calculate an existing member formula on the cube outline
- Define a new formula that lasts for the duration of the calculation script's execution

To calculate an existing formula that is applied to a member in the outline, use the member name followed by a semicolon (;). For example, the following command in a calc script calculates the formula that is applied to the Variance member in the outline:

```
Variance;
```

To override values that result from outline calculation, you can manually apply a formula in the calculation script. For example, the following formula in a calculation script cycles through the cube, adding the values in the members Payroll, Marketing, and Misc, and placing the result in the Expenses member. This formula overrides any formula placed on the Expenses member in the outline:

```
Expenses = Payroll + Marketing + Misc;
```

> **Note:**
>
> You cannot apply formulas to shared members or label only members.

- Basic Equations in Essbase Calc Scripts
- Conditional Equations in Essbase Calc Scripts
- Interdependent Formulas in Essbase Calc Scripts
- Calculation Script Formulas and Intelligent Calculation Status
- Group Formulas and Dimensions in the Script

See also: Develop Formulas for Block Storage Cubes.

Basic Equations in Essbase Calc Scripts

In Essbase calculation scripts, you use equations to assign values to cube outline members. The assignment lasts for the duration of the calculation script's execution.

The syntax for an equation in an Essbase calculation script is:

```
member = mathematical_expression;
```

member is a member name from the outline, and *mathematical_expression* is any valid mathematical expression.

Essbase evaluates the expression and assigns the value to the specified member.

In the following example, Essbase cycles through the database, subtracting the values in COGS from the values in Sales, and placing the result in Margin:

```
Margin = Sales - COGS;
```

In this example, Essbase cycles through the database, subtracting the values in Cost from the values in Retail, calculating the resulting values as a percentage of the values in Retail, and placing the results in Markup:

```
Markup = (Retail - Cost) % Retail;
```

You can also use the > (greater than) and < (less than) logical operators in equations.

In the following example, if February sales are greater than January sales, Sales Increase Flag results in a value of 1; if false, the result is a value of 0:

```
Sales Increase Flag = Sales -> Feb > Sales -> Jan;
```

Conditional Equations in Essbase Calc Scripts

You can use conditional equations in Essbase calculation script formulas to define if/else logic and control the flow of events during calculation.

When you use an IF statement as part of a member formula in a calculation script, you must:

- Associate the IF statement with a single member
- Enclose the IF statement in parentheses

In the following example, the entire IF…ENDIF statement is enclosed in parentheses and associated with the Profit member, `Profit (IF(...)...)`:

```
Profit
(IF (Sales > 100)
   Profit = (Sales - COGS) * 2;
ELSE
   Profit = (Sales - COGS) * 1.5;
ENDIF;)
```

Essbase cycles through the database and performs the following calculations:

1. The IF statement checks whether the value of Sales for the current member combination is greater than 100.
2. If Sales is greater than 100, Essbase subtracts the value in COGS from the value in Sales, multiplies the difference by 2, and places the result in Profit.
3. If Sales is less than or equal to 100, Essbase subtracts the value in COGS from the value in Sales, multiplies the difference by 1.5, and places the result in Profit.

For more information about conditional commands and controlling the flow of calc scripts, see Calculation Command Groups

Interdependent Formulas in Essbase Calc Scripts

As with formulas in the outline, a formula in a calculation script calculates mathematical relationships between members in the Essbase cube outline.

Formulas are considered *interdependent* if their evaluations depend on the value of other members of the same dimension.

A good example of an interdependent formula in Essbase is a cash flow evaluation, in which the opening inventory is dependent on the closing inventory from the previous month.

When you use an interdependent formula in a calculation script, keep in mind that the same rules apply as for the IF statement. You must:

- Associate the formula with a single member
- Enclose the formula in parentheses

In the following example, the entire interdependent formula is enclosed in parentheses and associated with the Opening Inventory member:

```
"Opening Inventory"
(IF(NOT @ISMBR (Jan))
   "Opening Inventory" = @PRIOR("Ending Inventory");
```

```
ENDIF;)
"Ending Inventory" = "Opening Inventory" - Sales + Additions;
```

Calculation Script Formulas and Intelligent Calculation Status

Formulas on sparse dimension members that contain relationship or financial functions always cause recalculation, even if Essbase Intelligent Calculation marks the blocks as clean.

Assume that you have a formula on a sparse dimension member, and the formula contains either of the following type of function:

- Relationship (for example, @PRIOR or @NEXT)
- Financial (for example, @NPV or @INTEREST)

Essbase always recalculates the data block that contains the formula, even if the block is marked as clean for the purposes of Intelligent Calculation.

See Calculating Data Blocks and Intelligent Calculation for Block Storage Cubes.

Group Formulas and Dimensions in the Script

You may achieve significant calculation performance improvements by carefully grouping Essbase member formulas and dimensions in a calculation script.

Calculate a Series of Member Formulas

When you calculate formulas, be sure to use parentheses correctly.

In the following example, incorrectly placed parentheses causes Essbase to perform two calculation passes through the cube: once calculating the formulas on the members Qtr1 and Qtr2; and once calculating the formula on Qtr3:

```
(Qtr1;
Qtr2;)
Qtr3;
```

In contrast, the following configurations cause Essbase to cycle through the cube only once, calculating the formulas on the members Qtr1, Qtr2, and Qtr3:

```
Qtr1;
Qtr2;
Qtr3;
```

or

```
(Qtr1;
Qtr2;
Qtr3;)
```

Similarly, the following formulas cause Essbase to cycle through the cube once, calculating both formulas in one pass:

```
Profit = (Sales - COGS) * 1.5;
Market = East + West;
```

Calculate a Series of Dimensions

When calculating a series of dimensions, you can optimize performance by grouping the dimensions wherever possible.

For example, the following formula causes Essbase to cycle through the cube only once:

```
CALC DIM(Year, Measures);
```

In contrast, the following syntax causes Essbase to cycle through the cube twice, once for each CALC DIM command:

```
CALC DIM(Year);
CALC DIM(Measures);
```

Substitution Variables in Calculation Scripts

Use Essbase substitution variables in calc scripts to reference information that changes frequently.

When you include a substitution variable in a calculation script, Essbase replaces the substitution variable with the value you specified for the substitution variable. Substitution variables are useful, for example, when you reference information or lists of members that change frequently.

You can create substitution variables at the server, application, and database levels. To use a substitution variable in a calculation script, the substitution variable must be available to the calculation script. For example, a database-level substitution variable is available only to calculation scripts within the cube; a server-level substitution variable is available to any calculation script on the server.

In a calculation script, insert an ampersand (&) before a substitution variable. Essbase treats any string that begins with a leading ampersand as a substitution variable, replacing the variable with its assigned value before parsing the calculation script.

For example, in Sample.Basic, to calculate Qtr1 as the current quarter:

- Create a substitution variable for the current quarter (`&CurQtr`) and assign it the value `Qtr1`
- Create a calculation script that uses the `&CurQtr` substitution variable. For example:

  ```
  FIX(&CurQtr)
     CALC DIM(Measures, Product);
  ENDFIX
  ```

You can also use environment variables as placeholders for user-specific system settings.

Also see Runtime Substitution Variables in Calculation Scripts Run in Essbase.

Runtime Substitution Variables in Calculation Scripts Run in Essbase

Runtime substitution variables (RTSVs) enable you to dynamically reference member names in Essbase calculation scripts instead of hard coding them. A runtime substitution variable can be included in a calculation script wherever substitution variables are allowed.

In a calculation script, an ampersand (&) must precede the name of the runtime substitution variable.

Every runtime substitution variable used in a calculation script must be declared in the SET RUNTIMESUBVARS calculation command, with a name and default value.

> **Note:**
>
> If a default value is not included in the runtime substitution variable declaration in SET RUNTIMESUBVARS, an error occurs when the calculation script is validated. Oracle recommends that you provide a default value to avoid the validation error and, when running the calculation script, provide the expected value. However, if you do not provide a default value, you can still provide a value at runtime using the execute calculation MaxL statement with the **with runtimesubvars** grammar.

A description of the runtime substitution variable's data type and data input limit is a string in the <RTSV_HINT>*rtsv_description*</RTSV_HINT> tag. This tag is optional when the calculation script with runtime substitution variables is run in Essbase; see Hints for Runtime Substitution Variables in Calculation Scripts Run in Essbase. The <RTSV_HINT> tag, with additional metadata, is required when the calculation script with runtime substitution variables is run in Smart View. See Runtime Substitution Variables in Calculation Scripts Run in Smart View.

In this example of SET RUNTIMESUBVARS, three runtime substitution variables are declared: myMarket, salesNum, and pointD. Default values are specified for each runtime substitution variable (for example, the value of myMarket is `"New York"`). This example applies to a calculation script that is run in Essbase:

```
SET RUNTIMESUBVARS
{
   myMarket = "New York";
   salesNum = 10;
   pointD = "Actual"->"Final";
};
```

At runtime, the default values that are specified in the SET RUNTIMESUBVARS command can be overwritten, using one of these methods:

- execute calculation MaxL statement with the **with runtimesubvars** grammar, in which runtime substitution variables are specified as a string of key/value pairs.

Using the SET RUNTIMESUBVARS example above, at runtime you can overwrite the salesNum default value of 10 with 500 by using the following MaxL statement:

```
execute calculation appname.dbname.calcScriptName with runtimesubvars
'salesNum=500';
```

Using this MaxL statement also allows you to provide values for runtime substitution variables that do not have a default value in the SET RUNTIMESUBVARS declaration.

- An API call in which runtime substitution variables are specified as a string of key/value pairs: IEssCube.calcFileWithRunTimeSubVars (Java API) or EssCalcWithRuntimeSubVars (C API)
- An API call in which runtime substitution variables can be specified in a text file on the client computer or as a string of key/value pairs: IEssCube.calcFileWithRunTimeSubVarFile (Java API) or EssCalcFileWithRuntimeSubVars (C API)

When specifying runtime substitution variables as a string of key/value pairs, the string must be enclosed with single quotation marks, and key/value pairs must be separated by a semicolon, including a semicolon after the last runtime substitution variable in the string and before the terminal single quotation mark. In this example of a runtime substitution variable string, the name and value of four runtime substitution variables are specified (for example, the value of the runtime substitution variable named "a" is 100):

```
'a=100;b=@CHILDREN("100");c="Actual"->"Final";d="New York";'
```

When specifying runtime substitution variables in a text file, create the text file with an `.rsv` extension on the client computer. (Essbase does not support runtime substitution variable files located on the Essbase Server computer.) Each line in the file specifies one runtime substitution variable as a key/value pair and must end with a semicolon. In this example of an `.rsv` file, the name and value of four runtime substitution variables are specified:

```
a=100;
b=200;
c=@CHILDREN("100");
d=@TODATE("DD/MM/YY","10/11/12");
```

When a calculation is executed, runtime substitution variable values are determined in the following order:

1. Values specified through the execute calculation MaxL statement with the **with runtimesubvars** grammar, or the APIs (IEssCube.calcFileWithRunTimeSubVars or IEssCube.calcFileWithRunTimeSubVarFile Java APIs; EssCalcWithRuntimeSubVars or EssCalcFileWithRuntimeSubVars C APIs).
2. Default values specified in the SET RUNTIMESUBVARS calculation command.

Consider these guidelines when using runtime substitution variables:

- If you declare a runtime substitution variable in SET RUNTIMESUBVARS but do not use the runtime substitution variable in the calculation script, Essbase ignores the unused runtime substitution variable declaration (no warning or exception is generated).
- Runtime substitution variables have a higher precedence than substitution variables. Therefore, if a substitution variable and a runtime substitution variable have the same name (for example, myProduct), the value of the runtime substitution variable overwrites the value of the substitution variable.

- If multiple runtime substitution variables have the same name but have different values, only the value of the first instance of the runtime substitution variable is used; all other subsequent values are ignored.

The rules for setting names and values for runtime substitution variables are the same as for substitution variables. See Rules for Setting Substitution Variable Names and Values.

Hints for Runtime Substitution Variables in Calculation Scripts Run in Essbase

When you design Essbase calculation scripts that use runtime substitution variables, you can use the RTSV_HINT to contain text to prompt users to input values at runtime.

The information in this topic applies to running a calculation script with runtime substitution variables in Essbase jobs or API, rather than in Smart View.

In the SET RUNTIMESUBVARS calculation command, the runtime substitution variable declaration can include the <RTSV_HINT>*rtsv_description*</RTSV_HINT> tag, in which *rtsv_description* is a string that describes the data type and data input limit (for example, an integer not greater than 100) for the runtime substitution variable. The *rtsv_description* string is not used in the calculation, but can be helpful as a user prompt.

API calls can retrieve all of the information (name, default value, and description) that is specified in the runtime substitution variable declaration in SET RUNTIMESUBVARS. For example, you can return this information using the REST API **Get Essbase Script RTSVs** endpoint, the IEssIterator.getCalcFileRunTimeSubVars or IEssIterator.getCalcRunTimeSubVars Java API methods, or the EssGetRuntimeSubVars C API. The *rtsv_description* string can then be used to prompt a user to input a value at runtime or to validate input data before passing the value to the calculation script.

In this example of SET RUNTIMESUBVARS, each declaration specifies the name, default value, and description of the runtime substitution variable:

```
SET RUNTIMESUBVARS
{
   myMarket = "New York" <RTSV_HINT>myMarket: Input the value as a member
name, such as "New York"</RTSV_HINT>;
   salesNum = 10 <RTSV_HINT>salesNum: Input the value as an integer, such as
100</RTSV_HINT>;
   pointD = "Actual"->"Final" <RTSV_HINT>pointD: Input the value as a member
combination, such as "Actual"->"Final"</RTSVVAR_HINT>;
};
```

Logging Runtime Substitution Variables

To log runtime substitution variables that are used in a calculation script, set the ENABLERTSVLOGGING configuration setting to TRUE. Logging can be implemented at the Essbase Server, application, or database level.

Runtime substitution variable log entries are written to the application log file. Essbase writes one entry to the application log for each string of key/value pairs (or a list of key/value pairs specified in an `.rsv` file when using the IEssCube.calcFileWithRunTimeSubVarFile Java API or EssCalcFileWithRuntimeSubVars C API).

In the following example, two runtime substitution variables (Entity and Currency) and their values are logged in one entry:

```
Executing calc script 'calcprofit.csc' with runtime substitution variables
{Entity = "MyCompany"; Currency = "USD";}
```

Runtime Substitution Variables in Calculation Scripts Run in Smart View

You can use runtime substitution variables (RTSVs) in Essbase calc scripts that you design for execution in Smart View. The RTSVs enable the calculation's region of focus to be dynamic, depending on the active grid context (POV).

To use a calculation script that includes runtime substitution variables in Smart View, the runtime substitution variable declaration in the SET RUNTIMESUBVARS calculation command must include the `<RTSV_HINT>` tag. Additionally, the `<RTSV_HINT>` tag must include the `<svLaunch>` tag. Typically, the `<svLaunch>` tag includes additional XML tags that provide metadata for executing the calculation script in Smart View.

Use the following syntax to define a runtime substitution variable definition for use in Smart View:

```
SET RUNTIMESUBVARS
{
   rtsvName = POV
   <RTSV_HINT>
      <svLaunch>
         <description>rtsv_description</description>
         <type>member | string | number</type>
         <dimension>dimName</dimension>
         <choice>single | multiple</choice>
         <allowMissing>true | false</allowMissing>
      </svLaunch>
   </RTSV_HINT>
};
```

In the definition above, the runtime substitution variable name is `rtsvName`, but you can use any name. The variable's value must be set to `POV`, to indicate that only the current data slice present in the spreadsheet grid should be calculated.

The `<RTSV_HINT>` element must also be part of the variable declaration. Its contents include:

- `<description>`: A descriptor to help Smart View users understand the member selection
- `<type>`: A specification of whether the variable is for a member, string, or number
- `<allowMissing>`: `true` if missing data should be included; `false` otherwise
- `<dimension>`: The name of the dimension from which this variable pulls information
- `<choice>`: `single` if only one selection may be passed to the runtime substitution variable; `multiple` otherwise.

For a full description of the XML tags listed above, refer to XML Tag Reference—Calculation Scripts with Runtime Substitution Variables for Smart View.

In Smart View, when you select a calculation script that includes runtime substitution variables on the **Calculation Scripts** dialog box, the **Runtime Prompts** area is populated with fields based on how the runtime substitution variables are defined in the SET RUNTIMESUBVARS calculation command. You can run the calculation script as defined or you can use the runtime prompts to enter different variable information based on the data type.

> **Note:**
>
> In Smart View, you cannot run a calculation script that includes runtime substitution variables if the SET RUNTIMESUBVARS command does not include the `<RTSV_HINT><svLaunch>...</svLaunch></RTSV_HINT>` tags.

Example: Runtime Substitution Variable Set to POV

To set a runtime substitution variable to the Smart View active grid context (POV), the value of the runtime substitution variable must be set to `POV`, and the data type must be set to `member`.

By default, the calculation script uses the active member in the POV at runtime unless the Smart View user specifies a different member in the runtime prompt.

In this example, assume that the database has these dimensions: Account, Entity, Period and Scenario. Three runtime substitution variables are defined (named rtsvEntity, rtsvScenario, and rtsvPeriod), and the value for each variable is set to POV. The runtime substitution variables are referenced in the FIX statement as &rtsvEntity, &rtsvScenario, and &rtsvPeriod.

```
SET RUNTIMESUBVARS {
   rtsvEntity = POV
      <RTSV_HINT>
         <svLaunch>
            <description>Entities to Copy</description>
            <type>member</type>
            <dimension>Entity</dimension>
            <allowMissing>false</allowMissing>
            <choice>multiple</choice>
         </svLaunch>
      </RTSV_HINT>;
   rtsvScenario = POV
      <RTSV_HINT>
         <svLaunch>
            <description>Scenarios to Copy</description>
            <type>member</type>
            <allowMissing>false</allowMissing>
            <dimension>Scenario</dimension>
            <choice>multiple</choice>
         </svLaunch>
      </RTSV_HINT>;
   rtsvPeriod   = POV
      <RTSV_HINT>
         <svLaunch>
            <description>Period to Copy</description>
            <type>member</type>
            <allowMissing>false</allowMissing>
            <dimension>Period</dimension>
```

```
            <choice>single</choice>
         </svLaunch>
      </RTSV_HINT>;
}
FIX(&rtsvEntity, &rtsvScenario, &rtsvPeriod)
"Opening Balance" (
@PREV("Closing Balance");
)
```

Below the variable declarations is a FIX block, which selects the calculation's region of focus, and executes a command to set the period's opening balance. The runtime substitution variables are passed into the FIX block using this syntax: `&varname`. The calculation's region of focus is, therefore, dynamic, depending on the POV in Smart View.

> **Note:**
>
> When using the Essbase jobs interface, the MaxL execute calculation statement, or APIs (such as the REST API Execute Jobs endpoint), you cannot run a calculation script that includes a runtime substitution variable that is set to POV. As these are dependent on current Smart View grid context, they must be run from Smart View.

XML Tag Reference—Calculation Scripts with Runtime Substitution Variables for Smart View

When you design Essbase calculation scripts with runtime substitution variables that are designed to execute based on Smart View POV/grid context, you need to use some XML tags which are explained here.

- `<RTSV_HINT>`

  Required tag for defining runtime substitution variables for use in Smart View.

- `<svLaunch>`

  Required tag that indicates that the runtime substitution variable is defined for use in Smart View.

  This tag is the parent tag for these tags: `<description>`, `<type>`, `<dimension>`, `<choice>`, and `<allowMissing>`.

- `<description>`*`rtsv_description`*`</description>`

  The *rtsv_description* is a string that describes the runtime substitution variable. The string is not used in the calculation.

- `<type>`*`value`*`</type>`

  Valid data type values are:

  – member—The runtime substitution variable value must be defined as a member (a single member name or a comma separated list of member names), or POV.

> **Note:**
>
> Member names must be enclosed in quotes; for example, `"New York"` (single member) or `"New York","Florida"` (a comma separated list of member names).
>
> If the runtime substitution variable value is set to POV, the `<type>` value must be member. Also see the `<choice>` tag.
>
> Using Essbase jobs or MaxL, you cannot run a calculation script that includes a runtime substitution variable that is set to the POV. These calculation scripts are meant to be run from Smart View, as POV is dependent on the current grid context to determine the slice.

  – string—The runtime substitution variable value can be defined as a single member name, a comma separated list of member names (for example, "New York","Florida"), or a date.

    When using the string data type, the **Member Selection** dialog box is not available; therefore, the Smart View user must be sure to use the correct syntax (enclosing a member name in quotes, separating multiple member names with a comma, or, for a date, matching the format of the date string to the format that is defined in the calculation script—mm-dd-yyyy or dd-mm-yyyy).

  – number—The runtime substitution variable value must be defined as a number

- `<dimension>`*`dimName`*`</dimension>`

  Name of the dimension.

  > **Note:**
  >
  > This XML tag is supported only if the `<type>` value is member.

- `<choice>`*`value`*`</choice>`

  Valid choice values are:

  – single:

    * If there is a single member on the grid or POV, that member is used.
    * If a dimension is on the POV, the active member is used.
    * If a dimension is on the POV and there are multiple members, an error occurs.

  – multiple—All dimension members on the grid or the POV are used.

  > **Note:**
  >
  > This XML tag is supported only if the `<type>` value is member.

- `<allowMissing>`*`boolean`*`</allowMissing>`

  Specifies whether to allow or suppress data cells for which no data exists in the database.

  Valid values: true and false.

Clear and Copy Data

You can clear a subset of data from an Essbase block storage cube using CLEARDATA and CLEARBLOCK commands. You can copy data values from one set of members to another set of members using DATACOPY.

Clear Data

The following commands clear data:

Table 24-5 List of Commands for Clearing Data

Command	Calculation
CLEARDATA	Change the values of the cells you specify to #MISSING; the data blocks are not removed. Use the FIX command with the CLEARDATA command to clear a subset of a database.
CLEARBLOCK	Remove the entire contents of a block, including all the dense dimension members. Essbase removes the entire block, unless CLEARBLOCK is inside a FIX command on members within the block.
CLEARBLOCK UPPER	Remove consolidated level blocks.
CLEARBLOCK NONINPUT	Remove blocks containing derived values. Applies to blocks that are completely created by a calculation operation, not to blocks into which any values were loaded.
CLEARBLOCK EMPTY	Remove empty blocks.

The following calculation script command yields different results depending on whether the Scenario dimension is dense or sparse:

```
FIX(Actual)
   CLEARBLOCK NONINPUT;
ENDFIX
```

- Dense: The command removes all data cells that do not contain input data values and that intersect with the member Actual from the Scenario dimension.
- Sparse: The command removes only the blocks whose Scenario dimension member is Actual.

The following formula clears all the Actual data values for Colas:

```
CLEARDATA Actual -> Colas;
```

Copy Data

The DATACOPY calculation command copies data cells from one range of members to another range of members in a database. The two ranges must be the same size. For example, the following formula copies Actual values to Budget values:

```
DATACOPY Actual TO Budget;
```

You can use the FIX…ENDFIX command to copy a subset of values. For example, the following formula copies Actual values to Budget values for the month of January only:

```
FIX (Jan)
   DATACOPY Actual TO Budget;
ENDFIX
```

See also Use the FIX Command.

Calculate a Subset of Data in the Cube

To avoid calculating more than you need to, calculate a subset of data in the Essbase database. You can indicate subsets using any combination of member set functions, FIX statements to specify inclusions, and EXCLUDE statements to specify exclusions.

To calculate a subset of data in the cube, use one of the following methods:

- Create a formula using member set functions to calculate lists of members.
- Use the FIX...ENDFIX commands to calculate a range of values by inclusion.
- Use the EXCLUDE...ENDEXCLUDE commands to calculate a range of values by exclusion.

> **Note:**
>
> When Intelligent Calculation is enabled, the newly calculated data blocks are not marked as clean after a partial calculation of a database. When you calculate a subset of a database, you can use the SET CLEARUPDATESTATUS AFTER command to ensure that the newly calculated blocks are marked as clean. Using this command ensures that Essbase recalculates the database as efficiently as possible using Intelligent Calculation.

Calculate Lists of Members

Essbase member set calculation functions generate a list of members that is based on a member you specify.

For example, the @IDESCENDANTS function generates a list of all the descendants of a specified member. When you use a member set function in a formula, Essbase generates a list of members before calculating the formula.

In the following example, using the @IDESCENDANTS command on the member Total Expenses generates a list of these members—Total Expenses, itself, and its descendants, which are Marketing, Payroll, and Misc:

```
@IDESCENDANTS("Total Expenses");
```

See Also

Member Set Functions

Use the FIX Command

Use the FIX command in Essbase block storage calc scripts to define which members to include in the calculation. FIX blocks enable you to optimize performance by avoiding overcalculation. FIX also enables you to calculate separate portions of the cube using different formulas.

A FIX block in your calc script helps you define slices of data to include in the calculation.

For example, the following FIX block calculates only the Budget values for only the descendants of East (New York, Massachusetts, Florida, Connecticut, and New Hampshire):

```
FIX(Budget,@DESCENDANTS(East))
   CALC DIM(Year, Measures, Product);
ENDFIX
```

The following example fixes on member combinations for the children of East that have a UDA of New Mkt:

```
FIX(@CHILDREN(East) AND @UDA(Market,"New Mkt"))
   Marketing = Marketing * 1.1;
ENDFIX
```

The following example uses a wildcard match (???) to fix on member names that end in the characters -10, which are members 100-10, 200-10, 300-10, and 400-10:

```
FIX(@MATCH(Product, "???-10"))
   Price = Price * 1.1;
ENDFIX
```

When you use the FIX command only on a dense dimension, Essbase retrieves the entire block that contains the required value or values for the members that you specify. I/O is not affected, and the calculation performance time is improved.

When you use the FIX command on a sparse dimension, Essbase retrieves the block for the specified sparse dimension members. I/O may be greatly reduced.

Essbase cycles through the database once for each FIX command that you use on dense dimension members. When possible, combine FIX blocks to improve calculation performance.

For example, by using one FIX command, the following calculation script causes Essbase to cycle through the database only once, calculating both the Actual and the Budget values:

```
FIX(Actual,Budget)
   CALC DIM(Year, Measures);
ENDFIX
```

In contrast, by using two FIX commands, the following calculation script causes Essbase to cycle through the database twice: once calculating the Actual data values and once calculating the Budget data values:

```
FIX(Actual)
   CALC DIM(Year, Measures);
ENDFIX
FIX(Budget)
   CALC DIM(Year, Measures);
ENDFIX
```

You cannot FIX on a subset of a dimension that you calculate within a FIX command. For example, the following calculation script returns an error message because the CALC DIM operation calculates the entire Market dimension, although the FIX above it fixes on specific members of the Market dimension:

```
FIX(@CHILDREN(East) AND @UDA(Market,"New Mkt"))
   CALC DIM(Year, Measures, Product, Market);
ENDFIX
```

FIX commands can be nested within other FIX command blocks. However, using nested FIX commands incorrectly can result in incorrect results. For example, the intent of the following calculation script is to assign 1 to all children of East and then assign 2 to New York:

```
FIX (@CHILDREN(EAST))
   "100-10"=1;
      FIX ("New York")
         "100-10"=2;
      ENDFIX
ENDFIX
```

However, the nested FIX command fixes on a subset of the dimension that is specified by the FIX command above it (which is not allowed); therefore, the script assigns 2 to all children of East because the script runs as if it were written as:

```
FIX (@CHILDREN(EAST),''New York'')
   "100-10"=1;
   "100-10"=2;
ENDFIX
```

Rather than using nested FIX commands, use two separate FIX command blocks. For example:

```
FIX (@CHILDREN(EAST))
   "100-10"=1;
ENDFIX
```

```
FIX ("New York")
   "100-10"=2;
ENDFIX
```

The variable (*varName*) that is defined by a VAR calculation command cannot be used within the FIX member statement. The FIX members are evaluated before the calculation is executed, and variables are evaluated during runtime after the FIX statement is set. Because variables can change during the calculation execution, you cannot use the variable as part of the FIX statement. The following example shows the incorrect use of the variable in the FIX member statement:

```
VAR varName=1;
FIX (@relative(@memberat(@List("Product1","Product2"),varName),0))
   COMMANDS;
ENDFIX
```

See Also

FIX…ENDFIX

Use the Exclude Command

Use the EXCLUDE..ENDEXCLUDE command block in Essbase block storage calc scripts to define which members to exclude from the calculation.

Sometimes it is easier to specify which members not to include in a calculation than to define which members to include.

Specifying members that should not be calculated in an EXCLUDE…ENDEXCLUDE command may be simpler than specifying a complex combination of member names in a FIX…ENDFIX command.

See Also

EXCLUDE…ENDEXCLUDE

Enable Calculations on Potential Blocks

When you use a formula on a dense member in a dense dimension, if the resultant values are from a dense dimension and the operand or operands are from a sparse dimension, Essbase does not automatically create the required blocks.

If you need to create the blocks, you can design Essbase calculation scripts to copy values, create blocks, and limit creation of unneeded values.

In the following example, assume that you want to create budget sales and expense data from existing actual data. Sales and Expenses are members in the dense Measures dimension; Budget and Actual are members in the sparse Scenario dimension.

```
FIX(Budget)
   (Sales = Sales -> Actual * 1.1;
   Expenses = Expenses -> Actual * .95;)
ENDFIX
```

Sales and Expenses, the results of the equations, are dense dimension members; the operand, Actual, is in a sparse dimension. Because Essbase executes dense member formulas only on existing data blocks, the calculation script does not create the required data blocks and Budget data values are not calculated for blocks that do not already exist.

You can solve the problem using the following techniques:

- Copy Existing Blocks with DATACOPY
- Calculate All Potential Blocks with SET CREATENONMISSINGBLK

Copy Existing Blocks with DATACOPY

You can use the DATACOPY command in Essbase block storage calc scripts to create a block for each existing block, and then perform calculations on the new blocks.

For example:

```
DATACOPY Sales -> Actual TO Sales -> Budget;
DATACOPY Expenses -> Actual TO Expenses -> Budget;
FIX(Budget)
   (Sales = Sales -> Actual * 1.1;
   Expenses = Expenses -> Actual * .95;)
ENDFIX
```

Essbase creates blocks that contain the Budget values for each corresponding Actual block that exists. After the DATACOPY commands are finished, the remaining part of the script changes the values.

Using DATACOPY works well in these situations:

- There is a mathematical relationship between values in existing blocks and their counterparts created by the DATACOPY.

  For example, in the preceding example, the Budget values can be calculated based on the existing Actual values.

  > **Caution:**
  >
  > DATACOPY creates the new blocks with identical values in all cells from the source blocks. If the formula performs only on a portion of the block, these copied cells remain at the end of the calculation, potentially resulting in unwanted values.

- None of the blocks that are copied contain only #MISSING values.

  If #MISSING values exist, blocks are written that contain only #MISSING values. Unneeded #MISSING blocks require Essbase resource and processing time.

See Also

DATACOPY

Calculate All Potential Blocks with SET CREATENONMISSINGBLK

Use the SET CREATENONMISSINGBLK command in Essbase block storage calc scripts to limit unwanted values when calculating on dense or sparse dimensions.

If you are concerned about unwanted values, instead of using DATACOPY, you can use the SET CREATENONMISSINGBLK ON calculation command, which calculates all potential blocks in memory and then stores only the calculated blocks that contain data values. The SET CREATENONMISSINGBLK calculation command can be useful when calculating values on dense or sparse dimensions.

The following example creates budget sales and expense data from existing actual data. Sales and Expenses are members in the dense Measures dimension; Budget and Actual are members in the sparse Scenario dimension.

```
FIX(Budget)
SET CREATENONMISSINGBLK ON
   (Sales = Sales -> Actual * 1.1;
   Expenses = Expenses -> Actual * .95;)
ENDFIX
```

> **Note:**
>
> If SET CREATEBLOCKONEQ ON is set for sparse dimensions, SET CREATENONMISSINGBLK ON temporarily overrides SET CREATEBLOCKONEQ ON until a SET CREATENONMISSINGBLK OFF command is encountered or the calculation script is completed.

The advantage of using the SET CREATENONMISSINGBLK command is that, when applied on dense members, only data cells that are affected by the member formula are saved. The disadvantage is that too many potential blocks may be materialized in memory, possibly affecting calculation performance. When you use this command, it's best to limit the number of potential blocks by using FIX to restrict the scope of the blocks to be calculated.

See Also

SET CREATENONMISSINGBLK

Nonconstant Values Assigned to Members in a Sparse Dimension

FIX…ENDFIX

Use Calculation Scripts on Partitions

A partitioned Essbase application can span multiple servers, processors, or computers. You can achieve significant calculation performance improvements by partitioning applications and running separate calculations on each partition.

Write Calculation Scripts for Partitions

When writing calculation script for transparent or replicated partitions, review these guidelines:

- Evaluate the performance impact on the overall cube (database) calculation. To improve performance, you can:
  - Redesign the overall calculation to avoid referencing remote values that are in a transparent partition in a remote cube.
  - Dynamically calculate a value in a remote cube.

    See Dynamically Calculating Data in Partitions.
  - Replicate a value in the cube that contains the applicable formula.

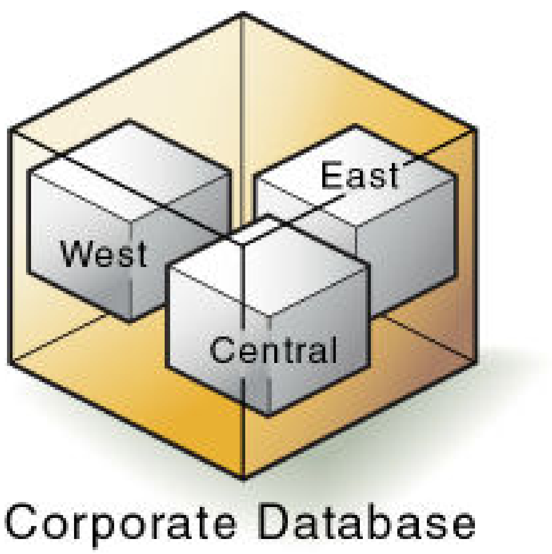
    For example, if replicating quarterly data for the Eastern region, replicate only the values for Qtr1, Qtr2, Qtr3, and Qtr4, and calculate the parent Year values locally.
- Ensure that a referenced value is up-to-date when Essbase retrieves it. Choose one of the options previously discussed (redesign, dynamically calculate, or replicate) or calculate the referenced cube before calculating the formula.

Control Calculation Order for Partitions

You must calculate the partitioned cubes in a specific order to ensure that Essbase calculates the required results.

The following image illustrates partitions in which you view information from the West, Central, and East databases transparently from the Corporate database:

Figure 24-1 Calculating Partitions



West, Central, and East contain only actual values. Corporate contains actual and budgeted values. Although you can view West, Central, and East data in the Corporate database, the data exists only in the West, Central, and East —it is not duplicated in Corporate.

Therefore, when Essbase calculates Corporate, it must take the latest values from West, Central, and East. To obtain the required results, you must calculate West, Central, and East before you calculate Corporate.

Save, Execute, and Copy Calculation Scripts

You can create and save calculation scripts in the Essbase web interface or REST API. You can also migrate or upload calculation scripts, run them as jobs, and reuse them for other cubes (databases).

Save Calculation Scripts

You save an Essbase calculation script as an artifact associated with the cube (database) that you plan to calculate. If a calculation script is intended to be executed on multiple cubes, you must upload the calculation script to each one.

If a calculation script is created in a text file and saved on a client computer, you can paste the script into the editor on the Scripts page in the Essbase web interface and then save the script, or you can upload the calculation script file to Essbase.

See Also

Create Calculation Scripts

Work with Files and Artifacts

Execute Calculation Scripts

There are multiple ways to run Essbase calculation scripts: from Essbase web interface, or the command line (CLI), MaxL, API, or Smart View.

Before you can execute a calculation script, whether you use the Essbase web interface, the REST API Execute Jobs endpoint or any other method, the calculation script must be saved as an artifact for the cube that you plan to calculate. When you run the job type to execute the calculation, the job runs in the background. You can see the status of the calculation. See Execute Calculations if you primarily use the Essbase web interface.

To execute a calculation script in Smart View, see Calculating Data in the *Working with Oracle Smart View for Office*.

You can also use the execute calculation MaxL statement or the CLI calc command to run Essbase calcs.

Copy Calculation Scripts

You can copy calculation scripts to applications and cubes (databases) on any Essbase Server, according to your permissions. You can also copy scripts across servers as part of application migration.

To copy a calculation script, you can also use the **create calculation as** MaxL statement or the **Copy Calc Script** REST API endpoint.

See Also

Uploading Files to a Cube

MaxL execute calculation

CLI calc

REST API Execute Jobs

Check Calculation Results

After you execute an Essbase calculation script, you should check the results of the calculation in Smart View.

Essbase provides the following information about the executed calculation script:

- Calculation order of the dimensions for each pass through the database
- Total calculation time

To display more-detailed information, you can use the SET MSG SUMMARY, SET MSG DETAIL, and SET NOTICE commands in a calculation script. Refer to Specify Global Settings for a Calculation.

You can use these messages to understand how the calculation is performed and to tune it for the next calculation.

Where you view this information depends on the tool used to execute the calculation script.

- Smart View—Application log
- MaxL—Standard output (command-line window)

  The amount of information depends on the message level set in MaxL Shell.

25

Examples of Calculation Scripts for Block Storage Databases

These examples of Essbase calculation scripts can help you develop your own calculation scripts. Examples include budgeting, forecasting, cube subsets, allocation, and looping.

- About These Calculation Script Examples
- Calculate Variance
- Calculate Database Subsets
- Load New Budget Values
- Calculate Product Share and Market Share Values
- Allocate Costs Across Products
- Allocate Values within a Dimension
- Allocate Values Across Multiple Dimensions
- Goal-Seeking Using the LOOP Command
- Forecast Future Values

The information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases.

About These Calculation Script Examples

All examples in this chapter are based on the Sample.Basic database.

For examples that use the Intelligent Calculation commands SET UPDATECALC and SET CLEARUPDATESTATUS, see Reviewing Examples That Use SET CLEARUPDATESTATUS and Reviewing Examples and Solutions for Multiple-Pass Calculations.

Calculate Variance

As an example of calculating variance in an Essbase cube, consider a calculation of the variance percentage between Budget and Actual values. You can calculate it with one or two passes.

The following shows an outline in which Variance and Variance % are tagged as Dynamic Calc, two-pass members.

Figure 25-1 Variance and Variance % in the Scenario Dimension

```
Scenario (Label Only)
    Actual (+)
    Budget (~)
    Variance (~) (Dynamic Calc) (Two Pass Calc) @VAR(Actual, Budget);
    Variance % (~) (Dynamic Calc) (Two Pass Calc) @VARPER(Actual, Budget);
```

During a default calculation, Essbase aggregates the values on the Market and Product dimensions. Because percentage values do not aggregate correctly, the Variance % formula must be recalculated after the default calculation.

Because Variance % is tagged as a Dynamic Calc, two-pass member, Essbase dynamically calculates Variance % values when they are retrieved. The dynamic calculation overwrites the incorrect values with the correctly calculated percentages.

If you choose not to tag Variance % as a Dynamic Calc, two-pass member, use the following calculation script—which assumes that Intelligent Calculation is turned on (the default)—to perform a default calculation and to recalculate the formula on Variance %:

```
CALC ALL;
SET UPDATECALC OFF;
SET CLEARUPDATESTATUS AFTER;
"Variance %";
```

Essbase performs the following actions:

1. Performs a default calculation of the cube (CALC ALL).

   Alternatively, you can run a default calculation without using a calculation script.
2. Turns off Intelligent Calculation (SET UPDATECALC OFF).
3. Marks the calculated blocks calculated by the variance formula of the calculation script as clean, even though the variance calculation is a partial calculation of the cube (SET CLEARUPDATESTATUS AFTER).

   By default, data blocks are marked as clean only after a full calculation.
4. Cycles through the cube calculating the formula for Variance %.

See Also

Interaction of Two-Pass and Intelligent Calculation

Two-Pass Calculation

Calculate Database Subsets

As an example of calculating subsets of an Essbase cube, this calc script uses the @DESCENDENTS function to limit calculation scope.

This example shows how a regional Marketing manager can calculate her respective area of the cube. The calculation script uses @DESCENDANTS(East) to limit the calculations to the East region, as it calculates the Year, Measures, and Product dimensions for each child of East.

The following image shows an outline of the East, West, South, and Central members in the Market dimension:

Figure 25-2 East, West, South, and Central Members in the Market Dimension

```
Market
    East (+) (UDAs: Major Market)
    West (+)
    South (+) (UDAs: Small Market)
    Central (+) (UDAs: Major Market)
```

Example script:

```
/* Calculate the Budget data values for the descendants of East */

FIX(Budget, @DESCENDANTS(East))
   CALC DIM(Year, Measures, Product);
ENDFIX

/* Consolidate East */

FIX(Budget)
   @DESCENDANTS(East);
ENDFIX
```

Essbase performs the following actions:

1. Fixes on the Budget values of the descendants of East.
2. Calculates the Year, Measures, and Product dimensions in one pass for all Budget values of the descendants of East.
3. Fixes on the Budget values for all members on the other dimensions.
4. Aggregates the descendants of East and places the result in East.

Load New Budget Values

This Essbase calculation script example uses FIX and CALC DIM to calculate Budget values, and then recalculates the Variance and Variance % members.

Example script:

```
/* Calculate all Budget values */

FIX(Budget)
   CALC DIM(Year, Product, Market, Measures);
ENDFIX

/* Recalculate the Variance and Variance % formulas, which requires two
passes */

Variance;
"Variance %";
```

Essbase performs the following actions:

1. Fixes on the Budget values.
2. Calculates all Budget values.

   CALC DIM calculates all the dimensions except for the Scenario dimension, which contains Budget.
3. Calculates the formula applied to Variance in the outline.
4. Calculates the formula applied to Variance % in the outline.

Calculate Product Share and Market Share Values

This Essbase calculation script example calculates product share and market share values by calculating each member as a percentage of the total and as a percentage of its parent.

The share values are calculated as follows:

- Each member as a percentage of the total
- Each member as a percentage of its parent

Assume that you added four members to the Measures dimension:

- Market Share
- Product Share
- Market %
- Product %

Example script:

```
/* First consolidate the Sales values to ensure that they are accurate */

FIX(Sales)
   CALC DIM(Year, Market, Product);
ENDFIX

/* Calculate each market as a percentage of the total market for each product
*/

"Market Share" = Sales % Sales -> Market;

/* Calculate each product as a percentage of the total product for each
market */

"Product Share" = Sales % Sales -> Product;

/* Calculate each market as a percentage of its parent for each product */

"Market %" = Sales % @PARENTVAL(Market, Sales);

/* Calculate each product as a percentage its parent for each market */

"Product %" = Sales % @PARENTVAL(Product, Sales);
```

Essbase performs the following actions:

1. Fixes on the Sales values and consolidates all the Sales values.

   The CALC DIM command calculates the Year, Market, and Product dimensions. The Measures dimension contains the Sales member and therefore is not consolidated. The Scenario dimension is label only and therefore does not need to be consolidated.
2. Cycles through the cube and calculates Market Share by taking the Sales value for each product in each market for each month and calculating this Sales value as a percentage of total Sales in all markets for each product (Sales -> Market).

3. Calculates Product Share by taking the Sales value for each product in each market for each month and calculating this Sales value as a percentage of total Sales of all products in each market (Sales -> Product).
4. Calculates Market % by taking the Sales value for each product in each market for each month and calculating this Sales value as a percentage of the Sales value of the parent of the current member on the Market dimension.

   The @PARENTVAL function obtains the Sales value of the parent on the Market dimension.
5. Calculates Product % by taking the Sales value for each product in each market for each month, and calculating this Sales value as a percentage of the Sales value of the parent of the current member on the Product dimension.

   @PARENTVAL obtains the Sales value of the parent on the Product dimension.

Allocate Costs Across Products

This Essbase calculation script example allocates overhead costs to each product in each market for each month.

Overhead costs are allocated based on each product's Sales value as a percentage of the total Sales for all products.

Assume that you added two members to the Measures dimension:

- OH_Costs for the allocated overhead costs
- OH_TotalCost for the total overhead costs

Example script:

```
/* Declare a temporary array called ALLOCQ based on the Year dimension */

ARRAY ALLOCQ[Year];

/* Turn the Aggregate Missing Values setting off. If this is your system
default, omit this line */

SET AGGMISSG OFF;

/* Allocate the overhead costs for Actual values */

FIX(Actual)
   OH_Costs (ALLOCQ=Sales/Sales->Product; OH_Costs =
   OH_TotalCost->Product * ALLOCQ;);

/* Calculate and consolidate the Measures dimension */

   CALC DIM(Measures);
ENDFIX
```

Essbase performs these calculations:

1. Creates a one-dimensional array called ALLOCQ to store the value of Sales as a percentage of total Sales temporarily for each member combination.

   The size of ALLOCQ is based on the number of members in the Year dimension.

2. #MISSING values do not aggregate to their parents (SET AGGMISSG is set to OFF). Data values stored at parent levels are not overwritten.

   If SET AGGMISSG OFF is your system default, omit this line. See #MISSING Values.

   - Fixes on the Actual values.
   - Cycles through the member combinations for Actual, and calculates OH_Costs.
   - Takes the Sales value for each product in each market for each month and calculates it as a percentage of total Sales for all products in each market (Sales -> Product). The result is placed in ALLOCQ.
   - Takes the total overhead costs for all products (OH_TotalCost -> Product) and multiplies it by the value it has just placed in ALLOCQ. The result is placed in OH_Costs.

     Note that both equations are enclosed in parentheses ( ) and associated with the OH_Costs member: OH_Costs (equation1; equation2;).

3. Calculates and consolidates the Measures dimension.

Allocate Values within a Dimension

This Essbase calculation script example uses the @ALLOCATE function to allocate budgeted total expenses across expense categories for two products. The budgeted total expenses are allocated based on the actual values for the previous year.

Assume that you made the following changes, as shown in the outline illustrated below:

- Added a child, Lease, under Total Expenses in the Measures dimension
- Added a child, PY Actual, to the Scenario dimension
- Removed the Dynamic Calc tag from the Total Expenses member

Figure 25-3 Modified Measures and Scenario Dimensions

```
Measures Accounts (Label Only)
    Profit (+) (Dynamic Calc)
        Margin (+) (Dynamic Calc)
        Total Expenses (-) (Expense Reporting)
            Lease (+)
            Marketing (+) (Expense Reporting)
            Payroll (+) (Expense Reporting)
            Misc (+) (Expense Reporting)
    Inventory (~) (Label Only)
    Ratios (~) (Label Only)
Product {Caffeinated, Intro Date, Ounces, Pkg Type }
Market {Population }
Scenario (Label Only)
    Actual (+)
    Budget (~)
    PY Actual (+)
    Variance (~) (Dynamic Calc) (Two Pass Calc) @VAR(Actual, Budget);
    Variance % (~) (Dynamic Calc) (Two Pass Calc) @VARPER(Actual, Budget);
```

Assume that data values of 1000 and 2000 are loaded into Budget -> Total Expenses for Colas and Root Beer, respectively. These values must be allocated to each expense category, evenly spreading the values based on the nonmissing children of Total Expenses from PY Actual. The allocated values must be rounded to the nearest dollar.

Example script:

```
/* Allocate budgeted total expenses based on prior year */

FIX("Total Expenses")
   Budget = @ALLOCATE(Budget->"Total Expenses",
   @CHILDREN("Total Expenses"),"PY Actual",,
   spread,SKIPMISSING,roundAmt,0,errorsToHigh)
ENDFIX
```

The results of the calculation script:

		Budget	PY Actual
Colas	Marketing	334 *	150
	Payroll	#MI	#MI
	Lease	333	200
	Misc	333	100
	Total Expenses	**1000**	**450**
Root Beer	Marketing	500	300
	Payroll	500	200
	Lease	500	200
	Misc	500	400
	Total Expenses	**2000**	**1100**

* Rounding errors are added to this value.

Essbase cycles through the database, performing the following calculations:

1. Fixes on the children of Total Expenses.

   Using a FIX statement with @ALLOCATE may improve calculation performance.

2. For Budget -> Colas -> Marketing, divides 1 by the count of nonmissing values for each expense category in PY Actual -> Colas for each month.

   In this case, 1 is divided by 3, because there are 3 nonmissing expense values for Budget -> Colas.

3. Takes the value from step 2 (.333), multiplies it by the value for Budget -> Colas -> Total Expenses (1000), and rounds to the nearest dollar (333). The result is placed in Budget -> Colas -> Marketing.

4. Repeats steps 2 and 3 for each expense category for Budget -> Colas and then for Budget -> Root Beer.

5. As specified in the calculation script, rounds allocated values to the nearest whole dollar.

   Essbase makes a second pass through the block to make the sum of the rounded values equal to the allocation value (for example, 1000 for Budget -> Colas -> Total Expenses). In this example, there is a rounding error of 1 for Budget -> Colas -> Total Expenses, because the expense categories add up to 999, not 1000, which is the allocation value. Because all allocated values are identical (333), the rounding error of 1 is added to the first value in the allocation range, Budget -> Colas -> Marketing (thus a value of 334).

Allocate Values Across Multiple Dimensions

This Essbase calculation script example uses the @MDALLOCATE function to allocate a loaded value for budgeted total expenses across three dimensions. The budgeted total expenses are allocated based on the actual values of the previous year.

Assume that you made the following changes:

- Added a child, PY Actual, to the Scenario dimension
- Copied data from Actual into PY Actual
- Cleared data from Budget

For this example, a value of 750 (for Budget -> Total Expenses -> Product -> East -> Jan) must be allocated to each expense category for the children of product 100 across the states in the East. The allocation uses values from PY Actual to determine the percentage share that each category should receive.

Example script:

```
/* Allocate budgeted total expenses based on prior year, across 3 dimensions
*/

SET UPDATECALC OFF;
FIX (East, "100", "Total Expenses")
   BUDGET = @MDALLOCATE(750,3,@CHILDREN("100"),@CHILDREN("Total
Expenses"),@CHILDREN(East),"PY Actual",,share);
ENDFIX
```

The values for PY Actual:

```
         Jan
         PY Actual
                         Marketing    Payroll   Misc   Total Expenses
100-10   New York        94           51        0      145
         Massachusetts   23           31        1      55
         Florida         27           31        0      58
         Connecticut     40           31        0      71
         New Hampshire   15           31        1      47
100-20   New York        199          175       2      376
         Massachusetts   #MI          #MI       #MI    #MI
         Florida         #MI          #MI       #MI    #MI
         Connecticut     26           23        0      49
         New Hampshire   #MI          #MI       #MI    #MI
100-30   New York        #MI          #MI       #MI    #MI
         Massachusetts   26           23        0      49
         Florida         #MI          #MI       #MI    #MI
         Connecticut     #MI          #MI       #MI    #MI
         New Hampshire   #MI          #MI       #MI    #MI
100      New York        #MI          #MI       #MI    #MI
         Massachusetts   12           22        1      35
         Florida         12           22        1      35
         Connecticut     94           51        0      145
```

```
                 New Hampshire    23            31         1       55
                 East             237           220        3       460
```

Essbase cycles through the database, performing these calculations:

1. Fixes on East, the children of 100, and Total Expenses.

   Using a FIX statement with @MDALLOCATE may improve calculation performance.

2. Before performing the allocation, determines what share of 750 (the value to be allocated) each expense category should receive, for each product-state combination, using the shares of each expense category from PY Actual. Starting with PY Actual -> 100-10 -> New York, Essbase divides the value for the first expense category, Marketing, by the value for PY Actual-> 100-10 -> East -> Total Expenses to calculate the percentage share of that category.

   For example, Essbase divides the value for PY Actual -> 100-10 -> New York -> Marketing (94) by the value for PY Actual -> 100-10 -> East -> Total Expenses (460), which yields a percentage share of approximately 20.4% for the Marketing category.

3. Repeats step 2 for each expense category, for each product-state combination.

4. During the allocation, Essbase uses the percentage shares calculated in step 2 and step 3 to determine what share of 750 should be allocated to each child of Total Expenses from Budget, for each product-state combination.

   For example, for Marketing, Essbase uses the 20.4% figure calculated in step 2, takes 20.4% of 750 (approximately 153), and places the allocated value in Budget -> 100-10 -> New York -> Marketing (see the results that follow this procedure).

5. Repeats step 4 for each expense category and for each product-state combination, using the percentage shares from PY Actual calculated in step 2 and step 3.

6. Consolidates the expense categories to yield the values for Total Expenses.

The results of the allocation for Budget:

```
         Jan
         Budget
                          Marketing     Payroll    Misc    Total Expenses
100-10   New York         153.26        83.15      0       236.41
         Massachusetts    37.50         50.54      1.63    89.67
         Florida          44.02         50.54      0       94.56
         Connecticut      65.22         50.54      0       115.76
         New Hampshire    24.26         50.54      1.63    76.63
100-20   New York         #MI           #MI        #MI     #MI
         Massachusetts    #MI           #MI        #MI     #MI
         Florida          42.39         37.50      0       79.89
         Connecticut      #MI           #MI        #MI     #MI
         New Hampshire    #MI           #MI        #MI     #MI
100-30   New York         #MI           #MI        #MI     #MI
         Massachusetts    #MI           #MI        #MI     #MI
         Florida          #MI           #MI        #MI     #MI
         Connecticut      #MI           #MI        #MI     #MI
         New Hampshire    19.57         35.87      1.63    57.07
100      New York         153.26        83.15      0       236.41
         Massachusetts    37.50         50.54      1.63    89.67
         Florida          86.41         88.04      0       174.46
         Connecticut      65.22         50.54      0       115.76
```

```
New Hampshire   44.02        86.41     3.26    133.70
East            386.41       358.70    4.89    750
```

Goal-Seeking Using the LOOP Command

This Essbase calculation script example shows how to calculate the sales value you must reach to obtain a certain profit on a specific product. In this case, the calculation script adjusts the Budget value of Sales to reach a goal of 15,000 Profit for Jan.

As shown in the outline below, assume that no members are tagged as Dynamic Calc, and that the Profit per Ounce member (under Ratios in the Measures dimension) is not included in the calculation.

Figure 25-4   Measures Dimension

```
Measures Accounts (Label Only)
    Profit (+)
        Margin (+)
            Sales (+)
            COGS (-) (Expense Reporting)
        Total Expenses (-) (Expense Reporting)
            Marketing (+) (Expense Reporting)
            Payroll (+) (Expense Reporting)
            Misc (+) (Expense Reporting)
    Inventory (~) (Label Only)
    Ratios (~) (Label Only)
        Margin % (+) (Two Pass Calc) Margin % Sales;
        Profit % (~) (Two Pass Calc) Profit % Sales;
```

Assume that, before running the goal-seeking calculation script, the data values are:

```
Product, Market, Budget   Jan
Profit                    12,278.50
   Margin                 30,195.50
      Sales               49,950.00
      COGS                19,755.00
   Total Expenses         17,917.00
      Marketing           3,515.00
      Payroll             14,402.00
      Misc                0
Inventory                 Label Only member
Ratios                    Label Only member
   Margin %               60.45
   Profit %               24.58
```

Example script:

```
/* Declare the temporary variables and set their initial values*/

VAR
   Target = 15000,
   AcceptableErrorPercent = .001,
   AcceptableError,
```

```
   PriorVar,
   PriorTar,
   PctNewVarChange = .10,
   CurTarDiff,
   Slope,
   Quit = 0,
      DependencyCheck,
      NxtVar;

/*Declare a temporary array variable called Rollback based on the Measures
dimension */

ARRAY Rollback [Measures];

/* Fix on the appropriate member combinations and perform the goal-seeking
calculation*/

FIX(Budget, Jan, Product, Market)
   LOOP (35, Quit)
      Sales (Rollback = Budget;
      AcceptableError = Target * (AcceptableErrorPercent);
      PriorVar = Sales;
      PriorTar = Profit;
      Sales = Sales + PctNewVarChange * Sales;);
      CALC DIM(Measures);
      Sales (DependencyCheck = PriorVar - PriorTar;
      IF(DependencyCheck <> 0) CurTarDiff = Profit - Target;
         IF(@ABS(CurTarDiff) > @ABS(AcceptableError))
            Slope = (Profit - PriorTar) / (Sales - PriorVar);
            NxtVar = Sales - (CurTarDiff / Slope);
            PctNewVarChange = (NxtVar - Sales) / Sales;
         ELSE
            Quit = 1;
         ENDIF;
      ELSE
         Budget = Rollback;
         Quit = 1;
      ENDIF;);
   ENDLOOP
   CALC DIM(Measures);
ENDFIX
```

Essbase performs the following calculations:

1. Declares the required temporary variables using the VAR command. Where appropriate, the initial values are set.
2. Declares a one-dimensional array called Rollback to store the Budget values.

   The size of Rollback is based on the number of members in the Measures dimension.
3. Fixes on the Jan -> Budget values for all Product and Market members.
4. Ensures that the commands between LOOP and ENDLOOP are cycled through 35 times *for each member combination*. If, however, the Quit variable is set to 1, the LOOP is broken and the calculation continues after the ENDLOOP command.
5. Cycles through the member combinations, performing the following calculations:

   a. Places the Budget -> Sales value in the Rollback temporary array variable.
   b. Calculates the acceptable error, by multiplying the Target value (15000) by the AcceptableErrorPercent value (0.001). The result is placed in the AcceptableError variable.
   c. Retains the current Sales value, and places the Sales value for the current member combination in the PriorVar temporary variable.
   d. Retains the current Profit value, and places the Profit value for the current member combination in the PriorTar temporary variable.
   e. Calculates a new Sales value by multiplying the PctNewVarChange value (0.1) by the current Sales value, and adding the current Sales value. The result is placed in Sales.
   f. Calculates and consolidates the Measures dimension.
   g. Subtracts the PriorTar value from the PriorVar value, and places the result in the DependencyCheck temporary variable.
   h. Checks that DependencyCheck is not 0 (zero) (IF).
      - If DependencyCheck is not 0, subtracts the Target value (15000) from the current Profit and places the result in the CurTarDiff temporary variable.

        The IF command checks whether the absolute value (irrespective of the + or – sign) of CurTarDiff is greater than the absolute value of AcceptableError:
        – If greater than AcceptableError, calculates the Slope, NxtVar, and PctNewVarChange temporary variables.
        – If not greater than AcceptableError, breaks the LOOP command by setting the value of Quit to 1. The calculation continues after the ENDLOOP command.
      - If DependencyCheck is 0, places the value in the Rollback array into Budget. Essbase breaks the LOOP command by setting the value of Quit to 1. The calculation continues after the ENDLOOP command.

6. Calculates and consolidates the Measures dimension.

The results for product 100-10:

```
Product, Market, Budget    Jan
Profit                     15,000.00
   Margin                  32,917.00
      Sales                52,671.50
      COGS                 19,755.00
   Total Expenses          17,917.00
      Marketing            3,515.00
      Payroll              14,402.00
      Misc                 0
Inventory                  Label Only member
Ratios                     Label Only member
   Margin %                28.47839913
   Profit %                62.49489762
```

Forecast Future Values

This Essbase calculation script example uses a linear regression forecasting method to produce a trend, or line, that starts with the known data values from selected previous months, and continues with forecasted values based on the known values.

The trend is calculated using @TREND. This example also shows how to check the results of the trend for "goodness of fit" to the known data values. In this case, the calculation script forecasts sales data for June–December, assuming that data currently exists only up to May.

Assume that the Measures dimension contains an additional child, ErrorLR, where the goodness-of-fit results are placed.

Example script:

```
Sales
(@TREND(@LIST(Jan,Mar,Apr),@LIST(1,3,4),,
  @RANGE(ErrorLR,@LIST(Jan,Mar,Apr)),
    @LIST(6,7,8,9,10,11,12),
      Jun:Dec,LR););
```

The following table describes the parameters used in the forecasting calculation script:

Table 25-1 Parameters Used in the Example Calculation Script for Forecasting Future Values

Parameter	Description
@LIST(Jan,Mar,Apr)	Represents the *Ylist*, or the members that contain the known data values. The @LIST function groups the three members as a comma-delimited list and keeps the list separate from other parameters.
@LIST(1,3,4)	Represents the *Xlist*, or the underlying variable values. Because Feb and May are skipped, Essbase numbers the *Ylist* values as 1,3,4.
,	The extra comma after the *Xlist* parameter indicates that a parameter (*weightList*) was skipped. This example uses a default weight of 1.
@RANGE(ErrorLR,@LIST(Jan,Mar,Apr)	Represents the *errorList*, or the member list where results of the goodness of fit of the trend line to *Ylist* are placed. The values placed in *errorList* are the differences between the data points in *Ylist* and the data points on the trend line that is produced. The @RANGE function combines the ErrorLR member with *Ylist* (Jan, Mar, Apr) to produce a member list.
@LIST(6,7,8,9,10,11,12)	Represents the *XforecastList*, or the underlying variable values for which the forecast is sought. This example forecasts values consecutively for Jun–Dec, so the values are 6,7,8,9,10,11,12.

Table 25-1 (Cont.) Parameters Used in the Example Calculation Script for Forecasting Future Values

Parameter	Description
Jun:Dec	Represents the *YforecastList*, or the member list into which the forecast values are placed. This example forecasts values for Jun–Dec, based on the values for Jan, Mar, and Apr.
LR	Specifies the Linear Regression method.

Essbase cycles through the database, performing the following calculations:

1. Finds the known data values on which to base the trend (Sales for Jan, Mar, Apr), as specified by the *Ylist* and *Xlist* parameters.
2. Calculates the trend line using Linear Regression and places the results in Sales for Jun–Dec, as specified by the *YforecastList* parameter.
3. Calculates the goodness of fit of the trend line for the data values for Jan, Mar, and Apr, and places the results in ErrorLR for those months.

   For example, the value in ErrorLR for Jan (4.57) means that after Essbase calculates the trend line, the difference between the Sales value for Jan (2339) and the Jan value on the trend line is 4.57. The ErrorLR values for Feb and May are #MISSING, because these months were not part of *Ylist*.

The results of the calculation script:

```
      100 West Actual
      Sales      ErrorLR
Jan   2339       4.57
Feb   2298       #MI
Mar   2313       -13.71
Apr   2332       9.14
May   2351       #MI
Jun   2315.14    #MI
Jul   2311.29    #MI
Aug   2307.49    #MI
Sep   2303.57    #MI
Oct   2299.71    #MI
Nov   2295.86    #MI
Dec   2292       #MI
```

26 Using Parallel Calculation

Essbase parallel calculation lets you streamline calculations by running them as concurrent, independent tasks.

With serial calculation, each calculation pass is scheduled to run on a single processor. If invoked from a calculation script, the calculations are executed sequentially in the order in which they appear in the calculation script. Parallel calculation breaks each calculation pass into sub-tasks. The sub-tasks that can run independently of one another are scheduled to run simultaneously on up 128 threads. Each thread may be on a different processor.

- About Parallel Calculation
- Using FIXPARALLEL Parallel Calculation
- Using CALCPARALLEL Parallel Calculation

About Parallel Calculation

Essbase provides two ways of invoking a calculation on a block storage database:

- The calculation may be implicitly specified by the outline itself.
- The calculation may be explicitly specified by a calculation script that you create. The script contains formulas and calculation instructions.

Regardless of how a calculation is triggered, Essbase can execute the calculation in one of two modes:

- *Serial calculation* is the default. With serial calculation, each calculation pass is scheduled to run on a single processor. If invoked from a calculation script, the calculations are executed sequentially in the order in which they appear in the calculation script.
- *Parallel calculation* breaks each calculation pass into sub-tasks. The sub-tasks that can run independently of one another are scheduled to run simultaneously on up 128 threads for block storage and aggregate storage databases running on 64-bit operating systems. Each thread may be on a different processor.

Parallel calculation operations do not dynamically create threads, but instead use a set number of threads from a pre-created pool. You can customize the size of the thread pool using the WORKERTHREADS configuration setting.

Using FIXPARALLEL Parallel Calculation

Although parallel calculation can still be performed using the CALCPARALLEL method, in many cases it might be more beneficial to use the newer FIXPARALLEL command block method.

The Essbase FIXPARALLEL method of parallel calculation lets you select a block of calculation script commands and divide their tasks into parallel threads.

In a FIXPARALLEL command block, you input some commands to be executed, along with a number of threads (*numThreads*) and a member list (*mbrList*) specifying the database regions (slices) to be calculated. Essbase creates a list of tasks from the combinations in the member list, and divides the tasks across the threads.

The FIXPARALLEL method can be advantageous in the following cases:

- If you need to use temporary variables during parallel calculation.
- If you need to use the DATACOPY, DATAEXPORT, or CLEARBLOCK calculation commands.
- In conjunction with the @XREF or @XWRITE functions.
- If you need to export regions of the database in parallel.
- In cases where CALCPARALLEL is not meeting performance requirements, and your outline generates many empty tasks, or contains many task groupings with fewer tasks than threads made available to the calculation.

Using CALCPARALLEL Parallel Calculation

To change from the default serial calculation to CALCPARALLEL parallel calculation, you change one or two configuration settings and restart the server, or add an instruction to the calculation script.

See Enabling CALCPARALLEL Parallel Calculation.

The following topics discuss the details of parallel calculation.

Analysis of Feasibility of CALPARALLEL

Essbase evaluates whether using CALPARALLEL method of parallel calculation is possible before each calculation pass for which you have enabled it.

Essbase analyzes the outline and the calculation requested for each calculation pass. Remember that one calculation may require multiple passes. Some situations may create the need for multiple passes, including dynamic calculation, the presence of a member tagged as two-pass, or calculations that create certain kinds of interdependencies. See Calculation Passes.

If Essbase determines that parallel calculation using the CALPARALLEL method is possible, Essbase splits the calculation into smaller tasks that are independent of each other. During the calculation, Essbase performs the smaller tasks simultaneously.

However, Essbase resorts to serial calculation if there are complex interdependencies between formulas that participate in the pass. Such interdependencies can render parallel calculation impossible.

Consider whether the FIXPARALLEL method might be more suited to your use case. See Using FIXPARALLEL Parallel Calculation.

CALCPARALLEL Parallel Calculation Guidelines

Outline structure and application design determine whether enabling parallel calculation can improve calculation performance. Before you enable CALCPARALLEL parallel calculation, review the following guidelines, which will help you get the full benefit of parallel calculation:

- One or more formulas present in a calculation may prevent Essbase from using CALCPARALLEL parallel calculation even if it is enabled. For a description of formulas that may force serial calculation regardless of parallel calculation settings, see Formula Limitations.

- Calculation tasks are generated along the last *n* sparse dimensions of an outline. These sparse dimensions used to identify tasks are called task dimensions. The number of task dimensions, *n*, is either selected dynamically by Essbase, or you can override the number by specifying a value for the calculation command SET CALCTASKDIMS (or application configuration setting CALCTASKDIMS).

  Order the sparse dimensions in an outline from smallest to largest, based on actual size of the dimension (as reported by the MaxL statement **query database DBS-NAME get dbstats dimension**). This ordering recommendation is consistent with recommendations for optimizing calculator cache size and consistent with other outline recommendations. For a description of situations that may need to use additional dimensions (more than the last sparse dimension) and for instructions on how to increase the number of sparse dimensions used, see Identifying Additional Tasks for Parallel Calculation.
- CALCPARALLEL parallel calculation is effective on non-partitioned applications and these partitioned applications:
  - Replicated partitions
  - Transparent partitions if the calculation occurs at the target database. The number of sparse dimensions specified by SET CALCTASKDIMS in a calculation script must be set at 1. For information on limitations imposed by the use of parallel calculation with transparent partitions, see Transparent Partition Limitations; for information on using SET CALCTASKDIMS, see Identifying Additional Tasks for Parallel Calculation.
- Update transactions, such as calculations and data updates, are more resource-consuming requests than MDX queries or report scripts.

Relationship Between CALCPARALLEL Parallel Calculation and Other Essbase Features

The following topics discuss the relationship between CALCPARALLEL parallel calculation and other Essbase functionality.

- Retrieval Performance
- Formula Limitations
- Calculator Cache
- Transparent Partition Limitations

Retrieval Performance

Placing the largest sparse dimension at the end of the outline for maximum parallel calculation performance may slow retrieval performance. See Optimizing Query Performance.

Formula Limitations

The presence of some formulas may force serial calculation. The following formula placements likely will force serial calculation:

- A formula on a dense member, including all stored members and any Dynamic Calc members upon which a stored member may be dependent, that causes a dependence on a member of the dimension that is used to identify tasks for parallel calculation.

- A formula that contains references to variables declared in a calculation script that uses @VAR, @ARRAY, @XREF, or @XWRITE. Consider using FIXPARALLEL.
- A sparse dimension member formula using @XREF, and the dimension for the sparse member is fully calculated. @XREF does not force serial calculation when it is on dense Dynamic Calc members that are not dependent on other stored members during the batch calculation.
- A member formula that causes a circular dependence. For example, member A has a formula referring to member B, and member B has a formula referring to member C, and member C has a formula referring to member A.
- A formula on a dense or sparse member with a dependency on a member or members from the dimension used to identify tasks for parallel processing.
- A sparse dimension member formula that contains references to members from other sparse dimensions.

If you need to use a formula that might prevent parallel calculation, consider using FIXPARALLEL. Otherwise, you can either mark the member of the formula as Dynamic Calc, or exclude the formula from the scope of the calculation. To see whether a formula is preventing parallel calculation, check the application log. For relevant error messages, see Monitoring CALCPARALLEL Parallel Calculation.

Calculator Cache

At the start of a calculation pass, Essbase checks the calculator cache size and the degree of parallelism and then uses the calculator cache bitmap option appropriate for maximum performance. Therefore, the bitmap option used for parallel calculation may be different from that used for serial calculation.

For example, assume Essbase performs a serial calculation and uses multiple bitmaps and a single anchoring dimension. Without explicit change of the calculator cache size, Essbase might perform a parallel calculation using only a single bitmap and a single anchoring dimension.

You can determine the calculator cache mode that controls the bitmap options by checking the application log at the start of each calculation pass for an entry similar to the following:

```
Multiple bitmap mode calculator cache memory usage has a limit of [50000]
bitmaps.
```

Transparent Partition Limitations

Parallel calculation with transparent partitions has the following limitations:

- You cannot use parallel calculation across transparent partitions unless the calculation occurs at the target.
- You must set the task dimensions to 1. To do this, use SET CALCTASKDIMS calculation command or CALCTASKDIMS configuration setting.
- You must increase the calculator cache so that multiple bitmaps can be used. You can identify the calculator cache mode that controls the bitmap options by checking the application log at the start of each calculation pass for an entry similar to the following:

  ```
  Multiple bitmap mode calculator cache memory usage has a limit of [50000]
  bitmaps.
  ```

Checking Current CALCPARALLEL Settings

You can check either the application configuration or the calculation script that you plan to use to see if parallel calculation is enabled.

To check whether parallel calculation has been enabled at the application level, search for the parameter CALCPARALLEL, and check its specified value.

The number of threads that can simultaneously perform tasks to complete a calculation is specified by a value between 1 and 128. Block storage and aggregate storage databases support up to 128 threads.

To check whether a calculation script sets parallel calculation, look for the SET CALCPARALLEL command. The number of threads that can simultaneously perform tasks to complete a calculation is specified by a value between 1 and 128. Block storage and aggregate storage databases running on 64-bit platforms support up to 128 threads. Review the script carefully, because the script may enable or disable parallel calculation more than once. Alternately, a script can use FIXPARALLEL command blocks for parallel calculation.

Enabling CALCPARALLEL Parallel Calculation

To use CALCPARALLEL parallel calculation, use either of these methods:

- Add or edit the application configuration settings CALCPARALLEL and CALCTASKDIMS.
- Use SET CALCPARALLEL and SET CALCTASKDIMS calculation commands in a calculation script.

To enable parallel calculation:

1. If you plan to enable parallel calculation in the application configuration, check the current status to see whether an entry exists. Use the process described in Checking Current CALCPARALLEL Settings.
2. Add SET CALCPARALLEL to a calculation script.
3. If needed, enable Essbase to use more than the one sparse dimension to identify tasks for parallel calculation.

   Use the process described in Identifying Additional Tasks for Parallel Calculation and Tuning CALCPARALLEL with Log Messages.
4. Run the calculation.

> **Tip:**
>
> You can combine the use of CALCPARALLEL and SET CALCPARALLEL if the site requires it. For example, you can set CALCPARALLEL as off at the server level, and use a calculation script to enable and disable parallel calculation as needed.

Identifying Additional Tasks for Parallel Calculation

By default, Essbase uses an iterative technique to select the optimal number of task dimensions to use for CALCPARALLEL parallel calculation.

If necessary, you can enable Essbase to use a specific number, *n*, of task dimensions. For example, if you have a FIX statement on a member of the last sparse dimension, you can include the next-to-last sparse dimension from the outline as well. Because each unique member combination of these two dimensions is identified as a potential task, more and smaller tasks are created, increasing the opportunities for parallel processing and improving load balancing.

To specify the number of task dimensions for parallel calculation:

1. If you are not sure, verify whether parallel calculation is enabled, using the process described in Checking Current CALCPARALLEL Settings. Without SET CALCPARALLEL enabled (or SET CALCPARALLEL used in a calculation script), CALCTASKDIMS has no effect.
2. **Optional:** Essbase selects a default number, *n*, of task dimensions to use for parallel calculation and this number is printed in the application log file as an informational message; for example: `Parallelizing using [2] task dimensions`. To override the default *n* setting, add or modify CALCTASKDIMS configuration setting for the application, or use the calculation script command SET CALCTASKDIMS.
3. Run the calculation script.

> **Note:**
>
> In some cases, Essbase uses fewer dimensions to identify tasks than is specified by CALCTASKDIMS or SET CALCTASKDIMS.

Tuning CALCPARALLEL with Log Messages

If you are using CALCPARALLEL parallel calculation, you may encounter the following log messages:

```
Current selection of task dimensions [n] will generate insufficient number of
tasks [n] for parallel calculation. See whether calculation time can be improved
by increasing the number of task dimensions by one (see SET CALCTASKDIMS topic in
the documentation). Also, consider using FIXPARALLEL to make custom task
selections that are different from CALCPARALLEL.
```

If this message is encountered, it means that during a parallel calculation, Essbase refrained from increasing the number of task dimensions, in case that would have resulted in tasks becoming too small. When tasks become too small, calculation scheduling overhead could overtake the benefits of parallelism. However, when tasks are too large, there might not be enough tasks for parallel calculation threads to work on.

If the next potential task dimension is not the first sparse dimension, consider increasing the number of task dimensions by one, using the SET CALCTASKDIMS calc command (or the CALCTASKDIMS configuration setting), and observe whether that improves the speed of the calculation. Also, consider using FIXPARALLEL to make custom task selections that are different from CALCPARALLEL (see Using FIXPARALLEL Parallel Calculation).

```
Current number of task dimensions [n] for parallel calculation might have caused
too many tasks [n] to be generated. See whether calculation time can be improved
by decreasing the number of task dimensions by one (see SET CALCTASKDIMS topic in
the documentation). Also, consider using FIXPARALLEL to make custom task
selections that are different from CALCPARALLEL.
```

For parallel calculation, having a sufficient number of tasks helps to reduce the effects of data skew. However, too many tasks (even for appropriately sized tasks) can cause the scheduling overhead to outweigh the benefits. Essbase targets an optimal range. If you see the above message, it means that Essbase tried to meet the recommended minimum number of tasks by adding one more task dimension; in doing so, it is possible that the upper boundary for task count may have been crossed.

If the last task dimension selected by Essbase is not the only task dimension, consider decreasing task dimensions by one, using the SET CALCTASKDIMS calc command (or the CALCTASKDIMS configuration setting), and observe whether that improves the speed of the calculation. Also, consider using FIXPARALLEL to make custom task selections that are different from CALCPARALLEL.

Monitoring CALCPARALLEL Parallel Calculation

You can view events related to parallel calculation in the application log.

For each calculation pass, Essbase writes several types of information to the application log to support parallel calculation:

- If you have enabled parallel calculation and Essbase has determined that parallel calculation can be performed, Essbase writes a message in the application log:

  ```
  Calculating in parallel with n threads
  ```

  n represents the number of concurrent tasks specified in CALCPARALLEL or SET CALCPARALLEL.
- For each formula that prevents parallel calculation (forces serial calculation), Essbase writes a message to the application log:

  ```
  Formula on (or backward dependence from) mbr memberName prevents
  calculation from running in parallel.
  ```

  memberName represents the name of the member where the relevant formula exists. You can look in the application log for such messages and consider removing the formula or, if possible, tagging the relevant member or members as Dynamic Calc so they do not feature in the calculation pass.
- Essbase writes a message to the application log specifying the number of tasks that can be executed concurrently (based on the data, not the value of CALCPARALLEL or SET CALCPARALLEL):

  ```
  Calculation task schedule [576,35,14,3,2,1]
  ```

  The example message indicates that 576 tasks can be executed concurrently. After the 576 tasks complete, 35 more can be performed concurrently, and so on.

  The benefit of parallel calculation is greatest in the first few steps and diminishes as fewer concurrent tasks are performed.

  The degree of parallelism depends on the number of tasks in the task schedule. The greater the number, the more tasks that can run in parallel, and the greater the performance gains.

- Essbase writes a message to the application log file indicating how many tasks are empty (contain no calculations):

```
[Tue Jun 27 12:30:44 2007]Local/CCDemo/Finance/essexer/
Info(1012681) Empty tasks [291,1,0,0,0,0]
```

In the example, Essbase indicates that 291 of the tasks at level 0 were empty.

If the ratio of empty tasks to the tasks specified in the calculation task schedule is greater than 50% (for example, 291 / 576), parallelism may not be giving you improved performance because of the high sparsity in the data model.

You can change dense-sparse assignments to reduce the number of empty tasks and increase the performance gains from parallel calculation.

27 Developing Custom-Defined Calculation Functions

To enhance the calculation functions available for Essbase block storage cubes, you can use Java to develop your own custom-defined functions (CDFs). After you write the functions, install the Java class and then register the functions, either globally with the Essbase Server, or locally to an application.

You can use your custom-defined functions in Essbase calculation scripts.

Essbase does not provide tools for creating Java classes and archives; you must have a supported version of the JDK.

For examples of custom-defined functions, see Java Code Examples.

Custom defined functions are available only for block storage cubes (not relevant to aggregate storage cubes).

To create a custom-defined function, use the following workflow.

1. Review the requirements for custom-defined functions: Requirements for Validity of Custom-Defined Functions
2. Write a public Java class that contains at least one public, static method to be used as a custom-defined function: Creating and Compiling a Java Class for Custom Defined Functions
3. Install the Java class: Installing Java Classes on Essbase Server
4. Register the custom-defined function as a local or global function: Registering Custom-Defined Functions

Requirements for Validity of Custom-Defined Functions

You design your Essbase custom-defined functions as methods in a Java class. For global functions, write methods in one class. For application functions, use separate classes and jar files per application. Test functions locally to one application before registering them globally. Note the supported data types, variables, and naming conventions.

You can create multiple methods in a class for use as a custom-defined function. Typically, Oracle recommends that you create the methods that you plan to use across all applications on an Essbase Server as custom-defined functions in a single class. If, however, you plan to add custom-defined functions that will be used in selective applications on the Essbase Server, create these custom-defined functions in a separate class and add them to Essbase Server in a separate `.jar` file.

When creating multiple Java classes that contain methods for use as custom-defined functions, verify that each class name is unique. Duplicate class names cause methods in the duplicate class not to be recognized, and you cannot register those methods as custom-defined functions.

Using test programs in Java, test the Java classes and methods. When you are satisfied with the output of the methods, install them on Essbase Server and register them in a single test application. Do not register functions globally for testing; doing so makes updating them more difficult if you encounter problems.

Methods in custom-defined functions can have any combination of the following supported data types as input parameters:

- boolean
- byte
- char
- com.hyperion.essbase.calculator.CalcBoolean
- float, double
- java.lang.String
- short, int, long
- arrays of any of these types

CalcBoolean is an Essbase-specific data type that can include three values—TRUE, FALSE, and #MISSING. For information about the other listed data types, see the JDK documentation.

The method return data type can be void or any of the preceding data types. Returned data types are converted to Essbase-specific data types. Strings are mapped to a string type. Boolean values are mapped to the CalcBoolean data type. All other values are mapped to a double type.

> **Note:**
>
> Essbase does not support double variables returned with infinite or Not-a-Number values. If these values are returned from a Java program, they may not be recorded or displayed correctly in Essbase. Double variables should be checked for infinite or Not-a-Number values and set to finite values before being returned to Essbase. See the entry for the class Double in the JDK documentation.

For creating, deleting, and managing custom-defined functions, Essbase requires these security permissions:

- Local, application-wide, custom-defined functions: Application Manager or higher
- Global, server-wide, custom-defined functions: System Administrator

When you register a custom-defined function in Essbase, you give the function a name, which is used in calculation scripts and formulas and is distinct from the Java class and method name used by the function.

Follow these requirements for naming custom-defined functions:

- Start the name with the @ symbol. The rest of a function name can contain letters, numbers, and the following symbols: @, #, $, and _. Function names cannot contain spaces.

  For example: @MYFUNCTION
- Start the names of custom-defined functions that are called only by custom-defined macros with “@_”, to distinguish them from general-use functions and macros.

  For example: @_MYFUNCTION

- Custom-defined functions must have unique names. Function names must be different from each other, from the names of custom-defined macros, and from the names of existing calculation functions.
- If an Essbase application contains a local function that has the same name as a global function, the local function is used for calculation.

Creating and Compiling a Java Class for Custom Defined Functions

To create and compile a Java class for Essbase custom defined functions, write the class using a text editor or IDE (integrated development environment), and compile it using the `javac` tool.

Here is a sample workflow for creating a Java class for a custom-defined function:

1. In a text editor, create a Java class.

   For example:

   ```
   public class CalcFunc {
     public static double sum (double[] data) {
       int i, n = data.length;
       double sum = 0.0d;
       for (i=0; i<n; i++) {
         double d = data [i];
         sum = sum + d;
       }
       return sum;
     }
   }
   ```

2. Save the file with a `.java` extension.

   For example:

   ```
   CalcFunc.java
   ```

3. Navigate to the directory where the `.java` file resides; at a command prompt, enter this command:

   ```
   javac java_filename
   ```

   For example:

   ```
   javac CalcFunc.java
   ```

4. Resolve any compiling errors until the compiler creates a new file with a `.class` extension.

   For example:

   ```
   CalcFunc.class
   ```

Installing Java Classes on Essbase Server

To install the Java classes for your custom defined calculation functions (CDFs) onto the Essbase Server, compile them, copy the jar file to a global or application level `udf` directory as specified in these instructions, and restart the application or server.

Java classes must be compiled in a JAR file, using the JDK `jar` tool.

To create a `.jar` file and install it on an Essbase Server:

1. Navigate to the directory where the `.class` file resides; at a command prompt, enter this command:

   ```
   jar cf jar_filename class_filename
   ```

   For example:

   ```
   jar cf CalcFunc.jar CalcFunc.class
   ```

2. On the computer running Essbase Server, copy the `.jar` file to one of the following directories (if the directory does not exist, create it):
   - For `.jar` files containing global custom-defined functions:

     ESSBASEPATH/java/udf/
   - For `.jar` files to be used only with specific applications:

     <Application Directory>/app/*appname*/udf/
     where *appname* is the name of the application where the local custom-defined function will be used.

   If you do not know the location of *ESSBASEPATH* or <*Application Directory*>, see Environment Variables in the Essbase Platform.

   If the `.jar` file is subsequently placed in another location, you must modify the *CLASSPATH* variable to include the full path and filename for the `.jar` file.

3. If the functions will be used only by specific applications, restart those applications. Otherwise, restart Essbase Server. See Start, Stop, and Check Servers for independent deployments, or Use Commands to Start, Stop, and View Status of Processes for stack deployment on OCI.

Registering Custom-Defined Functions

Use MaxL to register your custom defined functions with Essbase. The registration task comes after you have written the CDFs within Java classes, compiled the classes, and installed the jar files.

After you have compiled the Java classes for custom-defined functions into `.jar` files and installed the `.jar` files on Essbase Server, you must register the custom-defined functions before you can use them in calculation scripts and formulas. See Requirements for Validity of Custom-Defined Functions.

When you register a global custom-defined function, all Essbase applications on the Essbase Server can use it. Test custom-defined functions in a single application (and register them only in that application) before making them global functions.

Use the same process for updating the catalog of functions as for updating the catalog of macros. See Refreshing the Catalog of Custom-Defined Macros.

> **Caution:**
>
> Do not register global functions for testing; doing so makes changing them more difficult if you encounter problems.

To register a custom-defined function, use the create function MaxL statement.

To register a custom-defined function with local scope, include the application name as a prefix. For example, the following MaxL statement registers the custom-defined function, @JSUM, in the CalcFunc class as a local function for use within the Sample application:

```
create function Sample.'@JSUM'
as 'CalcFunc.sum'
spec '@JSUM(memberRange)'
comment 'adds list of input members';
```

To register a custom-defined function with global scope, do not include the application name as a prefix. For example, the following MaxL statement registers the custom-defined function, @JSUM, in the CalcFunc class as a global function for use in any application on Essbase Server:

```
create function '@JSUM'
as 'CalcFunc.sum'
spec '@JSUM(memberRange)'
comment 'adds list of input members';
```

> **Note:**
>
> Specifying input parameters for the Java method is optional. If you do not specify input parameters, Essbase reads them from the method definition in the Java code. If, however, you are registering multiple custom-defined functions with the same method name but with different parameter sets, you must register each version of the function separately, specifying the parameters for each version of the function.

Using Registered Custom-Defined Functions

You can use registered custom-defined functions in calculation scripts and formulas, the same way you use native Essbase calculation functions.

To use a registered custom-defined function:

1. Create or open an existing calculation script or formula.
   - If the custom-defined function was registered locally—within a specific application—you must use a calculation script or formula within that application.
   - If the custom-defined function was registered globally, you can use any calculation script or formula on Essbase Server.

2. Add the custom-defined function to the calculation script or formula.

   For example, to use JSUM, use this calculation script:

   ```
   "New York" = @JSUM(@LIST(2.3, 4.5, 6.6, 1000.34));
   ```

   Use this calculation script with the Sample.Basic sample database, or replace “New York” with the name of a member in a test database.

3. Save the calculation script or formula, and then run it as usual.

Updating Custom-Defined Functions

To update an Essbase custom defined function, determine if it is local or global in scope, shut down the impacted application(s), replace the `.jar` file that contains the code for the function, and re-register the function.

The procedure for updating custom-defined functions depends on these conditions:

- Whether the function is registered locally or globally
- Whether the signature of the custom-defined function—class name, method name, or input parameters—has been changed in the Java code for the custom-defined function

Typically, to update a custom-defined function, you must replace the `.jar` file that contains the code for the function, and then re-register the function. If, however, the signature of the custom-defined function has not changed, and the function has only one set of input parameters (it is not an overloaded method), you can replace the `.jar` file that contains the function.

> **Note:**
>
> Only DBAs should update global custom-defined functions.

To update a custom-defined function:

1. Determine whether the function is local or global.

   See Viewing Custom-Defined Functions.

2. Make the changes to the Java class for the custom-defined function and use Java test programs to test its output.

3. Compile the Java classes and archive them in a new `.jar` file, using the same name as the previous `.jar` file.

   Include any other classes and methods for custom-defined functions that were included in the previous `.jar` file.

4. Perform an action, depending on whether you are updating a local or global custom-defined function:

   a. Local: Shut down any Essbase applications that use the functions in the `.jar` file.

   b. Global: Shut down all Essbase applications

   If you are unsure which Essbase applications use which functions in the `.jar` file, shut down all Essbase applications.

5. Copy the new `.jar` file to Essbase Server, replacing the existing `.jar` file with the same name.

6. If the signature of the custom-defined function has not changed, skip to step 8.
7. To replace the custom-defined function, use the **create or replace function** MaxL statement. For example:
   - Local:

     ```
     create or replace function sample.'@JSUM'
     as 'CalcFunc.sum';
     ```

   - Global:

     ```
     create or replace function '@JSUM'
     as 'CalcFunc.sum';
     ```

8. Restart the applications that you shut down, which updates the catalog.

Viewing Custom-Defined Functions

View a custom-defined function in Essbase to determine whether it has been registered successfully and whether it is local or global in scope. Custom-defined functions are not displayed until they have been created and registered.

To view a custom-defined function:

Use the display function MaxL statement.

For example, use the following MaxL statement to view the custom-defined functions in the Sample application and any registered global functions:

```
display function Sample;
```

The **display function** statement lists global functions without an application name to indicate that they are global. If the application contains a function with the same name as a global function, only the local function is listed.

Removing Custom-Defined Functions

To remove Essbase custom defined functions, first be sure that they are not in use. Then, shut down the application(s) in which the CDFs are defined, remove the CDFs by issuing the MaxL **drop function** statement, and restart the affected application(s).

The following permissions are required to remove a custom-defined function:

- Local: At least Application Manager permission for the application
- Global: System Administrator permission

Before removing custom-defined functions, you should verify that no calculation scripts or formulas are using them. Global custom-defined functions can be used in calculation scripts and formulas across Essbase Server, so you must verify that no calculation scripts or formulas on Essbase Server are using a global custom-defined function before removing it.

> **Caution:**
>
> Remove global custom-defined functions only when users are not accessing Essbase cubes, and calculation routines are not being performed.

To remove a custom-defined function:

1. Determine whether the function is local or global.

   See Viewing Custom-Defined Functions.

2. Perform an action, depending on whether you are removing a local or global custom-defined function:

   a. Local: Shut down any Essbase applications that use the functions in the `.jar` file.

   b. Global: Shut down all Essbase applications.

3. To remove the custom-defined function, use the drop function MaxL statement. For example:

   - Local:

     ```
     drop function Sample.'@JSUM';
     ```

   - Global:

     ```
     drop function '@JSUM';
     ```

4. Restart the applications that you shut down, which updates the catalog.

Copying Custom-Defined Functions

You can copy custom-defined functions to any Essbase Server and application to which you have appropriate access.

To copy a custom-defined function, use the create or replace function as MaxL statement.

Performance Considerations for Custom-Defined Functions

Because custom-defined functions are implemented as an extension of the Essbase calculator framework, you can expect CDFs to operate less efficiently than native Essbase calc functions.

To optimize performance, limit the use of custom-defined functions to calculations that you cannot perform with native Essbase calculation commands and functions, particularly in applications where calculation speed is critical.

28 Developing Custom-Defined Calculation Macros

Essbase custom-defined macros (CDM) enable you to combine multiple calculation functions into a single function. You can use them in calculation scripts or formulas.

Create and test new macros locally within a test application. Register custom-defined macros globally only after you have tested them in a test application and are ready to use them in a production environment.

To create and manage custom-defined macros, you must be Database Manager or higher.

Custom defined macros are supported only for block storage cubes.

Naming Custom-Defined Macros

Essbase custom-defined macro names must be unique and start with the @ symbol. Local macros are prepended with the application name.

Follow these guidelines when naming custom-defined macros:

- Start the macro name with the "@" symbol; for example, @MYMACRO. The rest of a name can contain letters, numbers, and the following symbols: @, #, $, and _. Macro names must not contain spaces.
- For macros that are called only by other macros, start the macro name with "@_", to distinguish it from general-use macros and functions.
- Give macros unique names. Additionally, a macro name must be different from the names of custom-defined functions and from the names of existing calculation functions.

> **Note:**
>
> If an application contains a local macro that has the same name as a global macro, the local macro takes precedence and is used for calculation.

- For local macros, you must prepend the application name to the macro name, separating the application name from the macro name with a period:

  AppName.@MacroName

  For example:

  ```
  Sample.@MYMACRO
  ```

- Because global macros are available to any application running on the Essbase Server where the macro was created, you do not assign an application name to it.

Creating Custom-Defined Macros

When you create a custom defined macro, Essbase registers it in a catalog. Your macro can be global or local to an application. You can then use it in formulas and calculation scripts, until the macro is removed from the catalog.

You can register a custom-defined macro in the following ways:

- As local, in which the macro is available only in the Essbase application in which it was created
- As global, in which the macro is available to all Essbase applications running on the Essbase Server where the macro was created

To create a custom-defined macro:

Use the create macro MaxL statement.

The following MaxL statement creates a local macro named @COUNTRANGE for use in the Sample application:

```
create macro Sample.'@COUNTRANGE'(Any) AS
'@COUNT(SKIPMISSING, @RANGE(@@S))'
spec '@COUNTRANGE(MemberRange)'
comment 'counts all non-missing values';
```

The following MaxL statement creates a global macro named @COUNTRANGE:

```
create macro'@COUNTRANGE'(Any) AS
'@COUNT(SKIPMISSING, @RANGE(@@S))'
spec '@COUNTRANGE(MemberRange)'
comment 'counts all non-missing values';
```

Using Custom-Defined Macros

Use your custom-defined macros the same way you use Essbase native calculation commands: by calling them in calculation scripts or associating them with outline formulas.

To use a custom-defined macro:

1. Create or open an existing calculation script or formula.
   - If it was registered locally, you must use a calculation script or formula within the application in which the macro was created.
   - If it was registered globally, you can use any calculation script or formula within any application on the Essbase Server.
2. Add the custom-defined macro to the calculation script or formula.

   For example, to use the @COUNTRANGE custom-defined macro shown earlier in this chapter, create the following calculation script:

   ```
   CountMbr = @COUNTRANGE(Sales, Jan:Dec);
   ```

   Use this calculation script with the Sample.Basic database, or replace “Sales, Jan:Dec” with a range of members in a test database.

3. Save the calculation script or formula, and then run it as usual.

Viewing Custom-Defined Macros

View a custom defined macro in Essbase to determine whether it has been successfully created, and whether it is local or global in scope.

To view a custom-defined macro, use the display macro MaxL statement.

Examples

The following MaxL statement displays only macros defined in the Sample application:

```
display macro on application Sample;
```

The following MaxL statement displays all global and local macros, if you have permission to view them:

```
display macro;
```

Updating Custom-Defined Macros

To change the syntax or behavior of a custom defined macro in Essbase, first determine whether it is local or global in scope, and then use the MaxL statement **create or replace macro** to update its definition.

To update a custom-defined macro:

1. Determine whether the macro is registered locally or globally.

   See Viewing Custom-Defined Macros.

2. To update the macro definition; use the create macro MaxL statement.

Examples

The following MaxL statement changes the local macro @COUNTRANGE, which is used only in the Sample application:

```
create or replace macro Sample.'@COUNTRANGE'(Any)
as '@COUNT(SKIPMISSING, @RANGE(@@S))';
```

The following MaxL statement changes the global macro @COUNTRANGE:

```
create or replace macro '@COUNTRANGE'(Any)
as '@COUNT(SKIPMISSING, @RANGE(@@S))';
```

Copying Custom-Defined Macros

You can copy custom-defined macros to any Essbase Server and application to which you have appropriate access. You must be Database Manager or higher.

To copy a custom-defined macro, use the create or replace macro MaxL statement.

Example

The following MaxL statement creates a copy of @COVARIANCE macro:

```
create macro Sample.'@COVARIANCE2'(single, single) as '@COVARIANCE'(single,
single);
```

Removing Custom-Defined Macros

To remove a custom defined macro from Essbase, first determine whether it is local or global in scope, and then use the MaxL **drop macro** statement.

To remove a custom-defined macro:

1. Determine whether the macro is registered locally or globally.

   See Viewing Custom-Defined Macros.

2. Verify that no calculation scripts or formulas are using the custom-defined macro.
3. To remove the macro from the catalog of macros, use the **drop macro** MaxL statement.
4. Restart all applications associated with the macro.

   See Refreshing the Catalog of Custom-Defined Macros.

The following MaxL statement removes the local macro @COUNTRANGE, which is used only in the Sample application:

```
drop macro Sample.'@COUNTRANGE';
```

The following MaxL statement removes the global macro @COUNTRANGE:

```
drop macro '@COUNTRANGE';
```

Refreshing the Catalog of Custom-Defined Macros

Refresh the Essbase catalog of custom-defined macros after you add, update, or remove macros. Use the MaxL statement **refresh custom definitions** if the changes are local, or restart the Essbase Server if the changes are global.

To refresh the catalog of custom-defined macros for all applications on a server, restart the server. See Start, Stop, and Check Servers for independent deployments, or Use Commands to Start, Stop, and View Status of Processes for stack deployment on OCI.

To refresh the catalog of custom-defined macros for one application, use the **refresh custom definitions** MaxL statement.

For example, the following MaxL statement refreshes the catalog of custom-defined macros for the Sample application:

```
refresh custom definition on application sample;
```

29
Writing MDX Queries

MDX is a SQL-like language you can use to issue queries that retrieve data from Essbase. MDX is also used to define formulas on ASO cubes, query metadata, qualify member names, and delineate subsets of data or metadata. The best way to learn MDX is to write queries.

This section helps you learn MDX by writing queries against the Sample.Basic cube in a series of exercises.

Prerequisites for Writing MDX Queries

To complete the exercises, you will need:

- a text editor, to write the MDX queries
- Access to an Essbase instance with Sample Basic

  If you need to get Sample Basic, follow the steps in Create a Sample Cube to Explore Outline Properties (just do the import, and skip setting the outline properties)
- MaxL Client, to issue the queries to Essbase

Build an MDX Query Template

Learn the basic format of an MDX query, so you can get started using MDX with Essbase. Similarly to SQL statements, MDX queries usually start with SELECT.

In this section you will create a template to use as a basis for developing simple MDX queries.

Most queries can be built upon the following grammatical framework:

```
SELECT
  {}
ON COLUMNS
FROM Sample.Basic
```

SELECT in line 1 is the keyword that begins the main body of MDX statements.

The braces `{ }` in line 2 are a placeholder for a *set*. In the above query, the set is empty, but the braces remain as a placeholder.

Exercise 1: Create an MDX Query Template

To create a query template:

1. Create a folder to store sample queries that can be run against the Sample.Basic cube.
2. Using a text editor, type the following code into a blank file:

   ```
   SELECT
     {}
   ON COLUMNS
   FROM Sample.Basic
   ```

3. Save the file as `qry_blank.txt`.

MDX Sets and Tuples

MDX **sets** contain **tuples**, and MDX tuples contain **member names**. Learn the difference between sets, tuples and member names, and how they fit into Essbase MDX queries.

Write your first query, and run it in the MaxL Client to retrieve some data from the Sample Basic cube.

Sets

An MDX set can be empty, or be a collection of tuples, or a collection of sets.

For example, the following is an empty set.

```
{ }
```

A set must be enclosed in curly braces `{}`, except in cases where the set is represented by an MDX function that returns a set (more about functions later).

The following is a set consisting of one tuple.

```
  {[Cola]}
```

In the following query, `{([Cola], [Actual])}` is also a set consisting of one tuple, though in this case, the tuple has more than one member name.

```
SELECT
  {([Cola], [Actual])}
ON COLUMNS
FROM Sample.Basic
```

`{([Cola], [Actual])}` is a tuple consisting of two members (Cola and Actual) from two dimensions (Product and Scenario).

Dimensionality Rule

When a set has multiple tuples, the members of each tuple must represent the same Essbase dimensions, in the same order. In other words, all the tuples must have the same dimensionality as the others.

- OK: The following set consists of two tuples of the same dimensionality:

  ```
  {(West, Feb), (East, Mar)}
  ```

- Not OK: The following set breaks the dimensionality rule, because Feb and Sales are from different dimensions:

  ```
  {(West, Feb), (East, Sales)}
  ```

- Not OK: The following set breaks the dimensionality rule, because although the two tuples contain the same dimensions, the order of dimensions is reversed in the second tuple:

  ```
  {(West, Feb), (Mar, East)}
  ```

Tuples and Member Names

A *tuple* is a way to refer to a member or a member combination from any number of dimensions. For example, in the Sample.Basic cube, all of these are valid tuples:

- `Jan`
- `(Jan, Sales)`
- `([Jan],[Sales],[Cola],[Utah],[Actual])`

The member names can be specified in these ways:

- By specifying the actual name or the alias; for example:
  - `Cola`
  - `Actual`
  - `COGS`
  - `[100]`

  If the member name starts with number or contains spaces, it should be within brackets; for example, `[100]` or `[New York]`. However, the member name brackets are recommended for all member names, for clarity and code readability.

  If the member name starts with an ampersand (&) , it should be within quotation marks; for example, `["&xyz"]`. This is because the leading ampersand is reserved for substitution variables (see Variables in MDX Queries). You can also specify it as `StrToMbr("&100")`.

  For attribute members, the long name (qualified to uniquely identify the member) should be used; for example, `[Ounces_12]` instead of `[12]`.

- By specifying dimension name or any one of the ancestor member names as a prefix to the member name; for example, `[Product].[100-10]` and `[Diet].[100-10]`. This practice is recommended for all member names, as it eliminates ambiguity and enables you to refer accurately to shared members. See: Qualifying Members by Differentiating Ancestor

  > **Note:**
  >
  > Do not use multiple ancestors in the member name qualification. Essbase returns an error if multiple ancestors are included. For example, `[Market].[New York]` and `[East].[New York]` are valid names for New York; however, `[Market].[East].[New York]` returns an error.

- By specifying the name of a calculated member defined in the WITH Section.

Exercise 2: Run Your First Query

Recall that the braces `{}` in line 2 of your query template are a placeholder for a set. In this exercise, we will add a set to the query and run it.

To run the query:

1. Open `qry_blank.txt`, the query template you created from Build an MDX Query Template.
2. Because a set can be as simple as one tuple, add the tuple within the curly braces `{ }` that hold the set.

   Type `[Jan]` inside the `{ }` braces in line 2:

   ```
   SELECT
     {[Jan]}
   ON COLUMNS
   FROM Sample.Basic
   ```

3. Save the query as `qry_first.txt`.
4. Ensure that Essbase is running.
5. Start the MaxL Client and log in with a valid user name and password. For example:

   ```
   login admin1 my_Pa55w0rD on "https://myserver.example.com:9001/essbase/
   agent";
   ```

6. Copy and paste the entire SELECT query into the MaxL Client, but do not press **Enter** yet.
7. Enter a semicolon at the end, anywhere after Basic but before pressing **Enter**. (The semicolon is not an MDX requirement, but required by the MaxL Client to indicate the end of a statement).
8. Press **Enter** to send the query to Essbase.

   The results should be similar to the following:

   ```
   Jan
     8024
   ```

MDX Query Layout with Axes and Cube Specification

An MDX axis is an instruction shaping the grid layout of query results from an Essbase cube. ON COLUMNS and ON ROWS are axis keywords that describe where the results should appear. A cube specification includes the FROM keyword, and tells Essbase which cube to query.

MDX Axis

Axes fit into MDX queries after the Select:

```
SELECT <axis> [, <axis>...]
FROM <database>
```

In the following query, the axis specification is `{Jan} ON COLUMNS`:

```
SELECT
  {Jan} ON COLUMNS
FROM Sample.Basic
```

At least one axis must be specified in any MDX query.

Up to 64 axes may be indicated, beginning with AXIS(0) and continuing with AXIS(1)...AXIS(63). Using more than three axes is uncommon. The order of axes is not important; however, when a set of axes 0 through *n* are specified, no axis between 0 and *n* should be skipped. Additionally, a dimension cannot appear on multiple axes.

The first five axes have keyword aliases, as listed in the following table:

Table 29-1 Axes Keyword Aliases

Axes Keyword Alias	Axes
ON COLUMNS	Can be used in place of AXIS(0)
ON ROWS	May replace AXIS(1)
ON PAGES	May replace AXIS(2)
ON CHAPTERS	May replace AXIS(3)
ON SECTIONS	May replace AXIS(4)

MDX Cube Specification

A cube specification is the part of the query that determines which Essbase database is being queried. The cube specification fits into an MDX query as follows:

```
SELECT <axis> [, <axis>...]
FROM <cube>
```

The <cube> section follows the FROM keyword and should consist of delimited or nondelimited identifiers that specify first an application name and a then database name; for example, the following specifications are valid:

- `FROM Sample.Basic`
- `FROM [Sample.Basic]`
- `FROM [Sample].[Basic]`
- `FROM'Sample'.'Basic'`

Exercise 3: Running A Two-Axis Query

To run a two-axis query:

1. Open `qry_blank.txt`, the query template you created in Build an MDX Query Template.
2. Add a comma after `ON COLUMNS`; then add a placeholder for a second axis by adding `ON ROWS`:

```
SELECT
  {}
ON COLUMNS,
  {}
ON ROWS
FROM Sample.Basic
```

3. Save the new query template as `qry_blank_2ax.txt`.

4. As the set specification for the column axis, enter the Product members 100-10 and 100-20. For example:

```
SELECT
  {[100-10],[100-20]}
ON COLUMNS,
  {}
ON ROWS
FROM Sample.Basic
```

   Because these member names contain special characters, you must use brackets. The convention used here, to enclose all member names in brackets even if they do not contain special characters, is recommended.

5. As the set specification for the row axis, enter the Year members Qtr1 through Qtr4.

```
SELECT
  {[100-10],[100-20]}
ON COLUMNS,
  {[Qtr1],[Qtr2],[Qtr3],[Qtr4]}
ON ROWS
FROM Sample.Basic
```

6. Save the query as `qry_2ax.txt`.
7. Paste the query into the MaxL Client and run it, as described in the first exercise (in MDX Sets and Tuples).

The results of the query should look like the following:

Table 29-2 Results: Running A Two-Axis Query

	100-10	100-20
Qtr1	5096	1359
Qtr2	5892	1534
Qtr3	6583	1528
Qtr4	5206	1287

Exercise 4: Querying Multiple Dimensions on a Single Axis

To query multiple dimensions on a single axis:

1. Open `qry_blank_2ax.txt`, the query template you created in the previous exercise.
2. On the column axis, specify two tuples, each of which is a member combination rather than a single member. Enclose each tuple in parentheses, because multiple members are represented in each tuple.

```
SELECT
  {([100-10],[East]), ([100-20],[East])}
ON COLUMNS,
  {}
ON ROWS
FROM Sample.Basic
```

3. On the row axis, specify four two-member tuples, nesting each Quarter with Profit:

```
SELECT
  {([100-10],[East]), ([100-20],[East])}
ON COLUMNS,
  {
  ([Qtr1],[Profit]), ([Qtr2],[Profit]),
  ([Qtr3],[Profit]), ([Qtr4],[Profit])
  }
ON ROWS
FROM Sample.Basic
```

4. Save the query as `qry_1ax.txt`.
5. Paste the query into the MaxL Client and run it, as described in the first exercise (in MDX Sets and Tuples).

   The results should be similar to the following:

Table 29-3 Results: Querying Multiple Dimensions on a Single Axis

		100-10	100-20
		East	East
Qtr1	Profit	2461	212
Qtr2	Profit	2490	303
Qtr3	Profit	3298	312
Qtr4	Profit	2430	287

Use MDX Functions to Build Sets

You can use MDX functions to operate on Essbase metadata or data. Functions may return members, sets, values, tuples, or strings. They are useful whether you are using MDX to analyze, update, or export data.

Learn to use MemberRange and CrossJoin functions, by trying the exercises.

This introduction to MDX functions focuses on a few functions that generate sets. Rather than manually entering sets member-by-member or tuple-by-tuple into an MDX query, you can replace such enumerations with a simple function expression. MDX functions can return sets, as well as other values.

For example, Children is a set function. It returns the set of child members of the input member. Therefore, `Children(Qtr1)` returns `{Jan, Feb, Mar}`.

Exercises that follow below can help you learn to use MDX functions in simple queries. A complete reference of MDX functions supported by Essbase is listed at MDX Function List.

Exercise 5: Using the MemberRange Function

The MemberRange MDX function returns a range of members inclusive of and between two specified members of the same generation. Its syntax is as follows:

```
MemberRange (member1, member2, [,layertype])
```

where the first argument you provide is the member that begins the range, and the second argument is the member that ends the range. The *layertype* argument is optional.

> **Note:**
>
> An alternate syntax for MemberRange is to use a colon between the two members, instead of using the function name: *member1* : *member2.*

To use the MemberRange function:

1. Open `qry_blank.txt`, the query template you created in Build an MDX Query Template.
2. Delete the braces `{}`, which are unnecessary when you are using a function to return the set.
3. Use the colon operator to select a member range of Qtr1 through Qtr4:

```
SELECT
  [Qtr1]:[Qtr4]
ON COLUMNS
FROM Sample.Basic
```

4. Paste the query into the MaxL Client and run it, as described in the first exercise (in MDX Sets and Tuples).

   Qtr1, Qtr2, Qtr3, and Qtr4 are returned.
5. Use the MemberRange function to select the same member range, Qtr1 through Qtr4.

```
SELECT
  MemberRange([Qtr1],[Qtr4])
ON COLUMNS
FROM Sample.Basic
```

6. Paste the query into the MaxL Client and run it.
7. Save the query as `qry_member_range_func.txt`.

Exercise 6: Using the CrossJoin Function

The CrossJoin function returns the cross product of two sets from different Essbase dimensions. Its syntax is as follows:

```
CrossJoin(set,set)
```

This function takes two sets from different dimensions as input, and creates a set that is a cross product of them. This is useful for creating symmetric reports.

To use the CrossJoin function:

1. Open `qry_blank.txt`, the query template you created in Build an MDX Query Template.
2. Replace the braces `{}` from the columns axis with `CrossJoin()`.

```
SELECT
  CrossJoin ()
```

```
ON COLUMNS,
  {}
ON ROWS
FROM Sample.Basic
```

3. Add two comma-separated pairs of braces as placeholders for the two set arguments you will provide to the CrossJoin function:

```
SELECT
  CrossJoin ({}, {})
ON COLUMNS,
  {}
ON ROWS
FROM Sample.Basic
```

4. In the first set, specify the Product member `[100-10]`. In the second set, specify the Market members `[East]`, `[West]`, `[South]`, and `[Central]`.

```
SELECT
  CrossJoin ({[100-10]}, {[East],[West],[South],[Central]})
ON COLUMNS,
  {}
ON ROWS
FROM Sample.Basic
```

5. On the row axis, use CrossJoin to cross a set of Measures members with a set containing `Qtr1`:

```
SELECT
  CrossJoin ({[100-10]}, {[East],[West],[South],[Central]})
ON COLUMNS,
  CrossJoin (
    {[Sales],[COGS],[Margin %],[Profit %]}, {[Qtr1]}
  )
ON ROWS
FROM Sample.Basic
```

6. Save the query as `qry_crossjoin_func.txt`.
7. Paste the query into the MaxL Client and run it.

As you experiment with using CrossJoin, notice that the order of arguments affects the order of tuples in the output.

Results of the query are shown below:

Table 29-4 Results: Using the CrossJoin Function

		100-10	100-10	100-10	100-10
		East	West	South	Central
Sales	Qtr1	5731	3493	2296	3425
COGS	Qtr1	1783	1428	1010	1460
Margin %	Qtr1	66.803	59.118	56.01	57.372

Table 29-4 (Cont.) Results: Using the CrossJoin Function

		100-10	100-10	100-10	100-10
Profit %	Qtr1	45.82	29.974	32.448	24.613

> **Note:**
>
> Consider using CrossJoinAttribute if the input sets are a base dimension and its attribute dimension.

Exercise 7: Using the Children Function

The Children function returns a set of all child members of the given member. Use this syntax:

```
Children (member)
```

> **Note:**
>
> An alternate syntax for Children is to use it as an operator on the input member, as follows: *member.Children*. We will use the operator syntax in this exercise.

To use the Children function to introduce a shortcut in the first axis specification:

1. Open `qry_crossjoin_func.txt`, the query you built in in the previous exercise.
2. In the second set of the column axis specification, replace `[East],[West],[South],[Central]` with `[Market].Children`.

```
SELECT
  CrossJoin ({[100-10]}, {[Market].Children})
ON COLUMNS,
  CrossJoin (
    {[Sales],[COGS],[Margin %],[Profit %]}, {[Qtr1]}
  )
ON ROWS
FROM Sample.Basic
```

3. Save the query as `qry_children_func.txt`.
4. Paste the query into the MaxL Client and run it.

   You should see the same results as those returned for the previous, CrossJoin exercise.

Referencing Levels and Generations with MDX

Some MDX functions perform set operations based on an input *layer* argument. The layer represents a generation or level of an Essbase dimension. Learn to reference a set using the Members function.

In MDX, the concept of a layer refers to generations and levels in an Essbase hierarchy.

In Essbase, generation numbers begin counting with 1 at the dimension name; higher generation numbers are those closest to leaf members in a hierarchy.

Level numbers begin with 0 at the leaf-most part of the hierarchy, and the highest level number is a dimension name.

You can specify a layer argument in the following ways:

- Generation or level name; for example, `States` or `Regions`.
- The dimension name along with the generation or level name; for example, `Market.Regions` and `[Market].[States]`.
- The Levels function with a dimension and a level number as input. For example, `[Year].Levels(0)`.
- The Level function with a member as input. For example, `[Qtr1].Level` returns the level of quarters in Sample.Basic, which is level 1 of the Market dimension.
- The Generations function with a dimension and a generation number as input. For example, `[Year].Generations (3)`.
- The Generation function with a member as input. For example, `[Qtr1].Generation` returns the generation of quarters in Sample.Basic, which is generation 2 of the Market dimension.

> **Note:**
>
> In the Sample.Basic database, Qtr1 and Qtr4 are in the same layer. This means that Qtr1 and Qtr4 are also in the same generation. However, in a different database with a ragged hierarchy, Qtr1 and Qtr4 might not necessarily be in the same level, although they are in the same generation. For example, if the hierarchy of Qtr1 drills down to weeks, and the hierarchy of Qtr4 stops at months, Qtr1 is one level higher than Qtr4, but they are still in the same layer.

Exercise 8: Using the Members Function

Use the Members function to return all members of a specified generation or level. When used with a layer argument, the syntax is:

`Members (`*layer*`)`

where the *layer* argument indicates the generation or level of members to return.

> **Note:**
>
> An alternate syntax for Members is *layer*`.Members`.

To use the Members function:

1. Open `qry_blank.txt`, the query template you created in Build an MDX Query Template.
2. Delete the braces `{}`, which are unnecessary when you are using a function to return a set.
3. Use the Members function and the Levels function to select all level 0 members in the Market dimension of Sample.Basic:

```
SELECT
  Members(Market.levels(0))
ON COLUMNS
FROM Sample.Basic
```

4. Save the query as `qry_members_func.txt`.
5. Paste the query into the MaxL Client and run it, as described in the first exercise (in MDX Sets and Tuples).

   Results: All states in the Market dimension are returned.

Use a Slicer Axis to Set MDX Query Point-of-View

A *slicer* axis is a way to limit an MDX query to consider only a specific area of the Essbase cube. Learn to use the slicer in the WHERE clause by trying the sample exercise.

The slicer, if used, must be in the WHERE section of an MDX query. Also, the WHERE section must be the last component of the query, following the cube specification (the FROM section):

```
SELECT {set}
ON axes
FROM cube
WHERE slicer
```

Use the slicer axis to set the context of the query; it is usually the default context for all the other axes.

To select only Actual Sales in the Sample.Basic cube, excluding budgeted sales, the WHERE clause might look like the following:

```
WHERE ([Actual], [Sales])
```

Because (Actual, Sales) is specified in the slicer axis, you need not include them in the ON AXIS(*n*) set specifications.

> **Note:**
>
> The same dimension cannot appear on other axes and the slicer axis. To filter an axis using criteria from its own dimension, you can use a sub select.

Exercise 9: Limiting the Results with a Slicer Axis

To use the slicer axis to limit results:

1. Open `gry_crossjoin_func.txt`, the query you built in Exercise 6 of Use MDX Functions to Build Sets.
2. Paste the query into the MaxL Client and run it, as described in the first exercise (in MDX Sets and Tuples).

   Note the results in one of the data cells; for example, notice that the first tuple, `([Cola],[East],[Sales],[Qtr1])`, has a value of 5731.
3. Add a slicer axis to limit the data returned to budgeted values only.

```
SELECT
  CrossJoin ({[100-10]}, {[East],[West],[South],[Central]})
ON COLUMNS,
  CrossJoin (
    {[Sales],[COGS],[Margin %],[Profit %]}, {[Qtr1]}
  )
ON ROWS
FROM Sample.Basic
WHERE (Budget)
```

4. Paste the query into the MaxL Client and run it.
5. Notice that the value for tuple `([Cola],[East],[Sales],[Qtr1])` is now 5020.
6. Save the query as `qry_slicer_axis.txt`.

Common MDX Relationship Functions

MDX relationship functions return sets or members based on hierarchical member relationships in the Essbase cube outline.

The following MDX relationship functions return sets.

Table 29-5 List of MDX Relationship Functions That Return Sets

Relationship Function	Description
Children	Returns the children of the input member.
Siblings	Returns the siblings of the input member.
Descendants	Returns the descendants of a member, with varying options.

The following MDX relationship functions return a single member rather than a set:

Table 29-6 List of MDX Relationship Functions That Return a Single Member

Relationship Function	Description
Ancestor	Returns an ancestor at the specified layer.
Cousin	Returns a child member at the same position as a member from another ancestor.
Parent	Returns the parent of the input member.
FirstChild	Returns the first child of the input member.
LastChild	Returns the last child of the input member.

Table 29-6 (Cont.) List of MDX Relationship Functions That Return a Single Member

Relationship Function	Description
FirstSibling	Returns the first child of the input member's parent.
LastSibling	Returns the last child of the input member's parent.

Exercise 10: Try Some Relationship Function Examples from the Documentation

To learn how relationship functions work:

1. Click on a link to a function in one of the tables above, or click Children.
2. Read the example, and copy the SELECT query to your clipboard.
3. Paste the query into the MaxL Client, add a closing semicolon, and run the query, as described in the first exercise (in MDX Sets and Tuples).

MDX Functions for Set Operations

You can use these MDX functions with Essbase to compare, join, combine, or reduce sets: CrossJoin, CrossJoinAttribute, Distinct, Except, Generate, Head, Intersect, Subset, Tail, and Union.

Learn the difference between Intersect and Union functions by trying the exercises.

The following set functions operate on input sets without deriving further information from a cube:

Table 29-7 List of Pure Set Functions

Pure Set Function	Description
CrossJoin, CrossJoinAttribute	Returns a cross-section of two sets from different dimensions.
Distinct	Deletes duplicate tuples from a set.
Except	Returns a subset containing the differences between two sets.
Generate	An iterative function. For each tuple in *set1*, returns *set2*.
Head	Returns the first *n* members or tuples present in a set.
Intersect	Returns the intersection of two input sets.
Subset	Returns a subset from a set, in which the subset is a numerically specified range of tuples.
Tail	Returns the last *n* members or tuples present in a set.
Union	Returns the union of two input sets.

Exercise 11: Using the Intersect Function

The MDX Intersect function returns the intersection of two input sets, optionally retaining duplicates. Use it to compare sets by finding tuples that are present in both sets.

The syntax to follow is:

```
Intersect (set, set [,ALL])
```

1. Open `qry_blank.txt`, the query template you created in Build an MDX Query Template.
2. Delete the empty set braces `{}` from the axis, and replace them with `Intersect()`. Leave some room inside the Intersect braces for adding more code. For example:

   ```
   SELECT
      Intersect (

      )
   ON COLUMNS
   FROM Sample.Basic
   ```

3. Add two comma-separated pairs of braces to use as placeholders for the two set arguments you will provide to the Intersect function. For example:

   ```
   SELECT
      Intersect (
      { },
      { }
      )
   ON COLUMNS
   FROM Sample.Basic
   ```

4. Specify children of East as the first set argument. For example:

   ```
   SELECT
      Intersect (
      { [East].children },
      { }
      )
   ON COLUMNS
   FROM Sample.Basic
   ```

5. For the second set argument, specify all members of the Market dimension that have a UDA of “Major Market.” For example:

   ```
   SELECT
      Intersect (
      { [East].children },
      { UDA([Market], "Major Market") }
      )
   ON COLUMNS
   FROM Sample.Basic
   ```

6. Paste the query into the MaxL Client and run it, as described in the first exercise (in MDX Sets and Tuples).

All children of East that have a UDA of "Major Market" are returned. For example:

```
New York    Massachusetts    Florida

8202        6172             5029
```

7. Save the query as `qry_intersect_func.txt`.

Exercise 12: Using the Union Function

The MDX Union function joins two input sets, optionally retaining duplicates. Use it to combine two sets together into one set.

The syntax to follow is:

```
Union (set, set [,ALL])
```

1. Open `qry_intersect_func.txt`, the query you built in the previous exercise.
2. Replace **Intersect** with **Union**.
3. Save the query as `qry_union_func.txt`.
4. Paste the query into the MaxL Client and run it.

   While Intersect returned a set containing only those children of East that have a UDA of Major Market, Union returns a bigger set. It includes all children of East, AND all Market members that have a UDA of Major Market.

```
 (New York)      (Massachusetts)  (Florida)       (Connecticut)    (New
Hampshire)  (East)          (California)    (Texas)          (Central)
(Illinois)      (Ohio)          (Colorado)
+----------------+----------------+----------------+----------------
+----------------+----------------+----------------+----------------
+----------------+----------------+----------------+----------------
            8202            6712            5029
3093            1125            24161           12964    6425
38262           12577             4384            7227
```

Reusable Sets and Members: MDX WITH Section

Defining members and sets in the MDX query's WITH section helps you filter data without impacting the cube. A calculated member is a logical member that exists only within the query. A named set is a logical set that exists only within the query. Try a sample exercise.

Calculated members and named sets are logical entities in query that can be used multiple times during the life of the query. Calculated members and named sets can save time in lines of code written as well as in execution time. The optional WITH section at the beginning of an MDX query is where you define the calculated members and/or named sets.

The following query uses a calculated member:

```
WITH
MEMBER [Measures].[Max Qtr2 Sales] AS
  'Max (
    {[Year].[Qtr2]},
    [Measures].[Sales]
```

```
  )'
SELECT
{ [Measures].[Max Qtr2 Sales] } on columns,
{ [Product].children } on rows
FROM Sample.Basic
```

The following query uses a named set:

```
WITH SET [NewSet]
AS 'CrossJoin([Product].Children, [Market].Children)'
SELECT
   Filter([NewSet], NOT IsEmpty([NewSet].CurrentTuple))
ON COLUMNS
FROM Sample.Basic
WHERE
   {[Sales]}
```

Calculated Members

A calculated member is a hypothetical member that exists for the duration of the query execution. Calculated members enable complex analysis without the need to add physical members to the cube outline. Calculated members store calculation results performed on physical members.

Use the following guidelines for calculated member names:

- Associate the calculated member with a dimension; for example, to associated the member MyCalc with the Measures dimension, name it `[Measures].[MyCalc]`.
- Do not use actual member names to name calculated members; for example, do not name a calculated member `[Measures].[Sales]`, because Sales already exists in the Measures dimension.

Setting the solve order for each calculated member is recommended when you use multiple calculated members to create ratios or custom totals.

Named Sets

You define named sets using WITH SET keywords, before the SELECT portion of the query. Doing so is useful because you can reference the set by name when building the SELECT portion of the query.

For example, the named set `Best5Prods` identifies a set of the five top-selling products in December:

```
WITH
SET [Best5Prods]
  AS
  'Topcount (
    [Product].members,
    5,
    ([Measures].[Sales], [Scenario].[Actual], [Year].[Dec])
  )'
SELECT [Best5Prods] ON AXIS(0),
  {[Year].[Dec]} ON AXIS(1)
FROM Sample.Basic
```

Exercise 13: Creating a Calculated Member

This exercise uses the Max function, a common MDX function for calculations. It returns the maximum of values found in the tuples of a set.

The syntax to follow is:

```
Max (set, numeric_value)
```

1. Open `qry_blank_2ax.txt`, the query template you built in Exercise 3 of MDX Query Layout with Axes and Cube Specification.

   ```
   SELECT
     {}
   ON COLUMNS,
     {}
   ON ROWS
   FROM Sample.Basic
   ```

2. On the row axis set, specify the children of Product. For example:

   ```
   SELECT
     {}
   ON COLUMNS,
     {[Product].children}
   ON ROWS
   FROM Sample.Basic
   ```

3. At the beginning of the query, add a placeholder for the calculated member specification. For example:

   ```
   WITH MEMBER [].[]
    AS ''
   SELECT
     {}
   ON COLUMNS,
     {[Product].children}
   ON ROWS
   FROM Sample.Basic
   ```

4. To associate the calculated member with the Measures dimension and name it Max Qtr2 Sales, add this information to the calculated member specification. For example:

   ```
   WITH MEMBER [Measures].[Max Qtr2 Sales]
    AS ''
   SELECT
     {}
   ON COLUMNS,
     {[Product].children}
   ON ROWS
   FROM Sample.Basic
   ```

5. After the AS keyword and inside the single quotation marks, define the logic for the calculated member named Max Qtr2 Sales.

Use the Max function with the set to evaluate (Qtr2) as the first argument, and the measure to evaluate (Sales) as the second argument. For example:

```
WITH MEMBER [Measures].[Max Qtr2 Sales]
  AS '
  Max (
    {[Year].[Qtr2]},
    [Measures].[Sales]
  )'
SELECT
  {}
ON COLUMNS,
  {[Product].children}
ON ROWS
FROM Sample.Basic
```

6. The calculated member Max Qtr2 Sales is defined in the WITH section. To use it in a query, reference it on one of the axes in the SELECT portion of the query. For example:

```
WITH MEMBER [Measures].[Max Qtr2 Sales]
  AS '
  Max (
    {[Year].[Qtr2]},
    [Measures].[Sales]
  )'
SELECT
  {[Measures].[Max Qtr2 Sales]}
ON COLUMNS,
  {[Product].children}
ON ROWS
FROM Sample.Basic
```

7. Save the query as `gry_calc_member.txt`.
8. Paste the query into the MaxL Client and run it, as described in the first exercise (in MDX Sets and Tuples).

   Results of the query are shown below:

Table 29-8 Results: Creating a Calculated Member

	Max Qtr2 Sales
100	27187
200	27401
300	25736
400	21355
Diet	26787

> **Note:**
>
> Many more examples are available in the MDX reference documentation. See MDX With Section.

Iterative MDX Functions

Iterative MDX functions loop through sets of data in the Essbase cube, performing any search conditions that you specify to tailor the results. See an example that filters data based on a Boolean test.

Table 29-9 List of Iterative MDX Functions

Function	Description
Filter	Returns the subset of tuples in *set* for which the value of the search condition is TRUE.
IIF	Performs a conditional test and returns an appropriate numeric expression or set depending on whether the test evaluates to TRUE or FALSE.
Case	Performs conditional tests and returns the results you specify.
Generate	For each tuple in *set1*, returns *set2*.

Filter Function Example

The following query uses the MDX Filter function to return all Market dimension members for which the expression `IsChild([Market].CurrentMember,[East])` returns `TRUE`. The query returns all children of East.

```
SELECT
  Filter([Market].Members,
    IsChild([Market].CurrentMember,[East])
  )
ON COLUMNS
FROM Sample.Basic
```

The Filter function in MDX is comparable to the RESTRICT command in Report Writer.

Handling Missing Data with MDX

When querying an Essbase cube using MDX, you can use the NON EMPTY keywords on the axes to suppress cells containing no value. MDX functions that handle missing values include Avg, CoalesceEmpty, IsEmpty, NonEmptyCount, and NonEmptySubset. NONEMPTYMEMBER and NONEMPTYTUPLE properties help filter out empty values from large data sets.

Including the optional keywords NON EMPTY before the set specification in an axis causes suppression of slices in that axis that would contain entirely #MISSING values.

The following is the axis specification syntax with NON EMPTY:

```
<axis_specification> ::=
  [NON EMPTY] <set> ON
  COLUMNS | ROWS | PAGES | CHAPTERS |
  SECTIONS | AXIS (<unsigned_integer>)
```

For any given tuple on an axis (such as `(Qtr1, Actual)`), a *slice* consists of the cells arising from combining this tuple with all tuples of all other axes. If all of these cell values are #MISSING, the NON EMPTY keyword causes elimination of the tuple.

For example, if even one value in a row is not empty, the entire row is returned. Including NON EMPTY at the beginning of the row axis specification would eliminate the following row slice from the set returned by a query:

Qtr1					
Actual	#MISSING	#MISSING	#MISSING	#MISSING	#MISSING

In addition to suppressing missing values with NON EMPTY, you can use the following MDX functions to handle #MISSING results:

- CoalesceEmpty, which searches numeric value expressions for non #MISSING values
- IsEmpty, which returns TRUE if the value of an input numeric-value-expression evaluates to #MISSING
- Avg, which omits missing values from averages unless you use the optional IncludeEmpty flag

The NonEmptyCount MDX function returns the count of the number of tuples in a set that evaluate to non-#Missing values. Each tuple is evaluated and included in the count returned by this function. If the numeric value expression is specified, it is evaluated in the context of every tuple, and the count of non-#Missing values is returned.

On aggregate storage cubes only, the NonEmptyCount function is optimized so that the calculation of the distinct count for all cells can be performed by scanning the cube only once. Without this optimization, the database is scanned as many times as the number of cells corresponding to the distinct count. The NonEmptyCount optimization is triggered when an outline member formula has the following syntax:

```
NONEMPTYCOUNT(set, measure, exclude_missing)
```

The **exclude_missing** parameter supports the NonEmptyCount optimization on aggregate databases by improving the performance of a query that queries metrics that perform a distinct count calculation.

The NONEMPTYMEMBER and NONEMPTYTUPLE optimization properties enable MDX to query on large sets of members or tuples while skipping formula execution on non-contributing values that contain only #MISSING data.

- Use a single NONEMPTYMEMBER property clause at the beginning of a calculated member or formula expression to indicate to Essbase that the value of the formula or calculated member is empty when any of the members specified in **nonempty_member_list** are empty.
- Use a single NONEMPTYTUPLE property clause at the beginning of a calculated member or formula expression to indicate to Essbase that the value of the formula or calculated

member is empty when the cell value at the tuple given in **nonempty_member_list** is empty.

Given an input set, the NonEmptySubset MDX function returns a subset of that input set in which all tuples evaluate to non-empty. An optional value expression may be specified for the non-empty check. This function can help optimize queries that are based on a large set for which the set of non-empty combinations is known to be small. NonEmptySubset reduces the size of the set in the presence of a metric; for example, you might request the non-empty subset of descendants for specific Units.

Variables in MDX Queries

You can use predefined Essbase substitution variables in MDX to reference frequently changing information without changing your queries. To reference a variable in an MDX query or expression, enter the variable name preceded by an ampersand (&).

Substitution variables in Essbase act as placeholders for information that changes regularly. You set the substitution variables at the Essbase cube, application, or global level, and assign a value to each variable. You can change the value anytime. You must have the role of at least Database Manager to set substitution variables. See Using Substitution Variables.

To use a substitution variable in an MDX expression, consider:

- The substitution variable must be accessible from the application and cube you are querying.
- A substitution variable has two components: the name and the value.
- The variable name can be an alphanumeric combination whose maximum size is specified in Name and Related Artifact Limits. Do not use spaces, punctuation, or brackets ([ ]) in substitution variable names used in MDX.
- At the point in the expression where you want to use the variable, enter the variable name preceded by an ampersand (&); for example, where `CurMonth` is the name of the substitution variable set on the server, include `&CurMonth` in the MDX expression.
- When you perform the retrieval, Essbase replaces the variable name with the substitution value, and that value is used by the MDX expression.

For example, the expression is written showing the variable name `CurQtr` preceded by the &:

```
SELECT
  {[&CurQtr]}
ON COLUMNS
FROM Sample.Basic
```

When the expression is evaluated, the current value (Qtr1) is substituted for the variable name, so that the expression that is run is effectively:

```
SELECT
  {[Qtr1]}
ON COLUMNS
FROM Sample.Basic
```

Querying for Properties in MDX

Properties describe certain characteristics of Essbase data and metadata. MDX enables you to write queries that retrieve and analyze data based on Essbase properties, which can be intrinsic or custom. You can invoke properties on the MDX query axes, or in a value expression.

Intrinsic and Custom Properties

In MDX, *properties* describe certain characteristics of data and metadata. MDX enables you to write queries that use properties to retrieve and analyze data. Properties can be intrinsic or custom.

Intrinsic Properties

Intrinsic properties are defined for members in all dimensions. The intrinsic member properties defined for all members in an Essbase database outline are MEMBER_NAME, MEMBER_ALIAS, LEVEL_NUMBER, GEN_NUMBER, IS_EXPENSE, COMMENTS, and MEMBER_UNIQUE_NAME.

See MDX Intrinsic Properties for a description of each.

Custom Properties

MDX in Essbase supports two types of custom properties: attribute properties and UDA properties. Attribute properties are defined by the attribute dimensions in an outline. In the Sample.Basic database, the Pkg Type attribute dimension describes the packaging characteristics of members in the Product dimension. This information can be queried in MDX using the property name [Pkg Type].

Attribute properties are defined only for specific dimensions and only for a specific level in each dimension. For example, in the Sample.Basic outline, `[Ounces]` is an attribute property defined only for members in the Product dimension, and this property has valid values only for the level 0 members of the Product dimension. The [Ounces] property does not exist for other dimensions, such as Market. The [Ounces] property for a non level 0 member in the Product dimension is a NULL value. The attribute properties in an outline are identified by the names of attribute dimensions in that outline.

The custom properties also include UDAs. For example, [Major Market] is a UDA property defined on Market dimension members. It returns a TRUE value if [Major Market] UDA is defined for a member, and FALSE otherwise.

See also MDX Custom Properties.

Invoking Properties in a Query Axis

You can list the dimension and property combinations for each axis set. When a query is executed, the specified property is evaluated for all members from the specified dimension and included in the result set.

For example, on the column axis, the following query returns the GEN_NUMBER information for every Market dimension member. On the row axis, the query returns MEMBER_ALIAS information for every Product dimension member.

```
SELECT
  [Market].Members
    DIMENSION PROPERTIES [Market].[GEN_NUMBER] on columns,
  Filter ([Product].Members, Sales > 5000)
```

```
    DIMENSION PROPERTIES [Product].[MEMBER_ALIAS] on rows
FROM Sample.Basic
```

When querying for member properties using the DIMENSION PROPERTIES section of an axis, a property can be identified by the dimension name and the name of the property, or by using the property name itself. When a property name is used by itself, that property information is returned for all members from all dimensions on that axis, for which that property applies.

In the following query, the MEMBER_ALIAS property is evaluated on the row axis for Year and Product dimensions.

```
SELECT [Market].Members
  DIMENSION PROPERTIES [Market].[GEN_NUMBER] on columns,
  CrossJoin([Product].Children, Year.Children)
    DIMENSION PROPERTIES [MEMBER_ALIAS] on rows
FROM Sample.Basic
```

Invoking Properties in a Value Expression

Properties can be used inside value expressions in an MDX query. For example, you can filter a set based on a value expression that uses properties of members in the input set.

The following query returns all caffeinated products that are packaged in cans.

```
SELECT
   Filter([Product].levels(0).members,
    [Product].CurrentMember.Caffeinated and
    [Product].CurrentMember.[Pkg Type] = "Can")
       Dimension Properties
       [Caffeinated], [Pkg Type] on columns
FROM Sample.Basic
```

The following query calculates the value [BudgetedExpenses] based on whether the current Market is a major market, using the UDA [Major Market].

```
WITH
  MEMBER [Measures].[BudgetedExpenses] AS
    'IIF([Market].CurrentMember.[Major Market],
    [Marketing] * 1.2, [Marketing])'

SELECT  {[Measures].[BudgetedExpenses]} ON COLUMNS,
  [Market].Members ON ROWS
FROM Sample.Basic
WHERE ([Budget])
```

Value Type of Properties

The value of an MDX property in Essbase can be a numeric, Boolean, or string type. MEMBER_NAME and MEMBER_ALIAS properties return string values. LEVEL_NUMBER, and GEN_NUMBER properties return numeric values.

The attribute properties return numeric, Boolean, or string values based on the attribute dimension type. For example, in Sample.Basic, the [Ounces] attribute property is a numeric property. The [Pkg Type] attribute property is a string property. The [Caffeinated] attribute property is a Boolean property.

Essbase allows attribute dimensions with date types. The date type properties are treated as numeric properties in MDX. When comparing these property values with dates, use the Todate function to convert date strings to numeric before comparison.

The following query returns all Product dimension members that have been introduced on date 03/25/2018. Because the property [Intro Date] is a date type, the TODATE function must be used to convert the date string “03-25-2018” to a number before comparing it.

```
SELECT
  Filter ([Product].Members,
    [Product].CurrentMember.[Intro Date] =
    TODATE("mm-dd-yyyy","03-25-2018"))ON COLUMNS
FROM Sample.Basic
```

When a property is used in a value expression, you must use it appropriately based on its value type: string, numeric, or Boolean.

You can also query attribute dimensions with numeric ranges.

The following query retrieves Sales data for Small, Medium, and Large population ranges.

```
SELECT
  {Sales} ON COLUMNS,
  {Small, Medium, Large} ON ROWS
FROM Sample.Basic
```

When attributes are used as properties in a value expression, you can use range members to check whether a member's property value falls within a given range, using the IN operator.

For example, the following query returns all Market dimension members with the population range in Medium:

```
SELECT
  Filter(
    Market.Members, Market.CurrentMember.Population
    IN "Medium"
  )
ON AXIS(0)
FROM Sample.Basic
```

NULL Property Values

Not all members may have valid values for a given property name. For example, the MEMBER_ALIAS property returns an alternate name for a given member as defined in the outline; however, not all members may have aliases defined. In these cases A NULL value is be returned for those members that do not have aliases.

In the following query,

```
SELECT
  [Year].Members
   DIMENSION PROPERTIES [MEMBER_ALIAS]
```

```
ON COLUMNS
FROM Sample.Basic
```

None of the members in the Year dimension have aliases defined for them. Therefore, the query returns NULL values for the MEMBER_ALIAS property for members in the Year dimension.

The attribute properties are defined for members of a specific dimension and a specific level in that dimension. In the Sample.Basic database, the [Ounces] property is defined only for level 0 members of the Product dimension.

Therefore, if you query for the [Ounces] property of a member from the Market dimension, as shown in the following query, you will get a syntax error:

```
SELECT
  Filter([Market].members,
    [Market].CurrentMember.[Ounces] = 32) ON COLUMNS
FROM Sample.Basic
```

Additionally, if you query for the [Ounces] property of a non level 0 member of the dimension, you will get a NULL value.

When using property values in value expressions, you can use the function IsValid() to check for NULL values. The following query returns all Product dimension members with an [Ounces] property value of 12, after eliminating members with NULL values.

```
SELECT
  Filter([Product].Members,
    IsValid([Product].CurrentMember.[Ounces]) AND
    [Product].CurrentMember.[Ounces] = 12)
ON COLUMNS
FROM Sample.Basic
```

30 Exporting Data

You can export data from a Essbase database to move data between cubes, or for backup and migration purposes.

Topics:

- Exporting Data Using MaxL
- Exporting Text Data Using Calculation Scripts

To export whole cubes from the Essbase web interface to a workbook format, see Export a Cube to an Application Workbook.

See also Insert and Export Data with MDX.

Exporting Data Using MaxL

You can export data from an Essbase database using the **export data** MaxL statement. Data can be export serially or in parallel. If the data for a thread exceeds 2 GB, Essbase may divide the export data into multiple files with numbers appended to the file names.

- Block storage databases: You can export all data, level-0 data, or input-level data, which does not include calculated values.

  To export data in parallel, specify a comma-separated list of export files, up to a maximum of 1024 file names. The number of file names determines the number of export threads. The number of available block-address ranges limits the number of export threads that Essbase actually uses. Essbase divides the number of actual data blocks by the specified number of file names (export threads). If there are fewer actual data blocks than the specified number of export threads, the number of export threads that are created is based on the number of actual data blocks. This approach results in a more even distribution of data blocks between export threads.
- Aggregate storage databases: You can export only level-0 data, which does not include calculated values. (Level-0 data is the same as input data in aggregate storage databases.) You cannot perform upper-level data export or columnar export on an aggregate storage database.

  To export data in parallel, specify a comma-separated list of export files, from 1 to 8 file names. The number of threads Essbase uses typically depends on the number of file names you specify. However, on a very small aggregate storage database with a small number of data blocks, it is possible that only a single file will be created (in effect, performing serial export), even though parallel export to multiple files is requested.

Export files are stored in the database directory in the file catalog; for example, `Files > applications > Sample > Basic`.

Exporting Text Data Using Calculation Scripts

You can use the following calculation commands to select and format a text import file:

- DATAEXPORT

- DATAEXPORTCOND
- SET DATAEXPORTOPTIONS
- FIX...ENDFIX
- EXCLUDE...ENDEXCLUDE

Calculation script-based data export works with stored and dynamically calculated members only and provides fewer formatting options than report scripts. However, calculation script-based data exports provide decimal- and precision-based formatting options and can be faster than report scripts. The DATAEXPORT calculation command also enables export directly to relational databases, eliminating the usual intermediate import step.

The following calculation script example produces a text file that contains a subset of the database.

```
SET DATAEXPORTOPTIONS
{   DATAEXPORTLEVEL "ALL";
DATAEXPORTCOLFORMAT ON;
DATAEXPORTCOLHEADER Scenario;
};
FIX ("100-10","New York","Actual","Qtr1");
   DATAEXPORT "File" "," "actual.txt" "NULL";
ENDFIX;
```

These commands specify inclusion of all levels of data and indicate that data is to be repeated in columns, with the Scenario dimension set as the dense dimension column header for the output. The FIX command defines the data slice, and then the data is exported to a text file `actual.txt` in the database directory. Commas are used as delimiters, and missing data values are indicated by consecutive delimiters. Running this script against Sample.Basic generates the following data:

```
"Actual"
"100-10","New York","Sales","Qtr1",1998
"100-10","New York","COGS","Qtr1",799
"100-10","New York","Margin","Qtr1",1199
"100-10","New York","Marketing","Qtr1",278
"100-10","New York","Payroll","Qtr1",153
"100-10","New York","Misc","Qtr1",2
"100-10","New York","Total Expenses","Qtr1",433
"100-10","New York","Profit","Qtr1",766
"100-10","New York","Opening Inventory","Qtr1",2101
"100-10","New York","Additions","Qtr1",2005
"100-10","New York","Ending Inventory","Qtr1",2108
"100-10","New York","Margin %","Qtr1",60.01001001001001
"100-10","New York","Profit %","Qtr1",38.33833833833834
"100-10","New York","Profit per Ounce","Qtr1",63.83333333333334
```

31 Controlling Access to Database Cells Using Security Filters

When security levels defined for applications, databases, users, and groups are insufficient, Essbase security filters give you more specific control. Filters enable you to control access to individual data within a database by defining what kind of access is allowed to which parts of the database, and to whom these settings apply.

About Security Filters

When security levels defined for applications, databases, users, and groups are insufficient, Essbase security filters give you more specific control. Filters enable you to control access to individual data within a database by defining what kind of access is allowed to which parts of the database, and to whom these settings apply.

If you have Administrator permissions, you can define and assign any filters to any users or groups. Filters do not affect you.

If you have Create/Delete Applications permissions, you can assign and define filters for applications that you created.

If you have Application Manager or Database Manager permissions, you can define and assign filters within your applications or databases.

Defining Permissions Using Security Filters

Filters control security access to data values, or cells. You create filters to accommodate security needs for specific parts of a database. When you define a filter, you designate restrictions on particular database cells. When you save the filter, you give it a unique name to distinguish it from other filters. You can then assign the filters to users or groups.

For example, a manager designs a filter named RED and associates it with a database to limit access to cells containing profit information. The filter is assigned to a visiting group called REVIEWERS, so that they can read, but cannot alter, most of the database; they have no access to Profit data values.

Filters comprise one or more access settings for database members. You can specify the following access levels and apply them to data ranging from a list of members to one cell:

- None: No data can be retrieved or updated for the specified member list.
- Read: Data can be retrieved but not updated for the specified member list.

- Write: Data can be retrieved and updated for the specified member list.
- Metaread: Metadata (dimension and member names) can be retrieved and updated for the corresponding member specification.

> **Note:**
>
> The metaread access level overrides all other access levels. If additional filters for data are defined, they are enforced within any defined metaread filters. If you have assigned a metaread filter on a substitution variable and then try to retrieve the substitution variable, an unknown member error occurs, but the value of the substitution variable is displayed. This behavior is expected. Metadata security cannot be completely turned off in partitions. Therefore, do not set metadata security at the source database; otherwise, incorrect data may result at the target partition. When drilling up or retrieving on a member that has metadata security turned on and has shared members in the children, an unknown member error occurs because the prototype members of the shared members have been filtered. To avoid this error, give the prototype members of the shared members metadata security access.

Any cells that are not specified in the filter definition inherit the database access level. Filters can, however, add or remove access assigned at the database level, because the filter definition, being more data-specific, indicates a greater level of detail than the more general database access level.

Data values not covered by filter definitions default to the access levels defined for users.

Calculation access is controlled by permissions granted to users and groups. Users who have calculate access to the database are not blocked by filters—they can affect all data elements that the execution of their calculations would update.

> **Note:**
>
> During the calculation of MDX calculated members, cells to which a user does not have access are treated as #MISSING cells.

Creating Filters

You can create a filter for each set of access restrictions you need to place on database values. You need not create separate filters for users with the same access needs. After you have created a filter, you can assign it to multiple users or groups of users.

> **Note:**
>
> If you use a calculation function that returns a set of members, such as children or descendants, and it evaluates to an empty set, the security filter is not created. An error is written to the application log stating that the region definition evaluated to an empty set.

Before creating a filter, perform the following actions:

- Connect to the server and select the database associated with the filter.
- Check the naming rules for filters in Limits.

See Create Filters.

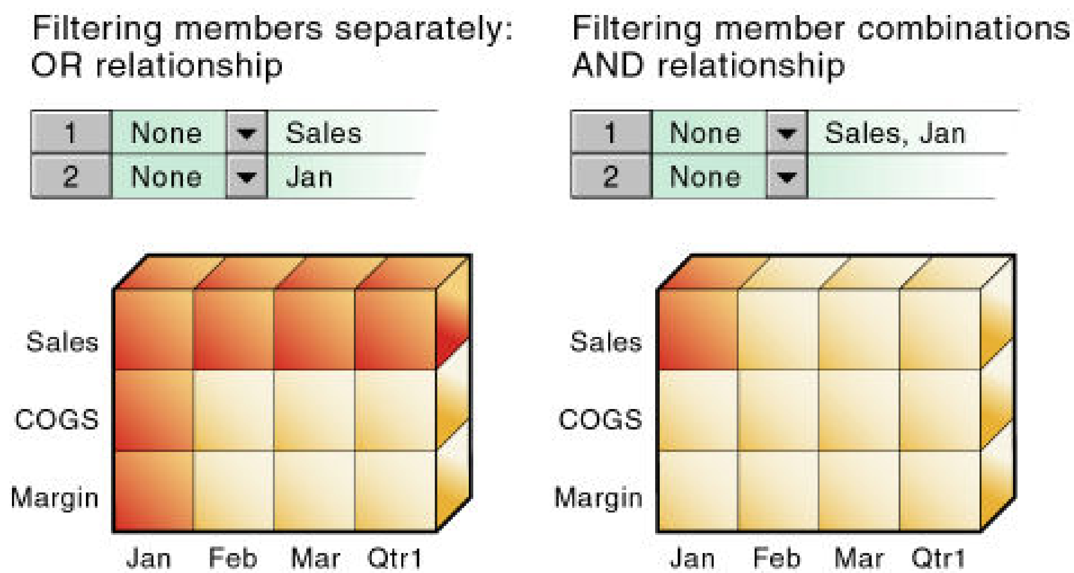
To create a filter, you can also use the **create filter** MaxL statement.

Filtering Members Versus Filtering Member Combinations

This topic illustrates different ways to control access to database cells. Data can be protected by filtering entire members or by filtering member combinations.

- Filtering members separately affects whole regions of data for those members.
- Filtering member combinations affects data at the member intersections.

Figure 31-1 How Filters Affect Data AND/OR Relationships



> **Note:**
>
> Filtering on member combinations (AND relationship) does not apply to metaread. Metaread filters each member separately (OR relationship).

Filtering Members Separately

To filter all the data for one or more members, define access for each member on its own row in Filter Editor. Filter definitions on separate rows of a filter are treated with an OR relationship.

For example, to block access to Sales or Jan, assume that user KSmith is assigned the following filter:

- Access: None. Member specification: Sales.
- Access: None. Member specification: Jan.

The next time user KSmith connects to Sample.Basic, her spreadsheet view of the profit margin for Qtr1 shows that she has no access to data values for the member Sales or the member Jan, which are marked with #NOACCESS. All data for Sales is blocked from view, as well as all data for January, inside and outside of the Sales member. Data for COGS (Cost of Goods Sold), a sibling of Sales and a child of Margin, is available, with the exception of COGS for January.

Figure 31-2 Results of Filter Blocking Access to Sales or Jan

	A	B	C	D	E
1				Product	Market
2	Sales	Jan	Scenario	#NoAccess	
3		Feb	Scenario	#NoAccess	
4		Mar	Scenario	#NoAccess	
5		Qtr1	Scenario	#NoAccess	
6	COGS	Jan	Scenario	#NoAccess	
7		Feb	Scenario	14307	
8		Mar	Scenario	14410	
9		Qtr1	Scenario	42877	
10	Margin	Jan	Scenario	#NoAccess	
11		Feb	Scenario	17762	
12		Mar	Scenario	17803	
13		Qtr1	Scenario	52943	

Filtering Member Combinations

To filter data for member combinations, define the access for each member combination using a row in Filter Editor. In filter definitions, two member sets separated by a comma are treated as union of those two member sets (an AND relationship).

For example, assume that user RChinn is assigned the following filter: Access: None. Member specification: Sales, Jan.

The next time user RChinn connects to Sample.Basic, her spreadsheet view of the profit margin for Qtr1 shows that she has no access to the data value at the intersection of members Sales and Jan, which is marked with #NoAccess. Sales data for January is blocked from view. However, Sales data for other months is available, and non-Sales data for January is available.

Figure 31-3 Results of Filter Blocking Access to Sales, Jan

	A	B	C	D	E
1			Product	Market	Scenario
2	Sales	Jan	#NoAccess		
3		Feb	32069		
4		Mar	32213		
5		Qtr1	95820		
6	COGS	Jan	14160		
7		Feb	14307		
8		Mar	14410		
9		Qtr1	42877		
10	Margin	Jan	17378		
11		Feb	17762		
12		Mar	17803		
13		Qtr1	52943		

Filtering Using Substitution Variables

Substitution variables enable you to more easily manage information that changes regularly. Each substitution variable has an assigned name and value. The Database Manager can change the value anytime. Where a substitution variable is specified in a filter, the substitution variable value at that time is used.

For example, if you want a group of users to see data only for the current month, you can set up a substitution variable named CurMonth and define a filter (MonthlyAccess) wherein you specify access, using `&CurMonth` for the member name. Using an ampersand (&) at the beginning of a specification identifies it as a substitution variable instead of a member name to Essbase. Assign the MonthlyAccess filter to the appropriate users.

Each month, you need to change only the value of the CurMonth substitution variable to the member name for the current month, such as Jan, Feb, and so on. The new value will apply to all assigned users.

See Using Substitution Variables.

Filtering with Attribute Functions

You can use filters to restrict access to data for base members sharing a particular attribute. To filter data for members with particular attributes defined in an attribute dimension, use the attribute member in combination with the @ATTRIBUTE function or the @WITHATTR function.

> **Note:**
>
> @ATTRIBUTE and @WITHATTR are member set functions. Most member set functions can be used in filter definitions.

For example, assume that user PJones is assigned this filter: Access: None. Member specification: @ATTRIBUTE(“Caffeinated_False”).

The next time user PJones connects to Sample.Basic, his spreadsheet view of first-quarter cola sales in California shows that he has no access to the data values for any base dimension members associated with Caffeinated_False. Sales data for Caffeine Free Cola is blocked from view. Note that Caffeine Free Cola is a base member, and Caffeinated_False is an associated member of the attribute dimension Caffeinated (not shown in the above spreadsheet view).

Figure 31-4 Results of Filter Blocking Access to Caffeine-free Products

	Sales	California	Qtr1	Actual
Cola	1998			
Diet Cola	367			
Caffeine Free Cola	#Miss			
Colas	2767			

Metadata Filtering

Metadata filtering provides data filtering and an additional layer of security. With metadata filtering, an administrator can remove outline members from a user's view, providing access only to those members that are of interest to the user.

When a filter is used to apply MetaRead permission on a member:

1. Data for all ancestors of that member are hidden from the filter user's view.
2. Data and metadata (member names) for all siblings of that member are hidden from the filter user's view.

Dynamic Filtering

You can create dynamic filters based on external source data to reduce the number of filter definitions needed. You do this using dynamic filter definition syntax, including the method `@datasourceLookup` and the variables `$LoginUser` and `$LoginGroup`.

For details, see Create Efficient Dynamic Filters.

Managing Filters

To view a list of filters, you can use the **display filter** MaxL statement.

To edit, copy, or rename a filter, you can use the **create filter** MaxL statement.

You can copy filters to applications and databases on any Essbase Server, according to your permissions. You can also copy filters across servers as part of application migration.

To delete a filter, you can use the **drop filter** MaxL statement.

Assigning Filters

After you define filters, you can assign them to users or groups, which lets you manage multiple users who require the same filter settings. Modifications to the definition of a filter are automatically inherited by users of that filter.

Filters do not affect users who have the Administrator role.

You can assign filters in MaxL or in the Essbase web interface.

> **Note:**
>
> To assign filters, roles must be assigned at the application level. In MaxL, the application-level user and group roles are assigned implicitly, whereas in the Essbase web interface an explicit application-level role assignment is required before you can assign filters.

Assigning Filters Using MaxL

You can assign filters in MaxL using the grant MaxL statement.

Example

1. Create a filter called filter1.

```
MAXL> create filter sample.basic.filter1 read on 'Jan, sales', no_access on
        '@children(Qtr2)';
```

2. Grant filter access to user1.

```
MAXL> grant filter sample.basic.filter1 to user1;
```

Assigning Filters Using the Essbase Web Interface

To assign filters using the Essbase web interface, first make role assignments at the application level, and then assign the filters.

Example

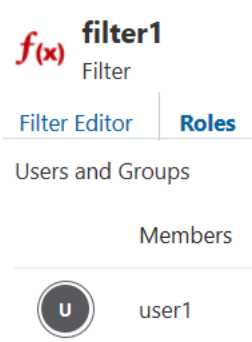
- Redwood
- Classic

Redwood

1. Assign roles at the application level:
   a. On the Applications page, open the application.
   b. Select the **Customization** page.
   c. Select the **Permissions** tab.
   d. Click **Add**.
      A list of users and groups is displayed.
   e. Click Add + next to a user.
      In this topic, we use **user1**, a user with the User role, as our example.
   f. Close the right hand panel by clicking Close ×.
2. Create a filter called filter1 with read access on 'Jan, sales', no access on @children(Qtr2).
3. Grant filter access to user1:
   a. On the Applications page, open the application.
   b. Open the cube.
   c. Select the **Customization** page.
   d. Select the **Filters** tab.
   e. Select **filter1**.
   f. Select the **Roles** tab.
   g. Click **Add**.
      A list of users is displayed.
   h. Click Add + next to a **user1**.
   i. Click Close ×.
      user1 is displayed as a member of **filter1**.

Classic

1. Assign roles at the application level:
   a. On the Applications page, click the **Actions** menu to the right of the application name.
   b. Select **Inspect**, and then select **Permissions**.
   c. Click Add on the right hand side of the dialog box.
   A list of users and groups is displayed.
   d. Click Add next to a user.
   In this topic, we use **user1**, a user with the User role, as our example.
   e. Close the right hand panel by clicking Close.
   f. To check your filter assignments, on the filter editor screen, click **Roles**.
   user1 is displayed.
2. Create a filter called filter1 with read access on 'Jan, sales', no access on @children(Qtr2).
3. Grant filter access to user1:
   a. On the Applications page, expand the application and click the **Actions** menu to the right of the cube name.
   b. Select **Inspect**, and then select **Filters**.
   c. Select **filter1** and then select **Roles**.
   d. Click Add.
   A list of users and groups is displayed.
   e. Click the Add icon next to **user1**.
   f. Click **Close**.
   user1 is displayed as a member of **filter1**.



Overlapping Filter Definitions

If a filter contains rows that have overlapping member specifications, the inherited access is set by the following rules, listed in order of precedence:

1. A filter that defines a more detailed dimension combination list takes precedence over a filter with less detail.

2. If the preceding rule does not resolve the overlap conflict, the highest access level among overlapping filter rows is applied.

For example, this filter contains overlap conflicts:

- Access: Write. Member specification: Actual.
- Access: None. Member specification: Actual.
- Access: Read. Member specification: Actual, @IDESCENDANTS(“New York”).

The third specification defines security at a greater level of detail than the other two. Therefore, read access is granted to all Actual data for members in the New York branch.

Because write access is a higher access level than none, the remaining data values in Actual are granted write access.

All other cells, such as Budget, are accessible according to the minimum database permissions.

If you have write access, you also have read access.

> **Note:**
>
> Changes to members in the database outline are not reflected automatically in filters. You must manually update member references that change.

Overlapping Metadata Filter Definitions

You should define a MetaRead filter using multiple rows only when the affected member set in any given row (the metaread members and their ancestors) has no overlap with MetaRead members in other rows. It is recommended that you specify one dimension per row in filters that contain MetaRead on multiple rows. However, as long as there is no overlap between the ancestors and MetaRead members, it is still valid to specify different member sets of one dimension into multiple MetaRead rows.

For example, in Sample Basic, the following filter definition has overlap conflicts:

- Access: MetaRead. Member specification: California.
- Access: MetaRead. Member specification: West.

In the first row, applying MetaRead to California has the effect of allowing access to California but blocking access to its ancestors. Therefore, the MetaRead access to West is ignored; users who are assigned this filter will have no access to West.

If you wish to assign MetaRead access to West, as well as California, the appropriate method is to combine them into one row: Access: MetaRead. Member specification: California, West.

Overlapping Access Definitions

When the access rights of user and group definitions overlap, the following rules, listed in order of precedence, apply:

1. An access level that defines a more detailed dimension combination list takes precedence over a level with less detail.
2. If the preceding rule does not resolve the overlap conflict, the highest access level is applied.

Example 1:

User Fred is defined with the following database access:

```
FINPLAN     R
CAPPLAN     W
PRODPLAN    N
```

He is assigned to Group Marketing, which has the following database access:

```
FINPLAN     N
CAPPLAN     N
PRODPLAN    W
```

His effective rights are set as:

```
FINPLAN     R
CAPPLAN     W
PRODPLAN    W
```

Example 2:

User Mary is defined with the following database access:

```
FINPLAN     R
PRODPLAN    N
```

She is assigned to Group Marketing, which has the following database access:

```
FINPLAN     N
PRODPLAN    W
```

Her effective rights are set as:

```
FINPLAN     R
PRODPLAN    W
```

In addition, Mary uses the filter artifact RED (for the database FINPLAN). The filter has two filter rows:

- Access: Read. Member specification: Actual.
- Access: Write. Member specification: Budget, @IDESCENDANTS(“New York”).

The Group Marketing also uses a filter artifact BLUE (for the database FINPLAN). The filter has two filter rows:

- Access: Read. Member specification: Actual, Sales.
- Access: Write. Member specification: Budget, Sales.

Mary’s effective rights from the overlapping filters, and the permissions assigned to her and her group:

- R: Entire FINPLAN database.

- W: For all Budget data in the New York branch.
- W: For data values that relate to Budget and Sales.

32

Using MaxL Data Definition Language

Using MaxL, you can automate administrative operations on Essbase databases. The MaxL data definition language is an interface for making administrative requests to Essbase using statements.

See the following resources:

- Managing Essbase Using the MaxL Client
- MaxL

33 Optimizing Database Restructuring

Some changes to an Essbase block-storage database outline affect the data storage arrangement, forcing a restructuring of the database. Because changes that require restructuring the database can be time-consuming (unless you discard the data before restructuring), consider deciding on such changes based on how they affect performance.

- Database Restructuring
- Optimization of Restructure Operations
- Actions That Improve Performance
- Outline Change Quick Reference

The information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases.

Also see Positioning Dimensions and Members.

Database Restructuring

As your business changes, you change the Essbase database outline to capture new product lines, provide information on new scenarios, reflect new time periods, and so on. Some changes to a database outline affect the data storage arrangement, forcing Essbase to restructure the database.

Because changes that require restructuring the database are time-consuming (unless you discard the data before restructuring), consider deciding on such changes based on how they affect performance. See:

- Implicit Restructures
- Explicit Restructures
- Conditions Affecting Database Restructuring
- Restructuring Requires a Temporary Increase in the Index and Data Cache Sizes

> **Note:**
>
> For information about clearing data and thus avoiding some restructuring, see CLEARDATA and CLEARBLOCK calculation commands.

Implicit Restructures

Essbase initiates an implicit restructure of the database files after an outline is changed using Outline Editor or Dimension Build. The type of restructure that is performed depends on the type of changes made to the outline:

- **Dense restructure:** If a member of a dense dimension is moved, deleted, or added, Essbase restructures the blocks in the data files and creates new data files. When

Essbase restructures the data blocks, it regenerates the index automatically so that index entries point to the new data blocks. Empty blocks are not removed. Essbase marks all restructured blocks as dirty, so after a dense restructure you must recalculate the database. Dense restructuring, the most time-consuming of the restructures, can take a long time to complete for large databases.

- **Sparse restructure:** If a member of a sparse dimension is moved, deleted, or added, Essbase restructures the index and creates new index files. Restructuring the index is relatively fast; the time required depends on the index size.
- **Outline-only restructure:** If a change affects only the database outline, Essbase does not restructure the index or data files. Member name changes, creation of aliases, and dynamic calculation formula changes are examples of changes that affect only the database outline.

> **Note:**
>
> How a database outline is changed (by using Outline Editor or using dimension build) does not influence restructuring. Only the type of information change influences what type of restructuring, if any, takes place. For information about outline changes and the type of restructures they cause, see Outline Change Quick Reference.

Explicit Restructures

When you manually initiate a database restructure, you perform an explicit restructure. An explicit restructure forces a full restructure of the database. A full restructure comprises a dense restructure plus removal of empty blocks. All data load and calculation transaction history is removed after an explicit restructure.

To initiate an full restructure, you can use the MaxL statement alter database **force restructure**.

Conditions Affecting Database Restructuring

Intelligent Calculation, name changes, and formula changes affect database restructuring:

- If you use Intelligent Calculation in the database, all restructured blocks are marked as dirty whenever data blocks are restructured. Marking the blocks as dirty forces the next default Intelligent Calculation to be a full calculation.
- If you change a name or a formula, Essbase does not mark the effected blocks as dirty. If intelligent calculation is on, default calculations do not recalculate these effected members. You must explicitly re-calculate them.

Use the following topics for information about restructuring:

Table 33-1 Topics Related To Database Restructuring

Topic	Related Information
Intelligent Calculation	Restructuring Databases: Impact on Block Status
Sparse and dense dimensions	• Sparse and Dense Dimensions, Selection of Dense and Sparse Dimensions • Dense and Sparse Selection Scenarios
Attribute dimensions	Designing Attribute Dimensions

Table 33-1 (Cont.) Topics Related To Database Restructuring

Topic	Related Information
Dimension building	Overview of Data Load and Dimension Build
Outline Editor	Creating and Changing Database Outlines

Restructuring Requires a Temporary Increase in the Index and Data Cache Sizes

Restructuring an Essbase cube results in a temporary increase (double at most) of the index and data cache sizes. When the restructure is completed, both the index and data cache return to the sizes they were before the restructure was performed.

Optimization of Restructure Operations

If a database outline changes frequently, analyze the outline and the types of changes that you are making. Changes to sparse dimensions or attribute dimensions are relatively fast, because only the index changes. Changes to dense dimensions are relatively slow, because data blocks are rebuilt.

These types of restructure operations are listed from fastest to slowest:

- Outline only (no index or data files)
- Sparse (only index files)
- Dense (index files and data files) as a result of adding, deleting, or moving members and other operations. See Outline Change Quick Reference
- Dense (index and data files) as a result of changing a dense dimension to sparse or changing a sparse dimension to dense

Actions That Improve Performance

Several actions improve performance related to database restructuring:

- If you change a dimension frequently, make it sparse.
- Use incremental restructuring to control when Essbase performs a required database restructuring.
- Consider using parallel restructuring on multiprocessor systems.
- Select options when you save a modified outline that reduce the amount of restructuring required.

Parallel Restructuring

By default, block storage restructuring is performed sequentially. Blocks are renumbered and reshaped from first to last, a time-intensive process. Parallel restructuring can reduce restructuring time by dividing block restructuring work across multiple concurrent threads to use available processor cores. Because calculation is performed separately from restructuring, each block can be restructured independently of other blocks.

Blocks are divided into *n* groups, where *n* is the number of restructuring threads. This division is performed by traversing the breadth of the index BTree until the number of keys at a level is equal to or greater than *n*. If there is no level with more than *n* keys, the number of restructuring threads is reduced accordingly.

The number of restructuring threads to use is defined using the RESTRUCTURETHREADS configuration setting. If RESTRUCTURETHREADS is not defined, the default is one thread.

Options for Saving a Modified Outline

Essbase displays a dialog box when you save outline changes that trigger database restructuring (using Outline Editor). In the Restructure Database dialog box, you define how data values are handled during restructure; for example, you can preserve all data, preserve only level 0 or input data, or discard all data during restructure.

If the database contains data, you need enough free disk space on the server to create a backup copy of the database. Backup ensures that any abnormal termination during the restructure process does not corrupt the database.

Essbase may display a message that restructuring is not required, yet still perform an index-only restructure. This event most likely occurs if you make changes to a sparse dimension. If you try to cancel a restructure operation, Essbase may issue a message indicating that the operation cannot be canceled. If such a message is displayed, Essbase is performing final cleanup, and it is too late to cancel.

Outline Change Quick Reference

The tables in this section show all outline changes that affect calculation and restructuring, including incremental restructuring.

> **Note:**
>
> If you are using Partitioning, restructuring affects only the database to which you are connected.

Table 33-2 Actions: Delete, Add, or Move Member

Action	Calculation and Standard Restructure Effects	Incremental Restructuring Applies? (If Enabled)
Delete member of sparse dimension	Data must be recalculated to reflect changes to relationships. Essbase deletes from the index file all pointers to blocks represented by the deleted member. Because the blocks are no longer pointed to, they become free space. No restructure.	For regular members, no. Essbase restructures the index, overriding incremental restructure. For label-only members, yes, restructuring is deferred.
Delete member of attribute dimension	None	No
Delete member of dense dimension	Data must be recalculated to reflect changes to relationships. Essbase restructures the data files to reflect a changed block size. Essbase restructures the index.	Yes. Changes to data and index files are deferred.

Table 33-2 (Cont.) Actions: Delete, Add, or Move Member

Action	Calculation and Standard Restructure Effects	Incremental Restructuring Applies? (If Enabled)
Delete shared member in sparse or dense dimension	Data must be recalculated. The data remains associated with the prototype member name, but, because the parent of the shared member may have depended on child data, recalculation is needed. No restructure.	No
Add member to sparse dimension	Data for the new member must be loaded or calculated to derive new values. Essbase restructures the index.	Yes. Changes to data and index files are deferred.
Add member to dense dimension	Data for the new member must be loaded or calculated to derive new values. Data must be recalculated. Essbase restructures the data files to reflect a changed block size. Essbase restructures the index.	Yes. Changes to data and index files are deferred.
Add member to attribute dimension	None	No
Add shared member to sparse or dense dimension	Data must be recalculated. The new shared member affects the consolidation to its parent. No restructure.	No
Move regular member within a sparse dimension	Data must be recalculated to reflect changes in consolidation. Essbase restructures the index file.	No. Essbase restructures the index file, overriding incremental restructure.
Move regular member within a dense dimension	Data must be recalculated to reflect changes in consolidation. Essbase restructures index and data files.	Yes. Changes to data and index files are deferred.
Move an attribute dimension member	None	No

Table 33-3 Actions: Other Member-Related Changes

Action	Calculation and Standard Restructure Effects	Incremental Restructuring Applies? (If Enabled)
Change a member alias or add an alias to a member	None	No
Rename member	None	No
Change member formula	Data must be recalculated to reflect formula changes. No restructure.	No

Table 33-4 Actions: Dynamic Calculation-Related Changes

Action	Calculation and Standard Restructure Effects	Incremental Restructuring Applies? (If Enabled)
Define regular dense dimension member as Dynamic Calc	Essbase restructures both index and data files.	Restructure deferred.
Define sparse dimension Dynamic Calc member as regular member	No restructure	No
Define sparse dimension regular member as Dynamic Calc	Essbase restructures index and data files.	Yes. Changes to data and index files are deferred.
Define dense dimension Dynamic Calc member as regular member	Essbase restructures index and data files.	Yes. Changes to data and index files are deferred.
Define dense dimension regular member as Dynamic Calc member	Essbase restructures index and data files.	Yes. Changes to data and index files are deferred.
Add, delete, or move sparse dimension Dynamic Calc member	Essbase restructures index files.	For member add or delete, changes to data and index files are deferred. For member move, Essbase restructures index files, overriding incremental restructure.
Add, delete, or move dense dimension Dynamic Calc member	No restructure.	No

Table 33-5 Actions: Property and Other Changes

Action	Calculation and Standard Restructure Effects	Incremental Restructuring Applies? (If Enabled)
Change dense-sparse property	Data must be recalculated. Essbase restructures both index and data files.	Essbase restructures index and data files overriding incremental restructure.
Change label only property	Data must be recalculated. Essbase restructures index and data files.	Yes. Changes to data and index files are deferred.
Change shared member property	Data must be recalculated to reflect the changed data value of the child. Essbase restructures both index and data files.	Yes. Changes to data and index files are deferred.
Change properties other than dense-sparse, label, or shared	Data may need to be recalculated to reflect changed consolidation properties, such as changing time balance from first to last.	No
Change the order of two sparse dimensions	No calculation or data load impact. Essbase restructures the index.	Essbase restructures the index, overriding incremental restructure.
Change the order of dimensions	Data must be recalculated. Essbase restructures both index and data files.	Essbase restructures index and data files (overrides incremental restructure).
Change the order of attribute dimensions	None	No
Create, delete, clear, rename, or copy an alias table	None	No

Table 33-5 (Cont.) Actions: Property and Other Changes

Action	Calculation and Standard Restructure Effects	Incremental Restructuring Applies? (If Enabled)
Import an alias table or set a member alias	None	No
Change the case-sensitive setting	None	No
Name a level and generation	None	No
Create, change, or delete a UDA	None	No

34

Optimizing Data Loads

You can improve the performance of data loads to Essbase databases by minimizing the time spent on reading / parsing the data source and writing to the cube.

- Understanding Data Loads
- Grouping Sparse Member Combinations
- Making the Data Source as Small as Possible
- Making Source Fields as Small as Possible
- Positioning Data in the Same Order as the Outline
- Loading from Essbase Server
- Using Parallel Data Load

Some information in this chapter applies only to block storage databases and is not relevant to aggregate storage databases. Also see Comparison of Aggregate and Block Storage.

Understanding Data Loads

You can improve the performance of data loads to Essbase block storage cubes when you understand the stages of the data load process and learn how to optimize and parallelize the tasks.

This section does not apply to aggregate storage databases.

Loading a large data source into an Essbase database can take a long time. You can shorten the data loading process by minimizing the time spent on these actions:

- Reading and parsing the data source
- Reading and writing to the database

To optimize data load performance, think in terms of database structure. Essbase loads data block by block. For each unique combination of sparse dimension members, one data block contains the data for all the dense dimension combinations, assuming that at least one cell contains data. For faster access to block locations, Essbase uses an *index*. Each entry in the index corresponds to one data block. See Sparse and Dense Dimensions, Selection of Dense and Sparse Dimensions, and Dense and Sparse Selection Scenarios.

When Essbase loads a data source, Essbase processes the data in five stages of a pipeline.

For free form data load, the stages are:

1. **Input**—Essbase collects input from file or SQL connection
2. **Tokenize**—Essbase separates input fields from records, creating tokens
3. **Convert**—Essbase converts tokens into member items
4. **Preparation**—Essbase arranges the data in preparation for putting it into blocks
5. **Write**—Essbase puts the data into blocks in memory and then writes the blocks to disk, finding the correct block on the disk by using the index, which is composed of pointers based on sparse intersections

For rules-file based data load, the stages are:

1. **Input**—Essbase collects input from file or SQL connection
2. **Pre-Rule**—Essbase reads data load records
3. **Rule**—Essbase applies rules, embedded in rules file, to data load records
4. **Preparation**—Essbase arranges the data in preparation for putting it into blocks
5. **Write**—Essbase puts the data into blocks in memory and then writes the blocks to disk, finding the correct block on the disk by using the index, which is composed of pointers based on sparse intersections

> **Note:**
>
> On aggregate storage databases, the fifth stage does not apply.

This process is repeated until all data is loaded. By using one or more processing threads in each stage, Essbase can perform some processes in parallel. See Using Parallel Data Load.

Examples in this chapter assume that you are familiar with the information in this topic: Sources of Data.

Grouping Sparse Member Combinations

To improve performance of Essbase block storage data load, you can structure the data to minimize the amount of necessary disk I/O (reads and writes).

This section does not apply to aggregate storage cubes.

The most effective strategy to improve performance is to minimize the number of disk I/Os that Essbase must perform while reading or writing to the cube. Because Essbase loads data block by block, organizing the source data to correspond to the physical block organization reduces the number of physical disk I/Os that Essbase must perform.

Arrange the source data so that records with the same unique combination of sparse dimensions are grouped together. This arrangement corresponds to blocks in the database.

The examples in this chapter illustrate ways that you can organize the data following this strategy. These examples use a subset of the Sample.Basic cube, as described below:

Table 34-1 Dimensions and Values for Examples

Sparse, Nonattribute Dimensions	Dense Dimensions
Scenario (Budget, Actual)	Measures (Sales, Margin, COG, Profit)
Product (Cola, Root Beer)	Year (Jan, Feb)
Market (Florida, Ohio)	

> **Note:**
>
> Because you do not load data into attribute dimensions, they are not relevant to this discussion although they are sparse.

Consider the following source of data. Because it is not grouped by sparse-dimension member combinations, this data has not been sorted for optimization. As Essbase reads each record, it must deal with different members of the sparse dimensions.

```
Jan
Actual     Cola          Ohio      Sales    25
Budget     "Root Beer"   Florida   Sales    28
Actual     "Root Beer"   Ohio      Sales    18
Budget     Cola          Florida   Sales    30
```

This data loads slowly because Essbase accesses four blocks instead of one.

An optimally organized source of data for the same Sample.Basic cube shows different records sorted by a unique combination of sparse-dimension members: Actual -> Cola -> Ohio. Essbase accesses only one block to load these records.

```
Actual      Cola    Ohio    Jan   Sales     25
Actual      Cola    Ohio    Jan   Margin    18
Actual      Cola    Ohio    Jan   COGS      20
Actual      Cola    Ohio    Jan   Profit     5
```

You can use a source that loads many cells per record. Ensure that records are grouped together by unique sparse-dimension member combinations. Then order the records so that the dimension in the record for which you provide multiple values is a dense dimension.

The next example uses a header record to identify the members of the Measures dimension, which is dense. The data is sorted first by members of the dense dimension Year and grouped hierarchically by members of the other dimensions. Multiple values for the Measures dimension are provided on each record.

```
                                 Sales  Margin   COG  Profit
Jan Actual  Cola         Ohio       25      18    20       5
Jan Actual  Cola         Florida    30      19    20      10
Jan Actual  "Root Beer"  Ohio       18      12    10       8
Jan Actual  "Root Beer"  Florida    28      18    20       8
```

Notice that the heading and first data line that requires two lines in this example; the previous example needs four lines for the same data.

Making the Data Source as Small as Possible

To improve performance of Essbase block storage data load, you can make the source data as small as possible and group it into ranges to remove redundancy.

Make the data source as small as possible. The fewer fields that Essbase reads in the data source, the less time is needed to read and load the data.

Group the data into ranges. Eliminating redundancy in the data source reduces the number of fields that Essbase must read before loading data values.

The following example data source is not organized in ranges. It includes unneeded repetition of fields. All values are Profit values. Profit must be included only at the beginning of the group of data applicable to it. This example contains 33 fields that Essbase must read to load the data values properly.

```
Profit
Jan     "New York"    Cola           4
Jan     "New York"    "Diet Cola"    3
Jan     Ohio          Cola           8
Jan     Ohio          "Diet Cola"    7
Feb     "New York"    Cola           6
Feb     "New York"    "Diet Cola"    8
Feb     Ohio          Cola           7
Feb     Ohio          "Diet Cola"    9
```

The next example provides the same data optimized by grouping members in ranges. By eliminating redundancy, this example contains only 23 fields that Essbase must read in order to load the data values properly.

```
Profit
Jan   "New York"    Cola           4
                    "Diet Cola"    3
      Ohio          Cola           8
                    "Diet Cola"    7
Feb   "New York"    Cola           6
                    "Diet Cola"    8
      Ohio          Cola           7
                    "Diet Cola"    9
```

Essbase assigns the first value, 4, to Jan->New York->Cola; it assigns the next value, 3, to Jan->New York->Diet Cola and so on.

Although sorted efficiently, the data source sorted and grouped by dense dimensions shows a lot of repetition that can slow down the load process. You can further optimize this data by grouping the data into ranges. The optimized data source below eliminates the redundant fields, reducing processing time.

```
                                  Sales  Margin   COG  Profit
Jan Actual  Cola         Ohio       25      18     20       5
                         Florida    30      19     20      10
            "Root Beer"  Ohio       18      12     10       8
                         Florida    28      18     20       8
```

Making Source Fields as Small as Possible

To improve performance of Essbase block storage data load, you can make fields in the source data as small as possible.

Making fields in a data source smaller enables Essbase to read and load faster.

Make the fields in the data source as small as possible by performing the following tasks:

- Remove excess white space in the data source. For example, use tabs instead of blank spaces.
- Round computer-generated numbers to the precision you need. For example, if the data value has nine decimal points and you care about two, round the number to two decimal points.
- Use #MI instead of #MISSING.

Positioning Data in the Same Order as the Outline

To improve performance of Essbase block storage data load, you can make the source data follow the same order as the order of sparse dimensions in the outline.

This section does not apply to aggregate storage databases.

The index is organized in the same order as the sparse dimensions in the outline. To further optimize the data source, with the sparse data combinations in the data source grouped together, arrange the data so that sparse dimensions are in the same order as the outline.

Essbase pages portions of the index in and out of memory as requested by the data load or other operations. Arranging the source data to match the order of entries in the index speeds the data load because it requires less paging of the index. Less paging results in fewer I/O operations.

The index cache size is used to determine how much of the index can be paged into memory. Adjusting the size of the index cache may also improve data load performance.

> **Note:**
>
> If the index cache size is large enough to hold the entire index in memory, positioning data in the same order as the outline does not affect the speed of data loads.

Loading from Essbase Server

Loading the data source from Essbase Server is faster than loading from a client computer. To load a data source from the server, move the data source to the server and start the load.

Loading data from the server improves performance because the data need not be transported over the network from the client computer to the server computer.

Using Parallel Data Load

The following topics discuss parallel data load and how it might improve performance for your site.

Understanding Parallel Data Load

One aspect of parallel data load describes the pipeline optimization you can achieve by using the configuration settings DLTHREADSPREPARE and DLTHREADSWRITE. While the minimum and default number of threads allocated for a data load is 5 (one thread per stage of the pipeline), these settings enable you to add threads to selected stages in the pipeline. For example, with the following configuration, you can increase the threads used in the Prepare and Write stages from 1 each to 4 each:

```
DLSINGLETHREADPERSTAGE Sample Basic FALSE
DLTHREADSPREPARE Sample Basic 4
DLTHREADSWRITE Sample Basic 4
```

With the above configuration, the data load is set to run with 11 threads.

Another aspect of parallel data load refers to the concurrent loading of multiple data files into an Essbase database. When working with large data sets (for example, a set of ten 2 GB files), loading the data sources concurrently enables you to fully utilize the CPU resources and I/O channels of modern servers with multiple processors and high-performance storage subsystems.

You can also adjust the number of threads used in multiple-file data loads. For example, specifying the above configuration while also specifying two data files results in the creation of two data load pipelines, each having 11 threads.

Enabling Parallel Data Load With Multiple Files

To enable parallel data load, specify multiple files as the data source, by using a wildcard character (`*` and/or `?`) to match all data sources files you intend to use. See the **import data** MaxL statement. If necessary, control the number of threads spawned by the parallel data load, using the **using max_threads** grammar in the MaxL statement.

35 Block Storage Calculation Optimization

You can configure an Essbase block storage cube to optimize calculation performance.

The information in this chapter applies only to block storage cubes, and is not relevant to aggregate storage cubes.

Design for Calculation Performance

To optimize calculation performance for an Essbase block storage cube, consider the impact of data block size and density, the efficiency of formulas and calc scripts, and ways you can tune outline characteristics (such as order of sparse dimensions, use of levels, and the Time dimension).

You can configure a database to optimize calculation performance.

The best configuration for the site depends on the nature and size of the database. Use the information in the following topics as guidelines only.

Block Size and Block Density

For Essbase block storage cubes to have optimal calculation performance, learn how the data block size and block density can impact performance and storage. Review the block statistics by using the Essbase web interface or MaxL.

A data block size of 8 Kb to 100 Kb provides optimal performance in most cases.

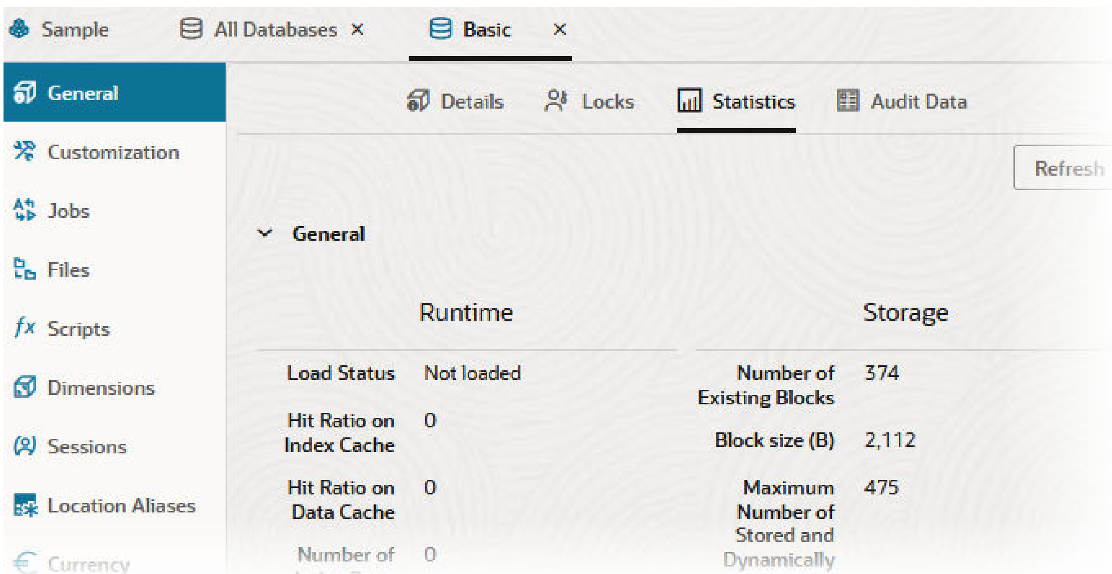
If data blocks are much smaller than 8 KB, the index is usually very large, forcing Essbase to write to and retrieve the index from disk. This process slows calculation.

If data blocks are much larger than 100 KB, Intelligent Calculation does not work effectively. Refer to Intelligent Calculation for Block Storage Cubes.

To optimize calculation performance and data storage, you may need to balance data block density and data block size by rearranging the dense and sparse dimension configuration of the database. Keep these suggestions in mind:

- Keep data block size between 8 KB and 100 KB with as high a block density as possible.
- Run test calculations of the most promising configurations of a cube that contains representative data. Check results to determine the configuration that produces the best calculation performance.
- Consider using hybrid mode. In a hybrid mode cube, you can easily customize the solve order instead of rearranging dimensional order.

View information about a cube, including the potential and actual number of data blocks and the data block size, by using the Essbase web interface or MaxL.



The equivalent statement for viewing block statistics using MaxL is:

```
query database appname.dbname get dbstats data_block;
```

Order of Sparse Dimensions

In Essbase block storage cubes, calculation performance may be improved by placing smaller sparse dimensions first in the outline order.

Alternatively, consider using hybrid mode. In a hybrid mode cube, you can easily customize the solve order instead of rearranging dimensional order.

You may improve calculation performance by changing the order of standard (non attribute) sparse dimensions in the outline. Order standard sparse dimensions by the number of members they contain, placing the dimension that contains the fewest members first. This arrangement can improve:

- The function of the calculator cache can have ~10% performance improvement for an outline with a large dimension (for example, one containing 1000 members).

- Parallel calculation, if enabled, more likely will be used if the standard sparse dimension with the most members is the last standard sparse dimension in the outline.

Incrementally Loaded Cubes

For Essbase block storage cubes that are incrementally (periodically) loaded with data, calculation performance may be improved by making the Time dimension sparse in the outline.

Many companies load data incrementally. For example, a company may load data each month for that month.

To optimize calculation performance when you load data incrementally, make the dimension tagged as time a sparse dimension. If the time dimension is sparse, the database contains a data block for each time period. When you load data by time period, Essbase accesses fewer data blocks because fewer blocks contain the relevant time period. Thus, if you have Intelligent Calculation enabled, only the data blocks marked as dirty are recalculated. For example, if you load data for March, only the data blocks for March and the dependent parents of March are updated.

However, making the time dimension sparse when it is naturally dense may significantly increase the size of the index, creating possibly slower performance due to more physical I/O activity to accommodate the large index.

If the dimension tagged as time is dense, you still receive some benefit from Intelligent Calculation when you do a partial data load for a sparse dimension. For example, if Product is sparse and you load data for one product, Essbase recalculates only the blocks affected by the partial load, although time is dense and Intelligent Calculation is enabled.

Multiple Flat Dimensions

For Essbase block storage cubes that have thousands of members in flat dimensions, calculation performance may be improved by adding levels in the outline.

Calculation performance may be affected if a cube outline has multiple flat dimensions. A flat dimension has very few parents, and each parent has many thousands of children; in other words, flat dimensions have many members and few levels.

You can improve performance for outlines with multiple flat dimensions by adding intermediate levels to the outline.

Formulas and Calculation Scripts

To optimize calculation scripts and formulas for block storage cubes, take steps to reduce the number of cycles Essbase makes through data blocks.

You may achieve significant improvements in calculation performance by carefully grouping formulas and dimensions in a calculation script. In this way, you can ensure that Essbase cycles through the data blocks in the cube as few times as possible during a calculation.

Order commands in calculation scripts to make the database calculation as simple as possible. Consider applying all formulas to the outline and using a default calculation (CALC ALL).

See Develop Calculation Scripts for Block Storage Cubes and Calculation Passes.

Monitor and Trace Calculations

Monitor and trace Essbase calculations to gain insight into member formula processing. CALCTRACE and SET TRACE help you trace calculations run from Smart View or in stored calculations. To log performance statistics for testing, you can use SET MSG and SET NOTICE.

To enable calc tracing, use the CALCTRACE configuration property, and then use Smart View to begin context sensitive tracing, or use the SET TRACE command in a calculation script to select which data intersections to trace. Refer to Trace Calculations for details.

To display information in the application log about how Essbase is calculating the cube, you can use SET MSG and SET NOTICE in a calculation script. Refer to SET Commands for Calc Testing and Statistics.

Trace Calculations

Use Essbase calculation tracing to gain insight into member formula processing, helping you debug and refine your block storage calculation scripts. Enable CALCTRACE for context sensitive Smart View calc tracing, or use SET TRACE command to select data intersections to trace.

Calculation tracing enables you to access logged information about a calculation, after the calculation script successfully executes against a cube.

Tracing a calculation does not change anything about calculation behavior. If a calculation is launched in Smart View, and the connected server has calculation tracing enabled by an administrator, Smart View displays a pop-up dialog box containing details, after the calculation runs. The calculation tracing information can be pasted from the pop-up dialog into a text editor. Or, you can find the same information in `calc_trace.txt`, located in the database files directory in Essbase.

The calculation tracing information can help you debug calculation script execution, in case the results of the calculation are not what you expected.

Calculation tracing is not supported on applications with scenario management enabled.

To enable calculation tracing, the administrator must first turn on the CALCTRACE application configuration parameter. After calculation tracing is enabled for your application, there are two ways to take advantage of it:

- In Smart View, you can use context-sensitive tracing for a single cell value.
  1. In Smart View, connect a query sheet to the application for which you enabled calculation tracing.
  2. Highlight a data cell whose calculated value you would like to trace.
  3. In the Data panel of the Essbase tab, click the **Calculate** button and select a calculation script to execute. You will see the point-of-view from your highlighted data cell in the trace member runtime prompts.
  4. Click **Launch** to execute the calculation script.
     The full scope of the calculation as contained in the script will be calculated, but only the highlighted data cell context will be traced during the calculation.
  5. At the end of the calculation script, examine the **Calculation Result** dialog box, which shows the pre- and post-calculation results for your highlighted data cell.

If the highlighted data cell was not modified during the calculation, you will see a message indicating that the cell was not modified.

- In calculation scripts, you can use the SET TRACE calculation command to select data intersections to trace. SET TRACE enables you to trace multiple data cells. Additionally, you can trace sections of calculation scripts by using a combination of `SET TRACE` *`mbrList`* (to turn calculation tracing on over a member list) and `SET TRACE OFF` (to disable calculation tracing until a new SET TRACE is encountered in the script. To use SET TRACE command, you must execute the calculation script outside of Smart View, using Cube Designer, the CLI calc command, a Run Calculation job in the Essbase web interface, or MaxL (execute calculation statement).

```
SET TRACE ("100-10", "California", "Jan", "Sales", "Budget");
FIX("California", "Budget")
   "Sales" (
      "100-10" = @MEMBER(@CONCATENATE(@NAME(@PARENT("Product")), "-20")) / 10;
   );
ENDFIX;
```

Sample Basic has two sparse dimensions: Product and Market. The member formula is on Sales, a member of Measures, which is a dense dimension. The FIX statement's member list only contains one sparse member, California, which belongs to the Market dimension.

The number of existing blocks in the FIX scope determines the number of times the traced cell is calculated. In this example, the calculation cycles through all existing sparse member combinations of California. Each of these combinations represents a block.

After the calculation completes, the following tracing information is logged and displayed:

```
Tracing cell: [100-10][California][Jan][Sales][Budget]  (Cell update count: 1)
Previous value: 840.00
Dependent values:
    [100-20][California][Jan][Sales][Budget] = 140.00
New value: [100-10][California][Jan][Sales][Budget] = 14.00

Computed in lines: [91 - 93] using:
"Sales"(
"100-10"=@MEMBER(@CONCATENATE(@NAME(@PARENT("Product")),"-20"))/10;
)

Tracing cell: [100-10][California][Jan][Sales][Budget]  (Cell update count: 2)
Block from FIX scope: [100-30][California]
Actual block used in calculation: [100-10][California]
Previous value: 14.00
Dependent values:
    [100-20][California][Jan][Sales][Budget] = 140.00
New value: [100-10][California][Jan][Sales][Budget] = 14.00
Computed in lines: [91 - 93] using:
"Sales"(
"100-10"=@MEMBER(@CONCATENATE(@NAME(@PARENT("Product")),"-20"))/10;
)

Tracing cell: [100-10][California][Jan][Sales][Budget]  (Cell update count: 3)
Block from FIX scope: [200-10][California]
Actual block used in calculation: [100-10][California]
Previous value: 14.00
```

```
Dependent values:
    [200-20][California][Jan][Sales][Budget] = 520.00
New value: [100-10][California][Jan][Sales][Budget] = 52.00
Computed in lines: [91 - 93] using:
"Sales"(
"100-10"=@MEMBER(@CONCATENATE(@NAME(@PARENT("Product")),"-20"))/10;
)

[...calc iterations 4-7 are omitted from example...]

Tracing cell: [100-10][California][Jan][Sales][Budget]  (Cell update count: 8)
Block from FIX scope: [400-30][California]
Actual block used in calculation: [100-10][California]
Previous value: 9.00
Dependent values:
    [400-20][California][Jan][Sales][Budget] = 90.00
New value: [100-10][California][Jan][Sales][Budget] = 9.00
Computed in lines: [91 - 93] using:
"Sales"(
"100-10"=@MEMBER(@CONCATENATE(@NAME(@PARENT("Product")),"-20"))/10;
)
```

The calculation tracing log provides the following insights about how the calculation worked, on the cell that was traced:

- The traced cell was calculated several times, and the cell value was overwritten each time with the new value (the reported cell update count stops at 8).
- The value of the cell, before calculation, was 840.00.
- For each calculation occurrence, dependent values and new values are shown. Dependent values come from the member formula in the FIX statement.
- The final value of the traced cell, after all calculation completes, is 9, but it represents the value of product `"400-20"->California` divided by 10.
- Lines 91-93 of the calculation script, containing a member formula on Sales, are responsible for the updated values.

For each of the blocks cycled through, Sales is calculated using the formula:

```
"100-10"=@MEMBER(@CONCATENATE(@NAME(@PARENT("Product")),"-20"))/10
```

The formula contains a sparse member on the left hand side, which could cause the actual calculation block to be different than the initial FIX block. For example, when the calculation cycles through "California"->"100-20", the calculations are actually done in "California"->"100-10".

The trace log entries entitled `Block from FIX scope` and `Actual block used in calculation` are only printed if there is a discrepancy between the blocks in the FIX statement and the block that is represented in the member formula. These log entries can provide indications as to why there are duplicate calculations, helping you to debug your calculation scripts.

SET Commands for Calc Testing and Statistics

You can use SET commands in Essbase block storage calculation scripts to gather calculation information, settings, and statistics that can be useful for testing and monitoring calc performance. The commands are SET MSG SUMMARY, SET MSG DETAIL, and SET NOTICE.

SET MSG SUMMARY and SET MSG DETAIL

You can use the SET MSG SUMMARY and SET MSG DETAIL calculation commands in a calculation script to do the following:

- Display calculation settings, for example, whether completion notice messages are enabled
- Provide statistics on the number of data blocks created, read, and written
- Provide statistics on the number of data cells calculated

SET MSG DETAIL also provides an information message every time Essbase calculates a data block. SET MSG DETAIL is useful for reviewing the calculation order of data blocks and for testing intelligent recalculations.

> **Caution:**
>
> Because SET MSG DETAIL causes a high processing overhead, use it only during test calculations.

SET MSG SUMMARY causes a processing overhead of approximately 1% to 5%, depending on cube size, and is therefore appropriate for all calculations.

SET NOTICE

You can use the SET NOTICE calculation command in a calculation script to display calculation completion notices that tell you what percentage of the cube has been calculated. You can use the SET MSG SUMMARY command with the SET NOTICE command to show calculation progress between completion notices. Completion notices do not significantly reduce calculation performance, except when used with a very small cube.

Calculate Selected Tuples

By selecting tuples, you can focus your Essbase calculations in the active Smart View grid, limiting their scope to specific slices of data in your block storage cube.

The following sections describe tuple calculation:

- Use Case for Tuple Calculation
- Understand Tuple-Based Calculation
- Select Tuples for Point of View Calculation
- Examples of Tuple Selection to Reduce Calculation Scope

For the syntax for employing @GRIDTUPLES in a calculation script, see FIX…ENDFIX.

Use Case for Tuple Calculation

By selecting tuples, you can focus your Essbase calculations in the active Smart View grid, limiting their scope to specific slices of data in your block storage cube.

Tuple selection helps you optimize asymmetric grid calculations across dimensions, avoiding over-calculation.

Essbase calculation tuples differ from tuples used in MDX queries. Calculation performance and cube size are mainly driven by the number of blocks in the cube (given a specific block size). For this reason, calculation tuples are specified only for sparse member combinations. In addition, for ease of calculation scripting, multiple members from a single sparse dimension can be included in a calculation tuple specification. For example, if you specify ("New York", "California", "Actual", "Cola") as a calculation tuple, then you calculate the following cell intersections:

```
"New York"->"Actual"->"Cola"
"California"->"Actual"->"Cola"
```

Consider the following symmetric grid. It is symmetrical because each product has the same markets and scenario (Actual) represented in the grid.

		Profit	Inventory	Ratios
		Actual	Actual	Actual
		Jan	Jan	Jan
Cola	New York			
	Massachus			
	Florida			
	Connecticu			
	New Hamp			
Diet Cola	New York			
	Massachus			
	Florida			
	Connecticu			
	New Hamp			

The following grid is asymmetric, because the Diet Cola product has fewer markets in the grid than the Cola product has.

		Profit	Inventory	Ratios
		Actual	Actual	Actual
		Jan	Jan	Jan
Cola	New York			
	Massachus			
	Florida			
	Connecticu			
	New Hamp			
Diet Cola	New York			
	Florida			

The default calculation scope, when more than one dimension is in a FIX statement or a Smart View grid point of view (POV), is to calculate the cross product (all possible combinations) of the members in the FIX or grid. In other words, a POV-driven calculation in which product and market combinations are taken from the grid calculates all of these row-member combinations:

```
Cola->"New York"
Cola->"Massachusetts"
Cola->"Florida"
```

```
Cola->"Connecticut"
Cola->"New Hampshire"
"Diet Cola"->"New York"
"Diet Cola"->"Massachusetts"
"Diet Cola"->"Florida"
"Diet Cola"->"Connecticut"
"Diet Cola"->"New Hampshire"
```

This may be more calculation activity than you need. If you want to calculate *only* the combinations shown on the grid, you can specify which tuples to calculate, and limit the calculation to a smaller slice. Calculating tuples can also lower calculation time and cube size.

```
Cola->"New York"
Cola->"Massachusetts"
Cola->"Florida"
Cola->"Connecticut"
Cola->"New Hampshire"
"Diet Cola"->"New York"
"Diet Cola"->"Florida"
```

Understand Tuple-Based Calculation

A calculation **tuple** is a way to represent a data slice of members, from two or more sparse dimensions, to be used in an Essbase block storage calculation.

Examples of valid calculation tuples:

- `("Diet Cola", "New York")`
- `("Diet Cola", "Cola", Florida)`
- `(Cola, "New Hampshire")`

If you write MDX expressions, you might be aware of these tuple restrictions that apply to MDX:

- Only a single member from each dimension can be included in an MDX tuple
- All tuples in an MDX set must have the same dimensions represented, in the same order

However, when you select tuples in calculation scripts, these requirements are relaxed for convenience. You may freely write tuple expressions, and the tuples may describe member lists, as the following tuple does: `(@Children(East), Cola)`.

Select Tuples for Point of View Calculation

An easy way to select tuples is to insert them explicitly into a calculation script, as a list inside the FIX statement.

Recall that the format of a FIX statement is as follows:

```
FIX (fixMbrs)
COMMANDS ;
ENDFIX
```

In the FIX statement below, two tuples are specified before the command block begins. The tuples are enclosed within the curly braces { } that delimit a **set**, which is a collection of tuples.

```
FIX({
  (@Children(East), Cola),
  ("New York", Florida, "Diet Cola")
  })
Sales (Sales = Sales + 10;);
ENDFIX
```

Another way to select tuples is contextually, based on whichever members are present in a Smart View grid POV at calculation run time. You do this by providing the @GRIDTUPLES function as an argument to FIX, in your calculation script.

```
FIX ({@GRIDTUPLES(Product, Market)})
   Sales (Sales = Sales + 10;);
ENDFIX
```

If you execute this calculation script from Smart View against the grid below, then only the displayed combinations of products and markets are calculated. For example, "Diet Cola"->Massachusetts is not calculated, as it is not shown explicitly on the grid. Note that all scenarios (the third sparse dimension in this sample cube) are calculated, even though only Actual is shown on the grid. This is because the Scenario dimension is not part of the GRIDTUPLES statement in the calculation script.

		Profit	Inventory	Ratios
		Actual	Actual	Actual
		Jan	Jan	Jan
Cola	New York	[illegible]	[illegible]	[illegible]
	Massachus	[illegible]	[illegible]	[illegible]
	Florida	[illegible]	[illegible]	[illegible]
	Connecticu	[illegible]	[illegible]	[illegible]
	New Hamp	[illegible]	[illegible]	[illegible]
Diet Cola	New York	[illegible]	[illegible]	[illegible]
	Florida	[illegible]	[illegible]	[illegible]

Tuple selection, whether done using explicit lists of tuples or by using the @GRIDTUPLES function, is applicable only in the context of the FIX…ENDFIX calculation command. The syntax of the FIX statement is expanded to enable tuple selection:

```
FIX ([{ tupleList | @GRIDTUPLES(dimensionList) },] fixMbrs)
COMMANDS ;
ENDFIX
```

- *tupleList* - comma-separated set of tuples.
- *dimensionList* - at least two sparse dimensions whose members from the active Smart View grid are used to define the calculation regions. (In calculation scripts, you can use only sparse dimensions to define tuples.)
- *fixMbrs* - a member or list of members.

Examples of Tuple Selection to Reduce Calculation Scope

Using a Smart View grid and an Essbase calculation script FIX statement, you can calculate selected member tuples based on the grid point of view (POV). Alternatively, you can explicitly type the tuple combinations in your FIX statement, removing the dependency on a particular Smart View grid to define the calculation scope.

Calculating selected tuples helps you efficiently work with asymmetric regions in both calculation scripts and Smart View grids.

Consider the following examples:

- No Tuple Selection - Calculates in the default manner, based on current Smart View grid point-of-view (POV). The calculation is not limited to any specific tuples.
- Selection of Named Sparse Dimensions - Calculates tuples from two or more sparse dimensions named in a calculation script. The calculation is limited to members from the tuple dimensions that are present in the Smart View grid.
- Selection of Contextual Sparse Dimensions - Calculates tuples from sparse dimensions selected at run-time. The calculation is limited to members from the tuple dimensions present in the Smart View grid.

To try the examples, download the `CalcTuple_Tuple.xlsx` workbook template from the `Technical > Calc` section of the **gallery** folder in the **Files** area of the Essbase web interface. Refer to the README worksheet in the workbook for instructions.

No Tuple Selection

Demonstrating the default Essbase block storage calculation behavior that occurs when you do not select tuples, the following calculation script calculates the entire cross-product of Product and Market dimension members from a Smart View grid.

With the help of two runtime substitution variables (RTSV) defined in the SET RUNTIMESUBVARS block, calculation is limited to whichever Product and Market points of view are present in the grid when the calculation is run from Smart View.

```
SET RUNTIMESUBVARS
{
ProductGridMembers = POV
<RTSV_HINT><svLaunch>
<description>All Product's members on the grid</description>
<type>member</type>
<dimension>Product</dimension><choice>multiple</choice>
</svLaunch></RTSV_HINT>;
MarketGridMembers = POV
<RTSV_HINT><svLaunch>
<description>All Market's members on the grid</description>
<type>member</type> <dimension>Market</dimension><choice>multiple</choice>
</svLaunch></RTSV_HINT>;
};
FIX (
&ProductGridMembers, &MarketGridMembers
)
Marketing(
   Marketing = Marketing +1;
```

```
);
ENDFIX
```

Selection of Named Sparse Dimensions

Using the @GRIDTUPLES function to select the tuple of Product and Market dimensions, this Essbase block storage calculation script calculates tuples for only those two dimensions, limiting its scope to those members present in a Smart View grid at the time the calculation is executed from Smart View.

```
FIX (
{@GRIDTUPLES(Product, Market)}
)
Marketing(
   Marketing = Marketing + 1;
);
ENDFIX
```

By fixing on only the sparse dimensions named in the tuple, the calculation encompasses a much smaller number of blocks than a default calculation would. However, all members from dimensions not mentioned in the fix (Year, Scenario) are calculated by this calculation script.

Selection of Contextual Sparse Dimensions

Using the @GRIDTUPLES function and a runtime substitution variable, this Essbase block storage calculation script calculates only selected tuples from the grid, based on the sparse dimension selections in the RTSV prompt.

The runtime substitution variable *&DimSelections*, which is defined in the SET RUNTIMESUBVARS block, limits the calculation scope to only the sparse dimensions of the cube, excluding Scenario. The @GRIDTUPLES function used in the FIX statement calls this variable, limiting how many intersections are calculated.

```
SET RUNTIMESUBVARS
            {
            DimSelections = "Version", "Site", "Entity", "Product", "Market"
            <RTSV_HINT><svLaunch>
            <description>List two or more sparse dimensions used for forming
calculation tuples:</description>
            <type>string</type>
            </svLaunch></RTSV_HINT>;
            };
            FIX (
            {@GRIDTUPLES(&DimSelections)}
            )
            Marketing(
            Marketing = Marketing + 1;
            );
            ENDFIX
```

The calculation encompasses an even smaller number of blocks than the previous example, because in this case, the tuple definition extends to more sparse dimensions beyond Product->Market.

To try the examples, download the `CalcTuple_Tuple.xlsx` workbook template from the Technical > Calc section of the **gallery** folder in the **Files** area of the Essbase web interface. Refer to the README worksheet in the workbook for instructions.

Simulate Calculations to Estimate Calculation Time

You can simulate calculation on a model of your Essbase block storage cube to estimate how long a real calculation would take. To do this, you run a calculation script with SET MSG ONLY and SET NOTICE HIGH, take note of completion times, and extrapolate.

You can simulate a calculation using SET MSG ONLY in a calculation script. A simulated calculation produces results that help you analyze the performance of a real calculation that is based on the same data and outline.

By running a simulated calculation with a command such as SET NOTICE HIGH, you can mark the relative amount of time each sparse dimension takes to complete. Then, by performing a real calculation on one or more dimensions, you can estimate how long the full calculation will take, because the time a simulated calculation takes to run is proportional to the time that the actual calculation takes to run.

For example, if the calculation starts at 9:50:00 AM, and the first notice is time-stamped at 09:50:10 AM and the second is time-stamped at 09:50:20 AM, you know that each part of the calculation took 10 seconds. If you then run a real calculation on only the first portion and note that it took 30 seconds to run, you know that the other portion also will take 30 seconds. If there were two messages total, then you would know that the real calculation will take approximately 60 seconds (20 / 10 * 30 = 60 seconds).

Use the following topics to learn how to perform a simulated calculation and how to use a simulated calculation to estimate calculation time.

Simulate a Calculation

Simulate calculation on a model of your Essbase block storage cube by running a calculation script with commands SET MSG ONLY and SET NOTICE HIGH, and taking note of timestamps in the application log.

Before you can estimate calculation time, you must perform a simulated calculation on a data model that is based on your actual cube. Then you can check the application log to see how long calculations took.

To learn where to find the application log, refer to Environment Locations in the Essbase Platform.

To perform a simulated calculation:

1. Create a data model that uses all dimensions and all levels of detail about which you want information.
2. Load all data. This procedure calculates only data loaded in the cube.
3. Create a calculation script with these entries:

```
SET MSG ONLY;
SET NOTICE HIGH;
CALC ALL;
```

If you are using dynamic calculations on dense dimensions, substitute the CALC ALL command with the specific dimensions that you need to calculate; for example, CALC DIM EAST.

> **Note:**
>
> If you try to validate the script, Essbase reports an error. Disregard the error.

4. Run the script.
5. Find the first sparse calculation message in the application log and note the time in the message.
6. Note the time for each subsequent message.
7. Calculate the dense dimensions of the model that are not being dynamically calculated:

```
CALC DIM (DENSE_DIM1, DENSE_DIM2, …);
```

8. Calculate the sparse dimensions of the model:

```
CALC DIM (SPARSEDIM1, SPARSEDIM2, …);
```

9. Project the intervals at which notices will occur, and then verify against sparse calculation results. You can then estimate calculation time.

Reference the calculation commands:

- SET MSG
- SET NOTICE
- CALC ALL
- CALC DIM

Estimate Calculation Time

Record the results of the simulated calculation, and use them to estimate actual calculation time for your Essbase block storage cube.

To estimate total calculation time:

1. Note the times of all the intervals between application log messages generated by SET NOTICE HIGH.

   See the table below.
2. Use the following calculation to estimate the time for a real calculation:

   Total time required for simulated calculation, divided by the first simulated calculation notice interval, multiplied by the first real calculation time interval.

Table 35-1 Sample Intervals Between Log Messages

Calculation Notice Number	Simulated Calculation Time Interval (in seconds)	Sparse Dimension Calculation Interval (in seconds)
1	7	45
2	5	
3	6	
4	3	
5	4	
6	2	
7	6	
8	4	
9	3	
10	3	
Total calculation time	43	

In this example, 43 / 7 * 45 = 276.4 seconds, so the real calculation should take 276.4 seconds.

Factors Affecting Estimate Accuracy

The accuracy of estimations based on a simulated calculation of an Essbase block storage cube can be impacted by factors such as block density and outline structure.

The simulated calculation should return a time accurate to about 5%, if key factors causing variations are not present.

When these factors *are* present, this estimating technique more closely predicts calculation time when Essbase reaches 30%–40% of the simulated calculations (30%–40% of the messages generated by SET NOTICE HIGH).

Variations Due to a Chain of Influences

Using SET MSG ONLY as a calculation-time estimating technique should be validated against later CALCNOTICE intervals. The results of this estimating technique vary because of the following chain of influences:

1. Blocks differ in block density through the real consolidation process, therefore
2. The rate at which Essbase writes blocks to the disk differs, therefore
3. The rate at which blocks are processed in the cache differs, therefore
4. Actual results may differ from the predicted calculation time.

Variations Due to Outline Structure

Another factor that can make actual results diverge significantly from predicted is the outline structure. Calculations based on CALCNOTICE intervals assume evenly balanced processing time throughout the outline. Factors that can skew this balance include the following situations:

- The model contains one or two sparse dimensions that are large in relation to the other sparse dimensions.
- Larger dimensions have member configurations that result in multiple shared roll-ups.

Changing the Outline Based on Results

After you have estimated and analyzed a simulated calculation of a model of your Essbase block storage cube, you can change the outline to improve performance.

From top to bottom in the outline, order sparse dimensions to create the fewest percentage increases in upper blocks:

- Level 0 blocks following full model load: 100,000
- Upper level blocks after consolidating only sparse dimension 1: 1,000,000
- Upper level blocks after consolidating only sparse dimension 2: 3,000,000
- Upper level blocks after consolidating only sparse dimension 3: 10,000,000
- Upper level blocks after consolidating only sparse dimension 4: 300,000
- Upper level blocks after consolidating only sparse dimension 5: 5,700,000

For example:

- #4 (members = 10,000, 4 levels)
- #1 (members = 500, 2 levels)
- #2 (members = 100, 4 levels)
- #5 (members = 10,000, 4 levels)
- #3 (members = 20, flat)

Use the simulated calculation to generate the upper block count. These numbers may be accurate despite actual dimension sizes as noted next to the items above.

> **Caution:**
>
> The largest count of members is not always a good predictor.

Consider using hybrid mode. In a hybrid mode cube, you can easily customize the solve order instead of rearranging dimensional order.

Estimate Calculation Effects on Cube Size

Given the current number of blocks in an Essbase cube, you can issue the MaxL statement **query database … get estimated size** to estimate the number of blocks that a CALC ALL calculation will produce.

To estimate the cube size resulting from a calculation, using interactive mode:

1. Load data, issue a CALC ALL command, and note the average block size.
2. Start the MaxL client or shell, log on to Essbase, and start an application and cube. For example:

```
login username password;
```

```
alter system load application appname;
alter application appname load database dbname;
```

3. Providing the application and database (cube) name, enter the following MaxL query database statement, and note the value that is returned for the number of blocks:

```
query database appname.dbname get estimated size;
```

4. Multiply the number of blocks by the average size of the blocks in the cube.

   Results are accurate to ±10%.

Be aware of the following conditions when you query Essbase for an estimate of the full size of the cube:

- You must perform this query after a CALC ALL. Any other calculation will not produce accurate results.
- You can obtain accurate results with formulas only if they are on sparse dimensions.
- If you need to estimate partitions, you must query Essbase for a cube size estimate on every partition and add the results. If you query for the size of only the source cube, the estimate includes only the data on the source cube server.

Essbase Formula Optimization

To optimize Essbase block storage formula processing for calc and query, you can avoid formulas by using consolidation instead of calculation, use simpler formulas when possible, take steps to tune performance for complex formulas, or switch to hybrid mode.

You may achieve significant improvements in calculation performance by carefully using formulas in the cube outline. For example, you may achieve improved calculation performance by placing formulas on members in the database outline instead of placing the formulas in a calculation script. See Develop Formulas for Block Storage Cubes.

The sections that follow discuss how to handle formula issues that affect calculation performance in non-hybrid mode, block storage cubes.

For hybrid mode optimization, refer instead to Hybrid Mode for Fast Analytic Processing.

Essbase Member Consolidation

Using the cube outline to roll up Essbase data values is more efficient than using a formula to calculate them.

For example, the consolidation of members 100-10, 100-20, and 100-30 into member 100, as shown below, is more efficient than applying the following formula to member 100:

```
100-10 + 100-20 + 100-30
```

Figure 35-1 Consolidation Example

```
100 (+) (Alias: Colas)
    100-10 (+) (Alias: Cola)
    100-20 (+) (Alias: Diet Cola)
    100-30 (+) (Alias: Caffeine Free Cola)
```

Refer to Examples of Member Consolidation.

Simple Essbase Member Formulas

If an Essbase formula is simple, it affects block storage calculation performance less than complex formulas do. Block size is also a factor in performance. Simpler formulas are ones that don't use cross dimensional references, range functions, financial functions, or relationship functions.

If you use a simple formula, and block size is not unusually large, you can place the formula on a member of either a sparse or a dense dimension without significantly affecting calculation performance. The bigger the block size, the more impact simple formulas have on calculation performance.

A simple formula is, for example, a ratio or a percentage and meets the following requirements:

- Does not reference values from a different dimension (sparse or dense). For example, a simple formula cannot reference Product -> Jan.
- Does not use range functions. For example, a simple formula cannot use @AVGRANGE, @MAXRANGE, @MINRANGE, or @SUMRANGE.
- Does not use relationship or financial functions. For example, a simple formula cannot use @ANCESTVAL, @NEXT, @PARENTVAL, @SHIFT, @ACCUM, or @GROWTH.

References

For information on the relationship between block size and calculation performance, refer to Block Size and Block Density.

For information on how formulas affect calculation performance, refer to Bottom-Up and Top-Down Calculation.

Range and Financial Functions
Relationship Functions

Cross-Dimensional Operator

Complex Essbase Member Formulas

If an Essbase formula is complex, it affects block storage calculation performance more than simpler formulas do. To optimize complex formulas, apply them in dense dimensions, increase the density, and use FIX to narrow the scope.

If you use a complex formula, you can improve performance by applying the following guidelines:

- If possible, apply the formula to a member in a *dense* dimension. If not possible, determine whether you can change top-down calculations to bottom-up (refer to Formulas on Sparse Dimensions in Large Outlines).
- Use the FIX command in a calculation script to calculate only required data blocks.
- Increase the density of the database (ratio of existing data blocks to possible data blocks).

A complex formula is one that meets any of the following requirements:

- References a member or members in a different dimension (sparse or dense); for example, Product -> Jan.
- Uses one or more range functions, for example, @AVGRANGE, @MAXRANGE, @MINRANGE, or @SUMRANGE.

- Uses relationship or financial functions; for example, @ANCESTVAL, @NEXT, @PARENTVAL, @SHIFT, @ACCUM, or @GROWTH.

When applied to sparse dimension members, complex formulas create more calculation overhead and therefore slow performance. This problem occurs because the presence of complex formulas requires Essbase to perform calculations on all possible as well as all existing data blocks related to the member with the complex formula. The presence of a relationship or financial function on a sparse dimension member causes Essbase to perform calculations on all blocks, possible as well as existing, increasing the overhead even more.

Thus, a complex formula that includes a relationship or financial function creates a greater overhead increase than does a complex formula that does not include a relationship or financial function.

Two examples illustrate complex formula overhead:

- If the cube has 90 existing data blocks and 100 potential data blocks, the overhead for complex formulas is not large, not more than 10 extra blocks to read and possibly write values to.
- If the cube has 10 existing data blocks and 100 potential data blocks, the overhead is as much as ten times what it would be without the complex formula (depending on the outline structure and other factors), as many as 90 extra blocks to read and possibly write to.

In all cases, the lower the ratio of existing data blocks to possible data blocks, the higher the calculation performance overhead and the slower the performance.

References

For information on the relationship between block size and calculation performance, refer to Block Size and Block Density.

For information on how formulas affect calculation performance, refer to Bottom-Up and Top-Down Calculation.

Range and Financial Functions
Relationship Functions

Formulas on Sparse Dimensions in Large Outlines

To optimize the calculation of complex formulas on sparse dimensions in large Essbase block storage cube outlines, you can force a bottom-up calculation of the formulas.

You can use the SET FRMLBOTTOMUP calculation command to optimize the calculation of formulas in sparse dimensions in large database outlines. With this command, you can force a bottom-up calculation on sparse member formulas that otherwise would be calculated top-down. Refer to Bottom-Up and Top-Down Calculation.

Forcing a bottom-up calculation on a top-down formula enables more efficient use of the CALC ALL and CALC DIM commands. Review the discussions of the SET FRMLBOTTOMUP calculation command and the CALCOPTFRMLBOTTOMUP configuration setting.

Constant Values Assigned to Members in a Sparse Dimension

If you assign a constant to a member in a sparse dimension, Essbase automatically creates a data block for every combination of sparse dimension members that contains the member.

For example, assume that a member or a calculation script formula contains the following expression:

```
California = 120;
```

In this formula, California is a member in a sparse dimension and 120 is a constant value. Essbase automatically creates all possible data blocks for California and assigns the value 120 to all data cells. Many thousands of data blocks may be created. To improve performance, create a formula that does not create unnecessary values.

To assign constants in a sparse dimension to only those intersections that require a value, use FIX in a manner similar to the following example:

```
FIX(Colas,Misc,Actual)
   California = 120;
ENDFIX
```

In this example, Colas is a member of the sparse dimension, Product; Actual is a member of the dense dimension, Scenario; and Misc is a member of the dense dimension, Measures. The value 120 is assigned to any intersection of California (in the Market dimension), Actual (in the Scenario dimension), Misc (in the Measures dimension), Colas (in the Product dimension), and any member in the Year dimension, because a specific member of Year is not specified in the script.

Because Sample.Basic includes only two sparse dimensions, this example affects only one block. If more sparse dimensions existed, Essbase would ensure data blocks for all combinations of the sparse dimensions with California and Colas, creating blocks if necessary. Within the new blocks, Essbase sets Measures and Scenario values (other than those assigned the value 120) to #MISSING.

Nonconstant Values Assigned to Members in a Sparse Dimension

If you assign nonconstant values to members of a sparse dimension, blocks are created based on the Create Blocks on Equations setting. The Create Blocks on Equations setting is defined at the database level, as a database property. (See Nonconstant Values.)

Within calculation scripts, you can temporarily override the Create Blocks on Equations setting. Consider the effects of the following calculation when West does not have a value and Create Blocks on Equations is enabled:

```
West = California + 120;
```

Unneeded blocks may be created for all sparse-member intersections with West, even if the corresponding block value is #MISSING for all of the children of West. Especially in a large database, creation and processing of unneeded blocks requires additional processing time.

To control creation of blocks when you assign nonconstant values to members of a sparse dimension, use the SET CREATEBLOCKONEQ ON | OFF calculation command, as shown in the following script:

```
FIX (Colas);
   SET CREATEBLOCKONEQ OFF
   West = California + 120;
   SET CREATEBLOCKONEQ ON
```

```
   East = “New York” + 100;
ENDFIX
```

Because the Create Block on Equation setting is disabled at the beginning of the script, West blocks are created only when values exist for the children of West. Later, because the Create Block on Equation setting is enabled, all blocks for East are created.

> **Note:**
>
> Using SET CREATEBLOCKONEQ affects only creation of blocks during the execution of the calculation script that contains this command. This command does not change the overall database setting for Create Blocks on Equations.

Cross-Dimensional Operators in Member Formulas

When designing Essbase block storage member formulas, use caution when using a cross-dimensional operator (->) on the left side of an equation, or in equations on dense dimensions.

Left Side of an Equation

For faster calculation script performance, use FIX in the calculation script to qualify the use of a formula rather than a formula that includes a cross-dimensional operator on the left of an equation.

For example, assume that you want to increase the Jan -> Sales values in Sample.Basic by 5%. To improve performance by calculating only the relevant combinations of members, use the FIX command:

```
FIX(Jan)
   Sales = Sales * .05;
ENDFIX
```

With the FIX command, Essbase calculates the formula only for specified member combinations, in this example, for combinations that include Jan.

Compare this technique to using the slower cross-dimensional operator approach. For the previous example, you place the following formula on the Sales member in the database outline:

```
Sales(Sales -> Jan = Sales -> Jan * .05;)
```

As Essbase cycles through the database, it calculates the formula for every member combination that includes a member from the dimension tagged as time (Jan, Feb, Mar, and so on), although only January combinations need to be calculated.

See Use the FIX Command.

Equations in a Dense Dimension

When you use a cross-dimensional operator in an equation in a dense dimension, Essbase does not automatically create the required blocks if both of these conditions apply:

- Resultant values are from a dense dimension.
- The operand or operands are from a sparse dimension.

You can use the following techniques to create the blocks and avoid the performance issue.

- Ensure that the results members are from a sparse dimension, not from a dense dimension. In this example, the results member Budget is from a sparse dimension:

```
FIX(Sales)
    Budget = Actual * 1.1;
ENDFIX
FIX(Expenses)
    Budget = Actual *  .95;
ENDFIX
```

- Use the DATACOPY calculation command to create and then calculate the required blocks. See Copy Existing Blocks with DATACOPY.
- Use a member formula that contains the dense member equations:

```
FIX(Sales, Expenses)
Budget (Sales = Sales -> Actual * 1.1;
Expenses = Expenses -> Actual * .95;)
ENDFIX
```

Managing Formula Execution Levels

Formulas in a block storage outline can have dependencies on one another such that they cause a nested execution of formulas within one or more blocks. Such formulas are called recursive formulas. Sometimes recursive formulas result in large or unending loops that result in abnormal termination of the server.

To avoid abnormal termination, you can use the CALCLIMITFORMULARECURSION configuration setting to stop a formula execution that reaches beyond a default number of execution levels.

Bottom-Up and Top-Down Calculation

In an Essbase block storage cube, top-down calculations are less efficient than bottom-up calcs, because more blocks are calculated than is necessary. Top-down calculations may be necessary in some cases to ensure that calculation results are correct.

Essbase performs a full calculation of a database outline using either bottom-up calculation or top-down calculation. By default, Essbase uses bottom-up calculation in block storage cubes that are not hybrid mode.

> **Note:**
>
> If your cube uses hybrid mode, which is the default in Essbase 21c, you can skip the information in this topic. Hybrid mode cubes are already optimized for dynamic dependency analysis. To learn more, refer to Adopt Hybrid Mode for Fast Analytic Processing.

For a bottom-up calculation, Essbase determines which data blocks must be calculated before it calculates the cube. Essbase then calculates only the blocks that must be calculated. The calculation begins with the existing block with the lowest block number and works up through each block in number order until the existing block with the highest block number is reached. See Block Calculation Order.

If the outline contains a complex member formula, Essbase performs a top-down calculation for the relevant member.

Bottom-Up Calculations and Simple Formulas

For simple formulas, Essbase does a bottom-up calculation to determine which blocks must be calculated before running the full calculation. For example, for a simple formula on a member (such as A = B + C), A is calculated only if B or C exists in the database. That is, the dependency of the formula on B and C is known before the calculation is started.

Top-Down Calculations and Complex Formulas

Before starting a calculation, Essbase searches the outline and marks complex formulas that require top-down calculation; for example, a member formula that contains a cross-dimensional reference. When Essbase reaches a member with a top-down formula, it does a top-down calculation for the member.

When a formula on a member is complex, all possible blocks for the member must be examined to see if an existing block must be changed or a new block created; it is difficult to determine the dependency that blocks have on other blocks before the start of the calculation. The top-down method slows calculation performance because Essbase must search for appropriate blocks to calculate to execute the formula.

When a formula is compiled, if the formula is to be calculated top-down, Essbase logs a message in the application log file.

Consider the following complex formula:

```
A = B -> D + C -> D
```

To calculate the formula, Essbase must examine every combination of A to see whether B -> D or C -> D exists.

See Complex Essbase Member Formulas.

Forcing a Bottom-Up Calculation

If it is appropriate for the site, you can force a bottom-up calculation on a top-down formula.

To force a bottom-up calculation, use the use one of the following methods:

- Calculation function—@CALCMODE in a formula
- Calculation script command—SET FRMLBOTTOMUP
- Configuration settings:
  - CALCOPTFRMLBOTTOMUP
  - CALCMODE

Forcing a bottom-up calculation on a formula usually increases performance time. If the formula contains complex functions (for example, range functions) or if the formula's dependencies are not straightforward, a bottom-up calculation may produce results different from those of a top-down calculation.

> **Caution:**
>
> Before changing the setting CALCOPTFRMLBOTTOMUP or using the calculation script command SET FRMLBOTTOMUP in a production environment, check the validity of calculation results by comparing, relative to the same data, the results of a bottom-up calculation and the results of a top-down calculation.

Hybrid Mode for Fast Analytic Processing

The Oracle Essbase hybrid mode calculation and query processor enables you to perform real-time analytics using procedural calculations and read-and-write modeling. Hybrid mode is the default engine for dynamic dependency analysis for block storage queries. It is not the default for calculation scripts (you can enable it).

If you have worked with Essbase 11g On-Premise, then you likely are familiar with one or more of these cube design modes, tailored for different purposes:

- Block storage: best used when there are large, sparse dimensions. Cubes in this mode are stored and pre-aggregated to achieve good query performance. Includes a rich set of calculation functions for analysis.
- Aggregate storage: best used for cubes having a large number of dimensions, and many upper-level aggregations. Member formulas can be specified using MDX.
- Hybrid mode: block storage mode enhanced with the benefits of aggregate storage.

Hybrid mode is the default query engine for dynamic dependency analysis for queries, on block storage cubes in Essbase 21c, Essbase 19c, and Oracle Analytics Cloud - Essbase. Hybrid mode provides robust dependency analysis and fast aggregation. It is excellent at handling the complexity of querying members that have dependencies on dynamic members.

In your analytic applications, Oracle recommends the use of dynamic dependencies, including sparse aggregations. You are not limited to implementing Dynamic Calc selectively on sparse dimensions, as was the case in Essbase 11g On-Premise. In particular, sparse dynamic aggregations are possible and recommended, subject to performance tuning guidelines and testing.

Though hybrid mode is the default query processor for block storage cubes, it is not the default for executing calculation scripts. If your calculation scripts contain many dynamic dependencies, Oracle recommends you enable hybrid mode for calculation scripts as well. The way to do this is to turn on the HYBRIDBSOINCALCSCRIPT configuration setting in your application configuration properties (or use the SET HYBRIDBSOINCALCSCRIPT calculation command to control it on a per-calculation basis).

Most Essbase calculation functions will operate in hybrid mode. To see a list and syntax for all hybrid mode-supported calculation functions, as well as the few exceptions, see Functions Supported in Hybrid Mode. Parallel calculation using FIXPARALLEL is supported in hybrid mode, but not parallel calculation using CALCPARALLEL.

See ASODYNAMICAGGINBSO for the syntax to configure hybrid mode beyond the default settings, or to turn it off.

Topics in this section:

- Benefits of Hybrid Mode
- Comparison of Hybrid Mode, Block Storage, and Aggregate Storage

Benefits of Hybrid Mode

Essbase hybrid mode cubes enable you to benefit from fast aggregation even across sparse dimensions, smaller cube size, optimized memory footprint, flexible batch calculations, and robust formula dependency analysis.

Hybrid mode combines block storage (BSO) procedural calculation and write back functionality with aggregate storage (ASO) aggregation performance. Hybrid mode offers the benefit of fast performance by eliminating the need to store sparse aggregations. This, in turn, reduces database size and memory footprint, and speeds up batch calculation times. The deployment considerations are simplified, as you no longer have to consider using block storage for heavy use of level 0 calculations, versus aggregate storage for many upper-level aggregations, versus designing partitioned models in which the cube is split along dimensional lines to facilitate calculation performance.

The following are some scenarios where hybrid mode is likely to improve calculation performance:

- A block storage database has sparse members that are not level 0, and are calculated according to hierarchy (rather than by calculation scripts).
- A sparse, Dynamic Calc parent member has more than 100 children.
- You are using a transparent partition between an empty aggregate storage target and a block storage source. If the formulas on the aggregate storage target are simple and translatable to block storage formula language, you can achieve fast results on block storage using hybrid mode.
- You are using a transparent partition between two block storage databases, and calculation performance is a concern.

Another benefit of hybrid mode is that there is no outline order dependency. You can easily customize the solve order instead of rearranging dimensional order.

Hybrid mode also enables you to use scenario management, to test and model hypothetical data using a workflow format without adding storage requirements.

Comparison of Hybrid Mode, Block Storage, and Aggregate Storage

Without hybrid mode, the block storage algorithm for Dynamic Calc members has limitations when used with large, sparse dimensions. Hybrid mode (and aggregate storage) are more optimized for dynamic dependency analysis. Read about key differences to help you choose the best query processor type for an Essbase application.

Without hybrid mode, large, sparse dimensions in block storage databases must be stored; making them dynamic would result in too much block I/O at query or calculation time, affecting performance. Very large stored sparse dimensions can lead to lengthy batch aggregation times, as well as large database sizes that grow in relation to the number and size of the sparse dimensions. Even with such drawbacks, block storage is widely used for its powerful functionality.

Aggregate storage is designed specifically to enable large databases with more and larger dimensions. Unlike block storage, it does not require large sparse dimensions to be pre-aggregated to achieve good query performance. The key lies in the aggregate storage database kernel, which facilitates rapid dynamic aggregation across large dimensionality.

For all the benefits that aggregate storage offers, however, there are many uses that are better suited to block storage, such as the ability to load data at any granularity, or to frequently run complex batch allocations, or implement currency conversion for global financials. In such cases, and many more, hybrid mode might be the solution. Hybrid mode is a combination of the best features of block storage and aggregate storage. In hybrid mode, Essbase

- Enables full procedural calculation flexibility, even when the calculations depend on sparse, dynamic aggregations.
- Uses the hybrid engine for queries accessing dynamic sparse members. For the small percentage of queries that cannot be processed this way, Essbase employs the block storage calculation flow to satisfy the request.
- Offers these benefits, if you mark sparse members as dynamic:
  - Eliminates the need for pre-aggregation
  - Improves restructure performance
  - Improves backup performance
  - Reduces disk space requirements
- Because hybrid mode involves dynamic calculations, you can sequence the calculations by using solve order.

> **Note:**
>
> Hybrid calculations, whether driven by queries or calculation scripts, are performed in temporary memory space, utilizing a formula cache and the aggregate storage cache.

Key Differences

The following key differences can help you choose the best query processor type for your application.

Requirement	Aggregate Storage (ASO)	Block Storage (BSO)	Hybrid Mode
Optimized for rapid aggregation across many sparse dimensions	Yes	No	Yes
Optimized for minimal disk space usage and reduced backup time	Yes	No	Yes
Optimized for financial applications	No	Yes	Yes
Ability to perform allocations	Yes	Yes	Yes
Ability to perform batch calculations	No	Yes	Yes

Requirement	Aggregate Storage (ASO)	Block Storage (BSO)	Hybrid Mode
Member formulas supported	Yes, expressed as MDX	Yes, expressed as Essbase Calculation Functions	Yes, expressed as Essbase Calculation Functions
Optimized for forward references in member formulas	No	No	Yes
Ability to customize solve order of calculations/ aggregations	Yes	No	Yes Solve Order in Hybrid Mode
Ability to specify bottom-up query execution for faster dependency analysis of smaller input data sets	No	No	Yes QUERYBOTTOMUP configuration setting @QUERYBOTTOMUP calc function
Ability to trace and debug query execution	Yes QUERYTRACE	No	Yes QUERYTRACE
Ability to limit memory use permitted for a query	Yes MAXFORMULACACHE SIZE	No	Yes MAXFORMULACACHE SIZE
Support for two-pass calculation	No	Yes	No
Ability to load data at any level	No. Only level 0 cells without formula dependencies can be loaded	Yes	Yes for stored levels No for dynamic levels
Ability to load data incrementally using buffers	Yes	No	No
Evaluation of formulas on sparse dimensions can have different results than same formulas on dense dimensions	N/A	Yes. On block storage without hybrid mode, Essbase calculation scripts may be written iteratively with the purpose of resolving dependencies over sparse blocks. If you change the dimension type from sparse to dense or vice versa, you may get different results for the same formulas.	No. Formula dependencies are calculated the same without regard to sparsity or density. In hybrid mode, Essbase uses an algorithm to resolve dynamic dependencies. In some cases, the data derived from a calculation script may be different in hybrid mode than it would be in block storage mode without hybrid.

Get Started with Hybrid Mode

To get started with hybrid mode, follow these guidelines:

- Set up a development environment, and migrate existing block storage applications to it. Hybrid mode is enabled by default for block storage cubes.
- Where possible, make larger sparse dimensions dynamic.
- Run test queries and examine the application log, both before and after enabling hybrid mode. This activity can reveal the extent to which the aggregate storage query processor was used, and the benefits of hybrid mode that were gained. For each query, the application log states `Hybrid aggregation mode enabled` or `Hybrid aggregation mode disabled`.
- If too many queries are logged with hybrid mode disabled, contact Oracle Support.

Optimize the Cube for Hybrid Mode

To use hybrid mode most effectively:

- Avoid using two-pass calculation in hybrid mode. Use solve order instead.
- Convert non-level-0 stored members to Dynamic Calc wherever this is feasible.
- If the conversion to Dynamic Calc members affects solve order for dependent formulas, you may need to adjust the outline's order of dimensions to align the solve order with the previous batch calculation order and two-pass calc settings.

  The default solve order for hybrid mode cubes is similar to the calculation order of block storage cubes, with some enhancements. If you wish to use a non-default solve order, you can set a custom solve order for dimensions and members.
- A dynamically calculated formula processed in the wrong solve order can cause too many formulas to be executed by a query, degrading performance. When possible, a dynamically calculated sparse formula should have a higher solve order than hierarchically aggregated sparse dimensions.

  In some applications, this is not possible, as a different solve order is necessary to get the correct formula results. For example, an application with units and prices needs to have a sales value executed before the sparse aggregations in order to get the correct sales value at upper levels.
- You may need to adjust the dimensions' dense or sparse configurations (applies only to block-storage engine utilization in cases where the hybrid engine cannot be used).
- Minimize the size of blocks, if possible.

Essbase administrators can use the following tools to monitor and optimize query performance in hybrid mode:

- To limit how much memory may be consumed by any single query, use the MAXFORMULACACHESIZE configuration setting.
- If your cube has complex member formulas with cross-dimensional operators and multiple IF/ELSE statements, performance concerns may be related to formula execution. If this is suspected, you can activate bottom-up query processing for formula calculation. This optimizes query times by identifying the required intersections for calculation, making the query time proportional to input data size.

  To make these query optimizations for Release 21C, use the QUERYBOTTOMUP configuration setting, as well as the @QUERYBOTTOMUP calculation function. For Release 19C, use the IGNORECONSTANTS configuration setting with BOTTOMUP syntax, as well as the @NONEMPTYTUPLE calculation function.
- Use query tracing to monitor and debug query performance. Multiple application-level configuration settings are available, depending on your use case. Use QUERYTRACE for

short term debugging of a single query that you think might be problematic. Use TRACE_REPORT for statistics collection about concurrently running queries (ideal for debugging in a development environment). Use LONGQUERYTIMETHRESHOLD in production environments to print statistics to the application log file about any queries that run longer than a set time.

Limitations and Exceptions to Hybrid Mode

In some cases, a query would not execute optimally in hybrid mode. Essbase detects when these conditions are present, and aggregates them in block storage mode. If a query mixes supported and unsupported hybrid mode calculation types, Essbase defaults to block storage calculation execution.

If enabled, hybrid mode is in effect for member formulas using supported functions. For a list of supported and unsupported functions, see Functions Supported in Hybrid Mode.

The following types of queries are not executed in hybrid mode:

- Dynamic Calc members with formulas that are a target of transparent partitions
- Queries where the shared member is *outside* the target partition definition and its prototype member is *inside*, or the reverse
- XOLAP
- Text measures/text lists

Attribute calculations will execute in hybrid mode, for Sum only.

If dependent members have a higher solve order than the formula member, the following warning appears:

```
Solve order conflict - dependent member member_name with higher solve order
will not contribute value for formula of member_name
```

Solve Order in Hybrid Mode

Solve order in Essbase determines the order in which dynamic calculation executes in hybrid mode. You can customize the solve order or accept the default, which is optimized for high performance and dependency analysis.

The concept of solve order applies to dynamic calculation execution, whether initiated by a dynamic member formula or a dynamic dependency in a calculation script. When a cell is evaluated in a multidimensional query, the order in which the calculations should be resolved may be ambiguous, unless solve order is specified to indicate the required calculation priority.

You can set solve order for dimensions or members, or you can use the default Essbase solve order. The minimum solve order you can set is 0, and the maximum is 127. A higher solve order means the member is calculated later; for example, a member with a solve order of 1 is solved before a member with a solve order of 2.

When hybrid mode is enabled, the default solve order (also known as calculation order) closely matches that of block storage databases:

Dimension/Member Type	Default Solve Order Value
Stored members	0
Sparse Dimension Members	10

Dimension/Member Type	Default Solve Order Value
Dense Account dimension members	30
Dense Time dimension members	40
Dense regular dimension members	50
Attribute dimension members	90
Two pass dynamic members	100
MDX calculated members or named sets (defined in MDX With)	120

In summary, the default solve order in hybrid mode dictates that stored members are calculated before dynamic calc members, and sparse dimensions are calculated before dense dimensions, in the order in which they appear in the outline (top to bottom).

Dynamic members (with or without formulas) that do not have a specified solve order inherit the solve order of their dimension, unless they are tagged as two pass.

Two-pass calculation is a setting you can apply, in block storage mode, to members with formulas that must be calculated twice to produce the correct value.

Note:

Do not use two-pass calculation with hybrid mode cubes. Only use solve order.

Two pass is not applicable in hybrid mode, and any members tagged as two pass are calculated last, after attributes. In hybrid mode, you should implement a custom solve order, instead of two pass, if the default solve order does not meet your requirements.

The default solve order in hybrid mode is optimized for these scenarios:

- Forward references, in which a dynamic member formula references a member that comes later in the outline order. There is no outline order dependency in hybrid mode.
- Aggregation of child values based on outline order more closely matches aggregation using equivalent formulas.
- Dynamic dense members as dependencies inside sparse formulas. In hybrid mode, if a sparse formula references a dense dynamic member, the reference is ignored, because sparse dimensions are calculated first. To change this, assign a solve order to the sparse dimension that is higher than (calculated later than) the dense dimension's solve order.

Customizing the Solve Order

If you need to adjust the behavior of dynamic calculations in hybrid mode, customizing the solve order of dimensions and members helps you achieve it without making major changes to the outline.

If you implement a custom solve order, it overrides the default solve order. If members or dimensions have equal solve order, the order in which they appear in the outline (top to bottom) resolves the conflict.

Unless you customize a solve order for certain members, the top dimension member's solve order applies for all dynamic members in the dimension.

To change the solve order, use the outline editor in the Essbase web interface, or use Smart View (see Changing the Solve Order of a Selected POV).

The minimum solve order you can set is 0, and the maximum is 127. A higher solve order means the member is calculated later.

To explore use cases for solve order, see the Solve Order templates in the Technical section of the gallery of application workbooks, which you can find in the files catalog in Essbase.

Notes on Solve Order in Non-hybrid Mode

In aggregate storage cubes,

- Solve order is set to 0 for all dimensions.
- Aggregation executes in outline order, except:
  - Stored hierarchy members are processed first.
  - Dynamic hierarchy members are processed next.

In non-hybrid block storage cubes, the default solve order is

- sparse before dense
- accounts before time
- attributes last

> **Note:**
>
> If Accounts members' solve order are set manually to be greater than Time members' solve order, the Accounts will be evaluated after Dynamic Time Series members.

Essbase Caches and Calc Performance

To optimize calculation performance, especially for large calculations, Essbase uses caches to coordinate memory usage. Ensure that the calculator cache, if used, is large enough to optimize calculations. Review how the index cache, data cache, dynamic calculator cache, and application cache may be relevant.

When calculating a cube, Essbase uses approximately 30 bytes of memory per member in the outline. So if the cube has 5,000 members, Essbase needs approximately 150 KB of memory to calculate it.

> **Note:**
>
> You can avoid excess memory use by combining calculation scripts. You can obtain good performance by using parallel calculation with a single calculation script. See Using Parallel Calculation.

Essbase uses memory to optimize calculation performance, especially for large calculations. The amount of memory used is not controllable, except by altering the size of the cube outline. However, you can ensure that the memory cache sizes enable Essbase to optimize the calculation.

Essbase uses the following caches to coordinate memory usage:

- Calculator cache.

  When you first calculate a cube, the size of the calculator cache is significant for calculation performance. If possible, ensure that the calculator cache is large enough for Essbase to use the optimal calculator cache option.

  Refer to CALCCACHE configuration and SET CACHE command.
- Dynamic calculator cache. Refer to DYNCALCCACHEMAXSIZE application configuration property.
- Index cache. If the cube is large, the default index cache is not large enough to provide optimum calculation performance. Refer to INDEXCACHESIZE application configuration property, or MaxL alter database **set index_cache_size**.
- Data cache. Refer to DATACACHESIZE application configuration property, or MaxL alter database **set data_cache_size**.
- Data file cache.
- Application cache. If hybrid mode is used in block storage cubes, the application cache can help you manage memory usage for retrievals. The application cache is similar to the aggregate storage cache; for more information, refer to Aggregate Storage Cache.

Block Locking and Concurrent User Access

In block storage cubes, Essbase locks data blocks during calculations and data updates. The block locking system ensures that only one user at a time can update or calculate a particular data block.

During calculation, Essbase locks the data block and all blocks that contain the children of the block. Essbase calculates the block and then releases the block and the blocks containing the children.

By default, Essbase locks up to 100 blocks concurrently when calculating a block. This number of block locks is sufficient for most calculations. If you are calculating a formula in a sparse dimension, Essbase works most efficiently if it can lock all required child blocks concurrently. Therefore, when calculating a formula in a sparse dimension, you may want to set a lock number higher than 100 if you are consolidating very large numbers of children (for example, more than 100). By increasing the number, you ensure that Essbase can lock all required blocks, and performance is not impaired.

> **Note:**
>
> For consolidations in a sparse dimension, block locking is not a consideration, because Essbase does not need to lock all blocks containing children concurrently.

When Essbase calculates a data block, it creates an exclusive lock; other users cannot update or calculate it, but they can have read-only access. When Essbase finishes the calculation, it releases the block. Other users can then update the block if they have the appropriate security access.

When a user is updating a data block, the block is locked. If a database calculation requires a data block that is being updated by another user, the calculation waits for the data block to be released.

Essbase does not provide a message to say that the calculation is waiting for the data block to be released.

You can prevent calculation delays caused by waiting for locked blocks by using Essbase security options to do either of the following:

- Deny access to other users
- Disconnect users from Essbase

> **Note:**
>
> When Essbase locks a block for calculation, it does not put an exclusive lock on the dependent child blocks, so another user can update values in the child blocks. If necessary, you can use the above security options to prevent such updates.

Two-Pass Calculation

Two-pass calculation can be applicable for member formulas that must be calculated twice to produce the correct value. You can tag an accounts dimension member as two-pass to improve performance for some block storage Essbase applications.

> **Note:**
>
> Do not use two-pass calculation with hybrid mode cubes. Only use solve order.

For some applications, you may be able to improve performance by tagging an accounts dimension member as two-pass in the outline. However, the combination of data and calculation needs may require the use of a calculation script to calculate a formula twice, instead of two-pass tagging, to preserve accuracy.

Whenever possible, Essbase calculates two-pass formulas at the block level, simultaneously with the main calculation. In some situations, however, an extra calculation pass is needed.

How Essbase should calculate two-pass formulas depends on the dense-sparse configuration of the time and accounts dimensions.

Two-Pass Calculation Example

Consider this calculation required for Profit%:

```
Profit % = Profit % Sales
```

Assume that the table below shows a subset of a data block with Measures and Year as dense dimensions. Measures is tagged as accounts, and Year is tagged as time. The AGGMISSG configuration setting is turned off (the default).

Data values have been loaded into the input cells. Essbase calculates the cells in which the numbers 1 through 7 appear, in that order. For example, Profit % -> Jan is calculated first; Profit% -> Qtr1 has multiple consolidation paths.

Table 35-2 Two-Pass Calculation Example: Data and Calculation Order

Measures -> Year	Jan	Feb	Mar	Qtr1
Profit	75	50	120	**5**
Sales	150	200	240	**6**
Profit%	**1**	**2**	**3**	**4, 7**

> **Note:**
>
> For information on how cell calculation order depends on cube configuration, see Cell Calculation Order.

Essbase uses this calculation order:

1. Essbase calculates the formula `Profit % Sales` for Profit % -> Jan, Profit % -> Feb, Profit % -> Mar, and Profit % -> Qtr1 (1, 2, 3, 4 above).
2. Essbase calculates Profit -> Qtr1 and Sales -> Qtr1 by adding the values for Jan, Feb, and Mar (5, 6 above).
3. Essbase calculates Profit % -> Qtr1 by adding the values for Profit % -> Jan, Profit % -> Feb, and Profit % -> Mar (7 above). This addition of percentages produces the value 125%, which is not the correct result.

Table 35-3 Two-Pass Calculation Example: Incorrect Results

Measures/Year	Jan	Feb	Mar	Qtr1
Profit	**75**	**50**	**120**	**245 (5)**
Sales	**150**	**200**	**240**	**590 (6)**
Profit%	**50% (1)**	**25% (2)**	**50% (3)**	**0% (4)** **125% (7)**

4. If you tag Profit% as two-pass in the outline, Essbase uses the `Profit % Sales` formula to recalculate the Profit% values and produce the correct results.

Table 35-4 Two-Pass Calculation Example: Correct Results

Measures/Year	Jan	Feb	Mar	Qtr1
Profit	**75**	**50**	**120**	**245 (5)**
Sales	**150**	**200**	**240**	**590 (6)**
Profit%	**50% (1)**	**25% (2)**	**50% (3)**	**0% (4)** **125% (7)** **42% (8)**

For information about multiple calculation passes, see Calculation Passes.

Interaction of Two-Pass and Intelligent Calculation

If you are using Intelligent Calculation and two pass calculation in a block storage application, use the scenario that matches the configuration of the cube, to ensure that Essbase accurately calculates the two-pass formulas.

> **Note:**
>
> Do not use two-pass calculation with hybrid mode cubes. Only use solve order.

Scenario A: Two Pass Tag

In this scenario, you place formulas in the outline and, as appropriate, tag specific formulas as two-pass for best performance.

No Extra Calculation Pass for Two-Pass Formulas

Because Essbase calculates the two-pass formulas while it is calculating the data block, Essbase need not do an extra calculation pass.

All Data Blocks Marked As Clean

After the calculation, all data blocks are marked as clean for the purposes of Intelligent Calculation.

When you tag a member formula as two-pass in the outline, Essbase does the two-pass calculation while each data block is being calculated. However, when you repeat a formula in a calculation script, Essbase must read the data blocks and write them to memory to recalculate the formula.

Scenario B: Calculation Script

In this scenario, you create a calculation script to perform the formula calculation for best performance.

Extra Calculation Pass for Two-Pass Formulas

Essbase calculates the cube and then does an extra calculation pass to calculate the two-pass formulas. Even though all data blocks are marked as clean after the first calculation, Essbase ignores the clean status on the blocks that are relevant to the two-pass formula and recalculates these blocks.

Data Blocks for Two-pass Formulas Not Marked As Clean

After the first calculation, Essbase has marked all data blocks as clean. In a second calculation pass, Essbase recalculates the required data blocks for the two-pass formulas. However, because the second calculation is a partial calculation, Essbase does not mark the recalculated blocks as clean. When you recalculate with Intelligent Calculation turned on, these data blocks may be recalculated unnecessarily.

If the cube configuration allows Essbase to use Scenario B, consider using a calculation script to perform two-pass formula calculations. If you use a calculation script, Essbase still does an extra calculation pass through the database; however, you can ensure that Essbase has marked all the data blocks as clean after the calculation. See Calculation Scripts for Two-Pass and Intelligent Calculation.

Choosing Two-Pass Calculation Tag or Calculation Script Method

Although tagging an accounts member as two-pass may bring performance benefits, some applications cannot use this method. Check these qualifications to see whether you should apply a two-pass tag or create a calculation script that performs a calculation twice for best performance and accuracy:

- You can tag a member as two-pass if it is in a dimension tagged as accounts. When you perform a default calculation on the database, Essbase automatically recalculates any formulas tagged as two-pass if they are in the dimension tagged as accounts.
- You can tag a member as two-pass if it is a Dynamic Calc member of any dimension.
- You may need to use a calculation script to calculate a two-pass formula to obtain accurate results, even if the two-pass tag would provide performance benefits. See Calculation Scripts for Two-Pass and Intelligent Calculation.
- You must use a calculation script to calculate a formula twice if the database configuration means that Essbase uses Scenario A, and if the formula references values from another data block.
- You may want to use a calculation script to calculate two-pass formulas if the database configuration means that Essbase uses Scenario B.

Two-Pass on Default Calculations

A cube setting is available to enable two-pass calculation in default calculations. When you perform a default calculation on a cube with two-pass calculation enabled, Essbase automatically attempts to calculate formulas tagged as two-pass in the dimension tagged as accounts. This is true even if you have customized the default calculation script.

> **Note:**
>
> Do not use two-pass calculation with hybrid mode cubes. Only use solve order.

See these topics:

- Setting Advanced Cube Properties
- Setting Two-Pass Calculation Properties

To perform a default calculation, you can use the execute calculation MaxL statement.

To enable two-pass calculation, you can use the alter database MaxL statement.

Calculation Scripts for Two-Pass and Intelligent Calculation

When designing Essbase calculation scripts to perform two-pass calculations with Intelligent Calculation, learn how to use SET UPDATECALC, SET CLEARUPDATESTATUS, and CALC TWOPASS to optimize the calculations.

> **Note:**
>
> Do not use two-pass calculation with hybrid mode cubes. Only use solve order.

If your outline does use two-pass formulas, follow these guidelines to make the calculation as accurate and fast as possible:

- Before a command that recalculates a two-pass formula, add the SET UPDATECALC OFF command to disable Intelligent Calculation. If you have Intelligent Calculation enabled (the default), Essbase calculates only the data blocks that are not marked as clean, but when you perform a default calculation with Intelligent Calculation enabled, all data blocks are marked as clean, so Essbase does not perform the two-pass formula recalculation.
- Essbase does not automatically recalculate two-pass formulas in calc scripts, so you must use the CALC TWOPASS command.
- If you have changed the default calculation of CALC ALL, and Intelligent Calculation is enabled, the data blocks may not be marked as clean after the first calculation. See Intelligent Calculation for Block Storage Cubes.

To obtain the performance benefits of Intelligent Calculation when performing the first, full calculation of the database, use one of the methods shown in the following series of examples. Your best method depends on the calculation needs and outline structure of your cube.

In the examples that follow, assume the outline has a dimension tagged as accounts, and it is a dense dimension. You want to calculate sales for each product as a percentage of sales for all products. Assume this formula should calculate the dimension:

```
Sales % Sales -> Product
```

When Essbase calculates the data block for each product, it has not yet calculated the value Sales -> Product, so the results for the sales of each product as a percentage of total sales are incorrect.

Intelligent Calculation with a Large Index

If the cube index is large, and you want to use Intelligent Calculation, you can use any of the following options for the best performance.

Calculation Script Option

Use this model to create a calculation script that performs a full calculation with Intelligent Calculation enabled:

```
SET UPDATECALC ON;

CALC ALL;
SET UPDATECALC OFF;
```

```
SET CLEARUPDATESTATUS AFTER;
"Share of Sales" = Sales % Sales -> Product;
```

Calculation Script and Two-Pass Option

To tag a member as two-pass, and use a calculation script to calculate first the full cube, then the two-pass member:

1. Place a formula in the outline and tag it as two-pass.
2. Place the formula on the appropriate member in the dimension tagged as accounts (in our example, Share of Sales).
3. Create a calculation script that performs a full calculation and then a two-pass calculation:

```
SET UPDATECALC ON;
CALC ALL;
SET UPDATECALC OFF;
SET CLEARUPDATESTATUS AFTER;
CALC TWOPASS;
```

Client and Calculation Script Option

To perform a default calculation from a client and then use a calculation script to perform the formula calculation:

1. Enable Intelligent Calculation, if this default has been changed.
2. Perform a full calculation.
3. Use a calculation script similar to this example to disable Intelligent Calculation and calculate the formula:

```
SET UPDATECALC OFF;
SET CLEARUPDATESTATUS AFTER;
"Share of Sales" = Sales % Sales -> Product;
```

or

```
SET UPDATECALC OFF;
SET CLEARUPDATESTATUS AFTER;
CALC TWOPASS;
```

All the example options above perform these tasks:

1. Enable Intelligent Calculation.
2. Calculate the full cube and mark the data blocks as clean.
3. Disable Intelligent Calculation.
4. Mark the recalculated blocks as clean, even though this calculation is a partial calculation of the cube. If you do not use the command SET CLEARUPDATESTATUS AFTER, Essbase marks data blocks as clean only after a full calculation.
5. Essbase cycles through the cube, calculating only the formula for the relevant member (Share of Sales in our example), or calculating all formulas tagged as two-pass in the outline.

Example: Intelligent Calculation with a Small Index

To use Intelligent Calculation when the cube index is small:

1. Create a calculation script to calculate the cube, but tell Essbase not to mark the calculated data blocks as clean.
2. Mark all data blocks as clean and do not recalculate the data blocks.

```
SET CLEARUPDATESTATUS OFF;
CALC ALL;
CALC TWOPASS;
SET CLEARUPDATESTATUS ONLY;
CALC ALL;
```

   a. SET CLEARUPDATESTATUS OFF tells Essbase not to mark the calculated data blocks as clean.
   b. The first CALC ALL causes Essbase to cycle through the cube, calculating all dirty data blocks. Essbase does not mark the calculated blocks as clean.

      Essbase does not automatically recalculate the formulas tagged as two-pass.
   c. CALC TWOPASS causes Essbase to cycle through the cube, recalculating the two-pass formulas in the accounts dimension. Essbase recalculates the formulas because the required data blocks were not marked as clean by the previous CALC ALL. Essbase does not mark the recalculated data blocks as clean.
   d. SET CLEARUPDATESTATUS ONLY tells Essbase to mark the data blocks as clean, but not to calculate the data blocks (this command disables calculation).
   e. The last CALC ALL causes Essbase to cycle through the cube and mark all the data blocks as clean. Essbase does not calculate the data blocks.

Example: Intelligent Calculation Turned Off for a Two-Pass Formula

To turn Intelligent Calculation off for a Two-Pass formula, create a calculation script that performs these tasks:

- Disables Intelligent Calculation.
- Performs a full calculation.
- Repeats the following two-pass formula:

```
SET UPDATECALC OFF;
CALC ALL;
"Share of Sales" = Sales % Sales -> Product;
```

Member Set Functions and Performance

Complex member formulas with dynamic dependencies may impact performance of your Essbase block storage calculations, unless you enable hybrid mode processing for calculations.

Queries and calculations that reference a member that has been tagged as Dynamic Calc may be slower than queries and calculations involving the same members, if the member has formulas involving any of these functions:

- @CURRMBR
- @PARENT
- @SPARENTVAL
- @ANCEST
- @SANCESTVAL

If you have any of the above formulas referencing Dynamic Calc members, Oracle recommends using hybrid mode for query and calculations. Refer to Hybrid Mode for Fast Analytic Processing for more details.

#MISSING Values

If no data value exists for a combination of block storage dimension members, Essbase gives the combination a value of #MISSING. #MISSING values are different from zero (0) values. By default, #MISSING data is not consolidated, unless you enable a setting. You can remove #MISSING blocks to improve performance.

Consolidation of #MISSING

The following shows how Essbase calculates #MISSING values. In this table, X represents any number:

Table 35-5 How Essbase Treats #MISSING Values

Calculation/Operation	Result
X + #MISSING	X
X – #MISSING #MISSING – X	X -X
X * #MISSING	#MISSING
X / #MISSING #MISSING / X X / 0	#MISSING #MISSING #MISSING
X % #MISSING #MISSING % X X % 0	#MISSING #MISSING #MISSING
X == #MISSING	FALSE, unless X is #MISSING
X != #MISSING X < > #MISSING	TRUE, unless X is #MISSING TRUE, unless X is #MISSING
X <= #MISSING	(X <= 0)
X >= #MISSING	(X >= 0) or (X == #MISSING)
X > #MISSING	(X > 0)
X < #MISSING	(X < 0)
X AND #MISSING: Y AND #MISSING, where Y represents any nonzero value 0 AND #MISSING #MISSING AND #MISSING	#MISSING 0 #MISSING

Table 35-5 (Cont.) How Essbase Treats #MISSING Values

Calculation/Operation	Result
X OR #MISSING:	Y
Y OR #MISSING, where Y represents any nonzero value 0 OR #MISSING #MISSING OR #MISSING	#MISSING #MISSING
IF (#MISSING)	IF (0)
f (#MISSING)	#MISSING for any Essbase function of one variable
f (X)	#MISSING for any X not in the domain of *f* and any EssbaseEssbase function of multiple variables (except where specifically noted)

By default, Essbase does not roll up #MISSING values. However, if you always load data at level 0 and never at parent levels, you should enable the setting for consolidating #MISSING values. This setting provides a calculation performance improvement of 1%–30%. The performance improvement varies, depending on cube size and configuration.

Caution:

The default, not consolidating #MISSING values, must be in effect if you load data at parent, rather than child, levels, if any child member combinations have #MISSING values. If all child member combinations have any other values, including zero (0), Essbase rolls up the child values and overwrites the parent values correctly, so you can safely change the default.

To consolidate #MISSING values, use one of the following methods. The degree of performance improvement you achieve depends on the ratio between upper-level blocks and input blocks in the database.

To change how #MISSING values are consolidated, you can use one of these methods:

- Select the Aggregate missing values cube option in the Essbase web interface.

  See Setting Advanced Cube Properties.
- In a calculation script, use the SET AGGMISSG calculation command.
- Use the alter database **enable aggregate_missing** MaxL statement.

If you enable consolidation of #MISSING values in the cube properties, the cell calculation order within a data block changes.

When the setting for consolidating #MISSING values is disabled, note that the performance overhead is particularly high in the following situations:

- When the ratio of calculated data blocks to input data blocks is low
- When you load many data values at parent levels on sparse dimensions

In these situations, the performance overhead is 10%–30%. If calculation performance is critical, you may want to reconsider the database configuration or how you load data.

Removing #MISSING Blocks

You can use the CLEARDATA command to change the value of cells in a block to #MISSING. It does not remove the data blocks. These extra blocks can slow retrieval and calculation performance.

If the #MISSING blocks are slowing performance, perform either action:

- Use the CLEARBLOCK command to remove the data blocks.
- Export the data and re-import it.

Removing empty blocks improves performance when data values already have been loaded. However, data load process time increases if new values require that blocks be created.

Identify Additional Calculation Optimization Issues

The relationship between Essbase block storage calculation and performance is also described in topics about Dynamic Calc, developing calculation scripts, and partitioning. Links are provided for reference.

- In Dynamically Calculating Data Values, see the following topics:
  - Benefitting from Dynamic Calculation
  - Reducing the Impact on Retrieval Time
  - Dynamically Calculating Data in Partitions
- In Develop Calculation Scripts for Block Storage Cubes, see the following topics:
  - Specify Global Settings for a Calculation
  - Use Calculation Scripts on Partitions

When you convert currencies using the CCONV command, the resulting data blocks are marked as *dirty* for the purposes of Intelligent Calculation. This means that Essbase recalculates all the converted blocks when you recalculate the cube. .

36 Comparison of Aggregate and Block Storage

Essbase aggregate storage applications and databases differ in concept and design from block storage applications and databases. Some block storage concepts do not apply to aggregate storage. For example, aggregate storage cubes do not have dense and sparse dimensions.

- Inherent Differences with ASO
- Outline Differences for ASO
- Calculation Differences for ASO
- Partitioning Differences for ASO
- Data Load Differences for ASO
- Query Differences with ASO
- Feature Differences in ASO
- Hybrid Mode

Inherent Differences with ASO

Essbase aggregate storage (ASO) cubes are optimized for rapid aggregation across large, sparse dimensions. One cube per application is supported, and copying a cube is not supported. Configuration details are different than for block storage cubes.

Inherent Differences	Aggregate Storage	Block Storage
Storage kernel	Architecture that supports rapid aggregation, optimized to support high dimensionality and sparse data	Multiple blocks defined by dense and sparse dimensions and their members, optimized for financial applications
Copy database	Not supported, but you can copy the whole application	Supported
Databases supported per application	One	Several (one recommended)
Application and database names	Names reserved for tablespaces, cannot be used as application or database names: • default • log • metadata • temp Naming Conventions for Applications and Databases.	Naming Conventions for Applications and Databases.
Configuration settings	Aggregate Storage and Block Storage Settings Comparison	Aggregate Storage and Block Storage Settings Comparison

Outline Differences for ASO

Essbase aggregate storage (ASO) outlines have no dense or sparse dimensions, no two-pass calculation, no Dynamic Time Series, and no Dynamic Calc and Store. There are some restrictions on shared and label only members.

Table 36-1 Outline Differences Between Aggregate Storage and Block Storage

Outline Functionality	Aggregate Storage	Block Storage
Dense or sparse dimension designation	Not relevant	Relevant
Multiple hierarchies enabled, dynamic hierarchy, or stored hierarchy designation	Relevant	Irrelevant
Accounts dimensions and members on dynamic hierarchies	Support with the following exceptions: • No two-pass calculation (however, for information on specifying the calculation order, see Calculation Order and Solve Order in ASO Cubes) • No association of attribute dimensions with the compression dimension (aggregate storage only) • Additional restrictions for shared members. Review **Alternate Hierarchy Restrictions for Shared Members** in Hierarchies in ASO Cubes.	Full support
Members on stored hierarchies	Support with the following exceptions: • Support for the ~ (no consolidation) operator (underneath label-only members only) and the + (addition) operator • Cannot have formulas • Restrictions on label only members (See Member storage types.) • No Dynamic Time Series members • Stored hierarchy dimensions cannot have shared members. Stored hierarchies within a multiple hierarchies dimension can have shared members. Review **Stored Hierarchies** in Hierarchies in ASO Cubes.	Full support

Table 36-1 (Cont.) Outline Differences Between Aggregate Storage and Block Storage

Outline Functionality	Aggregate Storage	Block Storage

Member storage types	Support with the following exceptions:
	• Dynamic Calc and Store not relevant
	• On stored hierarchies, two limitations if a member is label only:
	  – All dimension members at the same level as the member must be label only
	  – The parents of the member must be label only.
	  – On dynamic hierarchies, ability to tag any member as label only
		Support for all member storage types in all types of dimensions except attribute dimensions

Ragged hierarchies and hierarchies with more than 10 levels	Support, with possible performance impact	Support
Outline validation	• When database is started • When outline is saved • When block storage outline is converted to aggregate storage outline • When user requests	• When outline is saved • When user requests
Outline paging	Support	No support
Database restructure	Levels of restructure; see Aggregate Storage Cube Restructuring	Levels of restructure; see Optimizing Database Restructuring

Calculation Differences for ASO

Calculation of aggregate storage (ASO) cubes differs from that of block storage cubes. Aggregations can be predefined as views. Formulas, if used on the outline, are written as MDX numeric value expressions, rather than using Essbase calculation functions.

Table 36-2 Calculation Differences Between Aggregate Storage and Block Storage

Calculation Functionality	Aggregate Storage	Block Storage
Cube calculation	Aggregation of the cube, which can be predefined by defining aggregate views	Block storage calculation scripts written with Essbase calculation commands and functions Outline consolidation Hybrid mode is encouraged.

Table 36-2 (Cont.) Calculation Differences Between Aggregate Storage and Block Storage

Calculation Functionality	Aggregate Storage	Block Storage
Formulas	Defined using MDX operators and functions. Note the following restrictions: • Must be valid numeric value expressions written in MDX; cannot contain the % operator—replace with expression: (value1 / value2) * 100 • No support for Essbase calculation functions • On dynamic hierarchy members, formulas are allowed without further restrictions	Defined using Essbase calculation functions
Calculation scripts	Aggregation scripts Custom ASO calculations, if the cube needs calculation scripts to handle special calculations and data dependencies	Stored/procedural calculation scripts defined using Essbase calculator syntax
Attribute calculations dimension	Support for Sum	Support for Sum, Count, Min, Max, and Average
Calculation order	Member formula calculation order can be defined by the administrator using the solve order member property	Defined by the administrator in the outline consolidation order or in a calculation script In hybrid mode, can be customized using the solve order member property

Related Links

Block Storage Calculation Features That Do Not Apply to Aggregate Storage

Aggregation of Data in an ASO Cube

Custom Calculations and Allocations on Aggregate Storage Cubes

Calculation Order and Solve Order in ASO Cubes

Partitioning Differences for ASO

Partitioned cubes in Essbase aggregate storage (ASO) differs from block storage in that outline synchronization is not supported.

Table 36-3 Partitioning Differences Between Aggregate Storage and Block Storage

Partitioning Functionality	Aggregate Storage	Block Storage
Partitioning	Support with the following restrictions: No outline synchronization	Fully supported

Data Load Differences for ASO

Loading data to Essbase aggregate storage cubes differs from block storage. Data loads only to stored, level-0 members. Temporary buffers help streamline flows from multiple sources, and slices can be merged and replaced. Date format strings help load to date-time dimensions. Aggregations cause recalculation post data load.

Table 36-4 Data Load Differences Between Aggregate Storage and Block Storage

Data Load Functionality	Aggregate Storage	Block Storage
Cells loaded through data loads	Only level 0 cells whose values do not depend on formulas in the outline are loaded	Cells at all levels can be loaded (except Dynamic Calc members)
Update of database values	At the end of a data load, if an aggregation exists, the values in the aggregation are recalculated	No automatic update of values. To update data values, you must execute all necessary calculation scripts.
Data load buffers	The loading of multiple data sources into aggregate storage databases is managed through temporary data load buffers	Not supported
Atomic replacement of the contents of a database	When loading data into an aggregate storage database, you can replace the contents of the database or the contents of all incremental data slices in the database	Not supported
Data slices	Aggregate storage databases can contain multiple slices of data. Data slices can be merged	Not supported
Dimension build for shared members	Full support for parent-child build method. Duplicate generation (DUPGEN) build method limited to building alternate hierarchies up to generation 2 (DUPGEN2).	Support for all build methods
Loading data mapped to dates	In a date-time dimension, you can load data into level 0 members using supported date-format strings instead of member names	Date-time dimension type is not supported
Renegade members	Supported	Not supported
Load rules and impact on existing values	Using load rules with the aggregate storage data load buffer, you can choose to overwrite all the cube data, or replace one slice at a time. Slices can be incrementally gathered into the data load buffer before they are loaded to the database.	Using load rules for block storage, you choose for each source of data whether to overwrite existing values, add values to existing values, or subtract values from existing values.

Related Topics

Source Data Differences for Aggregate Storage Data Loads

Data Source Differences for Aggregate Storage Dimension Builds

Renegade Members in Aggregate Storage Data Loads

Query Differences with ASO

Essbase aggregate storage (ASO) queries differ from block storage in some respsects. Writeback and export are supported only for level-0 members. Queries pertaining to density and sparsity are not applicable.

Table 36-5 Query Differences Between Aggregate Storage and Block Storage

Query Functionality	Aggregate Storage	Block Storage
Report Writer	Supported, except for commands related to sparsity and density of data	Fully supported
Smart View	Supported, with limited ability to change data (write-back)	Fully supported
API	Supported	Supported
Export	Support with the following restrictions: • Export of level 0 data only (no upper-level export) • No columnar export	Supported
MDX queries	Supported	Supported
Queries on attribute members that are associated with non-level 0 members	Returns values for descendants of the non-level 0 member. See also *Attribute Queries on ASO Cubes* in Design Considerations for Aggregate Storage Outlines.	Returns #MISSING for descendants of the non-level 0 member
Queries on attribute members and shared members	A shared member automatically shares the attribute associations of its prototype member	A shared member does not share the attribute associations of its prototype member
Query logging	Supported	Supported
Query performance	Considerations when querying data from a dimension that has multiple hierarchies. See also *Attribute Queries on ASO Cubes* in Design Considerations for Aggregate Storage Outlines.	Hierarchies not relevant

Feature Differences in ASO

Some features of Essbase aggregate storage (ASO) cubes differ from block storage or are not available. Only ASO offers date-time dimensions. Writeback is supported only to level-0 members. For time balance reporting, the time dimension must be stored, and shared members must be level-0.

Table 36-6 Feature Differences Between Aggregate and Block Storage

Features	Aggregate Storage	Block Storage
Aliases	Supported	Supported
Incremental data load	Supported	Supported
LROs	Not supported	Supported

Table 36-6 (Cont.) Feature Differences Between Aggregate and Block Storage

Features	Aggregate Storage	Block Storage
Time balance reporting	Support with the following restrictions: • Skip Zeros is not supported • Time dimension must contain only stored hierarchies • Shared members must be at level zero	Supported
Triggers	After-update triggers supported	On-update triggers and after-update triggers supported
Unicode	Supported	Supported
Variance reporting	Not supported	Supported
Date-time dimension type and linked attribute dimensions	Supported	Not supported
User ability to change data (write-back)	Supported with the following restriction: Write back to an aggregate storage database must be to level 0 members.	Supported

Hybrid Mode

Essbase hybrid mode combines block storage functionality with aggregate storage performance. In hybrid mode, you retain the use of block storage elements while enabling the use of more (and larger) sparse dimensions, while also reducing the database footprint and improving performance. See Adopt Hybrid Mode for Fast Analytic Processing.

37

Designing Aggregate Storage Applications

When you design an Essbase aggregate storage (ASO) application and cube, some starting factors to consider include retrieval optimization, use of hierarchies, and using MDX member formulas to calculate member relationships.

- Design Flow for Aggregate Storage Applications
- Aggregate Storage Applications, Cubes, and Outlines
- Developing Formulas on Aggregate Storage Outlines

Also see Comparison of Aggregate and Block Storage.

Design Flow for Aggregate Storage Applications

As part of the design phase for an Essbase aggregate storage (ASO) application and cube, you specify the cache size, load data and dimensions, perform aggregations, view statistics, write back to level 0 members, and analyze data.

This topic provides a high-level process for creating an aggregate storage application.

1. Create an aggregate storage application, cube (database), and outline.

   Aggregate Storage Applications, Cubes, and Outlines.

2. Specify the maximum size of the aggregate storage cache.

   Aggregate Storage Cache.

3. Load data into the aggregate storage cube. A data load can be combined with a dimension build.

   - Build Dimensions in Aggregate Storage Cubes
   - Load Data into Aggregate Storage Cubes

4. Precalculate chosen aggregations to optimize retrieval time.

   Aggregation of Data in an ASO Cube

5. View cube statistics.
6. Write back to an aggregate storage cube must be to level 0 members.
7. View data using Oracle tools (for example, Smart View) or third-party tools.

Aggregate Storage Applications, Cubes, and Outlines

You must create an Essbase aggregate storage (ASO) application to contain an aggregate storage cube. An aggregate storage application can contain only one cube. You can create a new ASO application and cube, or convert a block storage cube to ASO.

You can create an aggregate storage application, cube, and outline in the following ways:

- Create a new aggregate storage application and cube. The aggregate storage outline is created automatically when you create the cube. You can use an application workbook to create it.

- Convert a block storage outline to an aggregate storage outline, and then create an aggregate storage application to contain the converted cube and outline.

  Essbase supports the following scenarios for converting block storage outlines to aggregate storage outlines:

  – Non-Unicode block storage outline to non-Unicode aggregate storage outline
  – Non-Unicode block storage outline to Unicode aggregate storage outline
  – Unicode block storage outline to Unicode aggregate storage outline

  The following conversion scenarios are not supported:

  – Unicode block storage outline to non-Unicode aggregate storage outline
  – Aggregate storage outline to a block storage outline

For information on loading dimensions and members into an aggregate storage outline, see Build Dimensions in Aggregate Storage Cubes and Load Data into Aggregate Storage Cubes.

Aggregate storage application and cube information differs from block storage information, and specific naming restrictions apply to aggregate storage applications and cubes. See Inherent Differences with ASO.

To convert a block storage outline to an aggregate storage outline, you can use the create outline MaxL statement.

> **Note:**
>
> Do not use the file system to copy a block storage outline into an aggregate storage application.

To create an aggregate storage application or cube, use an application workbook or use these MaxL statements:

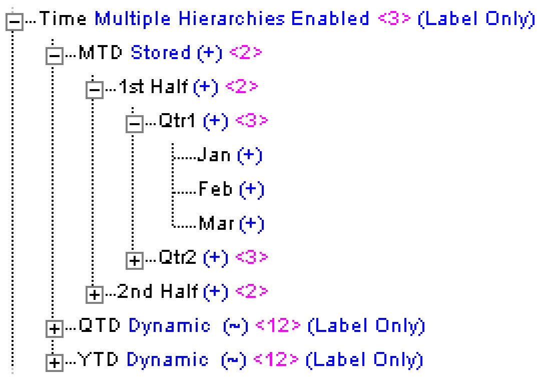
- create application
- create database

Hierarchies in ASO Cubes

Dimensions in Essbase outlines contain one or more hierarchies of related levels and members. Hierarchies are stored or dynamic. Stored hierarchies aggregate quickly, based on outline structure. Dynamic hierarchies are calculated at retrieval time, using formulas and solve order. An alternate hierarchy is a structure of attribute or shared members.

The Time dimension in the ASOsamp.Basic database includes the hierarchies MTD, QTD, and YTD:

Figure 37-1   Outline Showing Multiple Hierarchies and Members on the Time Dimension



In an aggregate storage cube, hierarchies can be **stored** or **dynamic**.

A dimension may contain both stored and dynamic hierarchies. Each type has its own advantages and restrictions. To use multiple hierarchies in a dimension (even if they are all stored), you must enable multiple hierarchies for that dimension.

When you tag a dimension member as multiple hierarchies enabled, it is automatically tagged label only.

If you do not tag the dimension as multiple hierarchies enabled, the dimension is automatically tagged as a stored hierarchy (except the dimension tagged as Accounts, which is tagged as a dynamic hierarchy).

> **Note:**
>
> The first hierarchy in a multiple hierarchies enabled dimension must be a stored hierarchy.

In the Essbase web interface, you enable multiple hierarchies for a dimension using a member property.

Stored Hierarchies

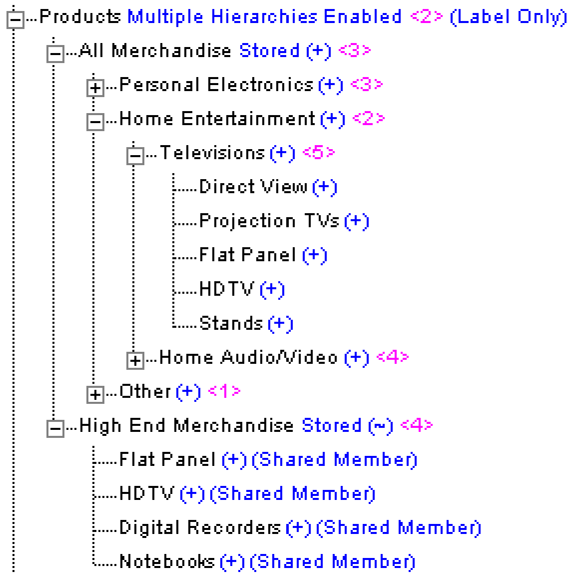
Members of stored hierarchies are aggregated according to the outline structure. Because aggregate storage databases are optimized for aggregation, the aggregation of data values for stored hierarchies is very fast. To allow this fast aggregation, members of stored hierarchies have the following restrictions:

- Stored hierarchies can have the no-consolidation (~) operator (only underneath label only members) or the addition (+) operator.
- Stored hierarchies cannot have formulas.

Stored hierarchies have restrictions on label only members. See Outline Differences for ASO.

In the hierarchy shown below, the All Merchandise hierarchy and the High End Merchandise hierarchy are stored hierarchies. The All Merchandise member and the High End Merchandise member are the tops of the hierarchies and are both tagged as top of a stored hierarchy.

Figure 37-2 Aggregate Storage Outline Displaying the Alternate Hierarchy High End Merchandise on the Product Dimension



In the Essbase web interface, you specify a stored hierarchy using a member property.

You can tag the top member of the hierarchy as top of a stored hierarchy by using the import database MaxL statement.

The following members can be tagged as top of a stored hierarchy:

- **A dimension member (generation 1).** If a dimension member is tagged as top of a stored hierarchy, the entire dimension is considered a single stored hierarchy, and no other member in the dimension can be tagged as top of a stored or dynamic hierarchy.
- **The children of the dimension member (generation 2).** If a generation 2 member is tagged as top of a stored hierarchy, all generation 2 members in the dimension also must be tagged as top of a stored or dynamic hierarchy. The first hierarchy in the dimension must be stored.

The dimension tagged as accounts is automatically considered a dynamic hierarchy. You cannot specify the accounts dimension as a stored hierarchy.

Dynamic Hierarchies

To evaluate a dynamic hierarchy, Essbase calculates, rather than aggregates, the members and formulas. The order in which members and formulas are evaluated is defined by the solve order property. See Calculation Order and Solve Order in ASO Cubes.

At the time of retrieval, Essbase calculates the required member combinations and calculates any required outline member formulas. Because dynamic hierarchies are calculated, the data retrieval time may be longer than for data retrieved from stored hierarchies. However, when you design your database, dynamic hierarchies provide the following advantages:

- They can contain any consolidation operator.
- They can have formulas.

In the Essbase web interface, you specify a dynamic hierarchy using a member property.

You can tag the top member of the hierarchy as top of a dynamic hierarchy using the import data MaxL statement.

The following members can be tagged as top of a dynamic hierarchy:

- **A dimension member (generation 1)**—If a dimension member is tagged as top of a dynamic hierarchy, the entire dimension is considered a single dynamic hierarchy, and no other member in the dimension can be tagged as top of a dynamic hierarchy or top of a stored hierarchy.
- **The children of the dimension member (generation 2)**—If a generation 2 member is tagged as top of a dynamic hierarchy, all generation 2 members in the dimension must also be tagged as either top of a dynamic hierarchy or top of a stored hierarchy. The first hierarchy in the dimension must be a stored hierarchy.

> **Note:**
>
> If a member has the no-consolidation operator (~) on all its children, the member must be tagged label only.

The dimension tagged accounts is automatically considered a dynamic hierarchy. You cannot specify the accounts dimension as a stored hierarchy.

Essbase cannot select dynamic hierarchy members for an aggregate view.

Alternate Hierarchies

An alternate hierarchy may be modeled in either of the following ways:

- As an attribute dimension, which uses attributes to classify members logically within the dimension (for example, a Product dimension can have attributes such as Size and Flavor).

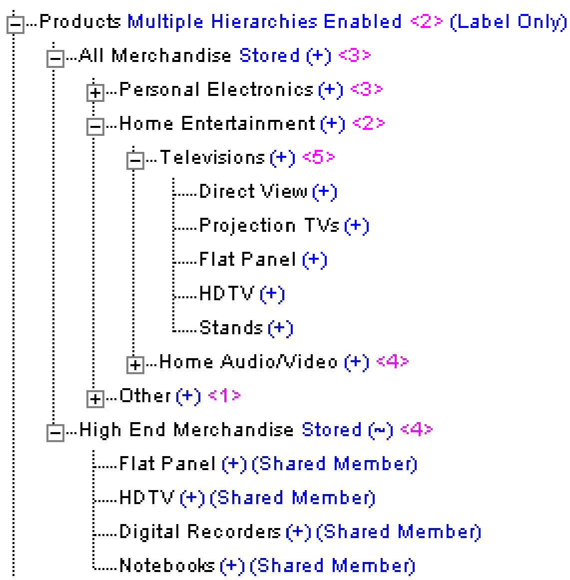
  > **Note:**
  >
  > If you use an attribute dimension as an alternate hierarchy, one useful query is a crosstab report of attribute dimension members by base dimension members. For example, a crosstab report of product sales information could show size attributes (such as small and large) on columns and products on rows. If you use shared members as an alternate hierarchy, you cannot create the equivalent crosstab report query of the shared members by prototype members.

- As a hierarchy of shared members. The alternate hierarchy has shared members that point to prototype members of previous hierarchies in the outline. The shared members roll up according to a different hierarchy than the prototype members. Shared members on dynamic hierarchies can have formulas. The following table shows the hierarchies for the ASOsamp.Basic database. The Products dimension is shown in the illustration that follows.

Table 37-1 Example Hierarchies and Alternate Hierarchies for the Product Dimension of ASOsamp.Basic

Product	Hierarchy	Alternate Hierarchy (containing shared members)
Flat Panel	Products, All Merchandise, Personal Electronics, Home Entertainment, Televisions	Products, High End Merchandise
HDTV	Products, All Merchandise, Personal Electronics, Home Entertainment, Televisions	Products, High End Merchandise

Figure 37-3 Aggregate Storage Outline Displaying the Alternate Hierarchy High End Merchandise on the Product Dimension



Alternate Hierarchy Restrictions for Shared Members

The following restrictions apply for alternate hierarchies in aggregate storage outlines:

- The prototype instance of the member must occur in the outline before any shared instances of the member. For example, in the above outline, the member HDTV occurs in the All Merchandise hierarchy before it occurs as a shared member in the alternate hierarchy of High End Merchandise.
- The first hierarchy in a dimension where multiple hierarchies are enabled cannot contain a shared member.
- Stored hierarchy dimensions cannot have shared members. Stored hierarchies within a multiple hierarchies dimension can have shared members.
- To ensure that values are not double-counted, a stored hierarchy cannot contain multiple copies of the same shared member. For example, a stored hierarchy cannot contain a shared member and any of its ancestors. In the above outline, you cannot add the shared member “Televisions” as a child of “High End Merchandise,” because doing so would make “Televisions” a sibling of its children, shared members “Flat Panel” and “HDTV,” causing the values of “Flat Panel” and “HDTV” to be added twice.

- Prototype instances of a member must be in the same dimension as the shared member (same for block storage outlines).
- A stored hierarchy cannot contain a prototype instance and a shared instance of the same member.
- A stored hierarchy can contain a shared instance of a dynamic hierarchy member only if the dynamic hierarchy member is a level 0 member without a formula.

> **Note:**
>
> In an aggregate storage cube, a shared member automatically shares any attributes that are associated with its prototype member.

Design Considerations for Aggregate Storage Outlines

This topic lists key design considerations when designing the outline for your Essbase aggregate storage (ASO) cube. How queries perform has much to do with how you plan the outline. Use the recommendations here when considering which hierarchies to implement as stored, dynamic, or alternate hierarchies (attribute dimensions and shared members).

For an implemented example of the design recommendations, see the ASOsamp.Basic cube, available in the gallery.

Designing the ASO Outline

Consider the following information when designing an aggregate storage outline:

- Use stored hierarchies (rather than dynamic hierarchies) as much as possible.
- Use shared members only when necessary.
- Minimize the number of hierarchies. (For example, each additional stored hierarchy slows down view selection and potentially increases the size of the aggregated data).
- If a hierarchy is a small subset of the first hierarchy, consider making the small hierarchy a dynamic hierarchy. Considerations include how often the hierarchy data is queried and the query performance impact when it is dynamically queried at the time of retrieval.
- The performance of attributes is the same as for members on a stored hierarchy.
- The higher the association level of an attribute to the base member, the faster the retrieval query.

Attribute Dimensions and ASO Cubes

This topic provides information on the differences between aggregate storage and block storage cubes with regard to attributes. To use the information in this topic, you should be already familiar with attribute concepts for block storage. See Working with Attributes.

The following information applies to attribute dimensions when used on aggregate storage cubes:

- Only the addition (+) consolidation operator is available within attribute dimensions.
- For a given attribute dimension, all associations must be with one level of the base dimension. For example, in the ASOsamp.Basic cube, associations for the Store Manager attribute dimension are with level 0 of the Stores dimension. The following restrictions apply to attribute associations:

- Level 0: You can associate attributes with any level 0 member of a dynamic or stored hierarchy that does not have a formula.
- Non-level 0: You can associate attributes only to upper level members in the primary stored hierarchy.

Attribute dimensions do not have hierarchy types. You cannot specify an attribute dimension as a dynamic or stored hierarchy. Essbase treats attribute dimensions as stored alternate hierarchies of the base dimension. For example, in the ASOsamp.Basic database, Essbase treats the Store Manager attribute dimension as if the Store Manager dimension were a stored alternate hierarchy of the Stores dimension.

When query tracking is enabled, Essbase considers queries on attribute dimension data, and may include attribute dimension members in aggregate view selections. Refer to Select Views Based on Usage.

> **Note:**
>
> Queries on attribute members that are associated with non-level 0 members return values for descendants of the non-level 0 member. This behavior of queries on attribute members in aggregate storage cubes is different from the behavior in block storage cubes.

Attribute Queries on ASO Cubes

When selecting and building views based on attribute query data, some queries on attribute data are always dynamically calculated at the time of retrieval, which may affect query performance.

Every query involving attribute dimension members must also include at least one member from the base dimension. If the query involves a single attribute dimension and a sum-of-all dimension member, Essbase aggregates the query data, potentially improving query performance. In other cases, Essbase must calculate the query at the time of retrieval.

The following table describes attribute query types and how Essbase calculates the query:

Table 37-2 Attribute Queries and Calculation Performance

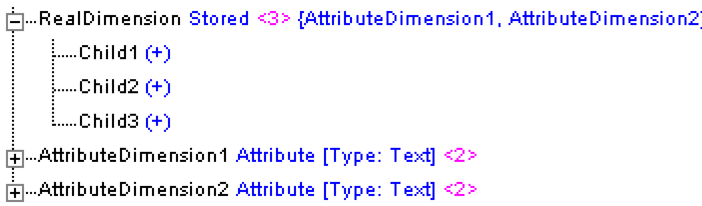
Attribute Query Type	Query Calculation Type
Query involves a sum-of-all base dimension member and members from one attribute dimension.	Essbase can aggregate query data, potentially improving query performance.
Query involves any member of the base dimension and members from multiple attribute dimensions.	Essbase calculates the query at the time of retrieval based on the level 0 input data.
Query involves any child member of the base dimension member (or dimension member that is tagged as label-only) and members from one attribute dimension.	Essbase calculates the query at the time of retrieval based on the level 0 input data, or on data from aggregations on the base dimension.

In the outline illustration below, RealDimension is the sum of all its descendents (it is not tagged as label-only). If a query involves one or more members from a single attribute dimension (for example, AttributeDimension1), crossed with the base dimension member (RealDimension), Essbase can build aggregate cells for the data, potentially improving query performance.

The following queries, however, are always calculated at the time of retrieval:

- Any query requesting data for members from an attribute dimension (for example AttributeDimension1) and any of the children of RealDimension is calculated dynamically at retrieval time based on the level 0 input data or on data from aggregations.
- Any query requesting data from multiple attribute dimensions (for example AttributeDimension1 and AttributeDimension2) and a base member dimension (for example RealDimension) is calculated dynamically at retrieval time based on level 0 input data.

Figure 37-4 Outline for Attribute Query Example



Attribute Queries and Multiple Hierarchies

When querying data from a dimension that has multiple hierarchies, query performance may improve if you query the data in the following way:

1. Select the hierarchy that you want to query.
2. Navigate to find the detailed data (for example, by zooming in on the hierarchy in Smart View).

Including dynamic hierarchy members and stored hierarchy members in the same query may require a large internal memory cache, which decreases query performance.

64-bit Dimension Size Limit for Aggregate Storage Outline

An Essbase aggregate storage (ASO) cube outline cannot exceed 64-bits per dimension.

The number of bits needed by a dimension is the maximum number of bits used by any level 0 child, including the level 0 children in alternate hierarchies and associated attribute dimensions. For the purposes of member numbering, attribute dimensions are treated as alternate hierarchies of their base dimensions.

In general, the formula to determine the number of bits required for any member in a dimension can be expressed as:

```
#_bits_member’s_parent + log(x)
```

where *x* is the number of children of the parent.

For example, if the member’s parent is member A, which requires 5 bits, and A has 10 children, the number of bits required by each child is:

```
5 +log(10) = 9 bits
```

The top member of a dimension or hierarchy usually uses 0 bits. However, when one or more top generations consist of label-only members, the label-only members do not receive member numbers (because they are not considered stored members). Therefore, if there are *x* members in the first non-label-only generation, those members use log(*x*) bits. The rest of the children below them are numbered normally.

Similarly, if a dimension or hierarchy is dynamic, only the level 0 members that are stored or shared receive member numbers. The number of bits required for those members is log(*x*), where *x* is the number of level 0 members that are stored or shared (that is, the number of level 0 members that are not formula members).

If, however, any alternate hierarchies have stored (non-shared) level 0 members, each member of every hierarchy in the dimension (including associated attribute dimensions) uses an extra log(*x*) bit, where *x* is the total number of hierarchies and associated attribute dimensions for this base dimension.

The following example uses the Products dimension in the ASOsamp.Basic database:

The Products dimension has two hierarchies: All Merchandise and High End Merchandise, which is an alternate hierarchy. High End Merchandise has one stored level 0 member: Stored Member. The Products dimension does not have any associated attribute dimensions.

Members All Merchandise and High End Merchandise use log(2) = 1 bit.

Note:

If the alternate hierarchy High End Merchandise did not have any stored level 0 members, the top members of each hierarchy (and associated attribute dimensions) would each use 0 bits.

The calculation of the number of bits required by each level 0 children:

```
All Merchandise = 1 bit
   Personal Electronics, Home Entertainment, Other = 1 + log(3) = 3 bits
      Digital Cameras/Camcorders, Handhelds/PDAs, Portable Audio = 3 + log(3)
= 5
         Children of Digital Cameras/Camcorders = 5 + log(3) = 7
         Children of Handhelds/PDAs = 5 + log(3) = 7
         Children of Portable Audio = 5 + log(2) = 6
      Televisions, Home Audio/Video = 3 + log(2) = 4
         Children of Televisions = 4 + log(5) = 7

         Children of Home Audio/Video = 4 + log(4) = 6
      Computers and Peripherals = 3 + log(1) = 3 **
      Systems, Displays, CD/DVD drives = 3 + log(3) = 5
         Children of Systems = 5 + log(2) = 6
   High End Merchandise = 1 bit
      Flat Panel, HDTV, Stored Member = 1 + log(3) = 3 bits
```

Member Computers and Peripherals has the same number of bits (3) as its parent Other.

The maximum bits used by any level 0 children in the Products dimension is 7 (Children of Digital Cameras and Children of Televisions). Therefore, Products uses 7 bits, which is less than the dimension size limit of 64 bits.

If the dimension size exceeds 64 bits:

- Essbase generates the following error when saving the outline:

  ```
  Hierarchy [DimensionName] is too complex. It exceeds the maximum member
  number width of 64 bits. See application log for details.
  ```

- Essbase logs messages similar to the following messages in the application log:

  ```
  Member number for member [level0member] requires [65] bits to encode
  Member [level0member] contributes [5] bits to member number
  Member [level1parent] contributes [20] bits to member number
  Member [level2parent] contributes [20] bits to member number
  Member [level3parent] contributes [20] bits to member number
  ```

To fix the error, use one of these recommendations:

- If possible, delete some siblings of any of the members referenced in the messages. Reducing the number of siblings by a power of two saves one bit. For instance, assume that level0member, which contributes 5 bits to the member number, has 18 siblings, including itself. Reducing the number of siblings to 16 or fewer saves one bit because log(16) = 4. Similarly, reducing the number of siblings to 8 or fewer saves two bits.
- Reclassify some siblings of members referenced in the messages. For example, move half of level0member's 18 siblings to another parent that doesn't have as many children. Alternately, create a new parent as a sibling of level1parent and move half of level1parent's children under the new member. This approach saves one bit.
- Combine some intermediate levels. For instance, move level0member, and all of its siblings, to be children of level2parent and then remove level1parent. This approach is more involved but it can save many bits.

Compression Dimension for Aggregate Storage Cubes

The choice of a compression dimension for the Essbase aggregate storage (ASO) cube can significantly affect performance. By default, it is the Accounts dimension. A good candidate is a dimension that optimizes data compression while maintaining retrieval performance.

Changing the compression dimension triggers a full restructure of the cube. The compression dimension must be a single, dynamic hierarchy. If the dimension has a different hierarchy setting, such as multiple hierarchies, it will be set to single dynamic hierarchy automatically. The original hierarchy setting is lost (setting a different dimension as compression does not return the original hierarchy setting). Attribute dimensions cannot be compression dimensions, nor can dimensions with attributes associated to them.

> **Note:**
>
> The information in this topic applies to loaded databases. See Load Data into Aggregate Storage Cubes.

Maintaining Retrieval Performance

Because compression dimensions are dynamically calculated, you must take into account design considerations for dynamically calculated dimensions when choosing a compression dimension. Dynamic dimensions are calculated at the time of retrieval, so the data retrieval time is longer than for stored hierarchies.

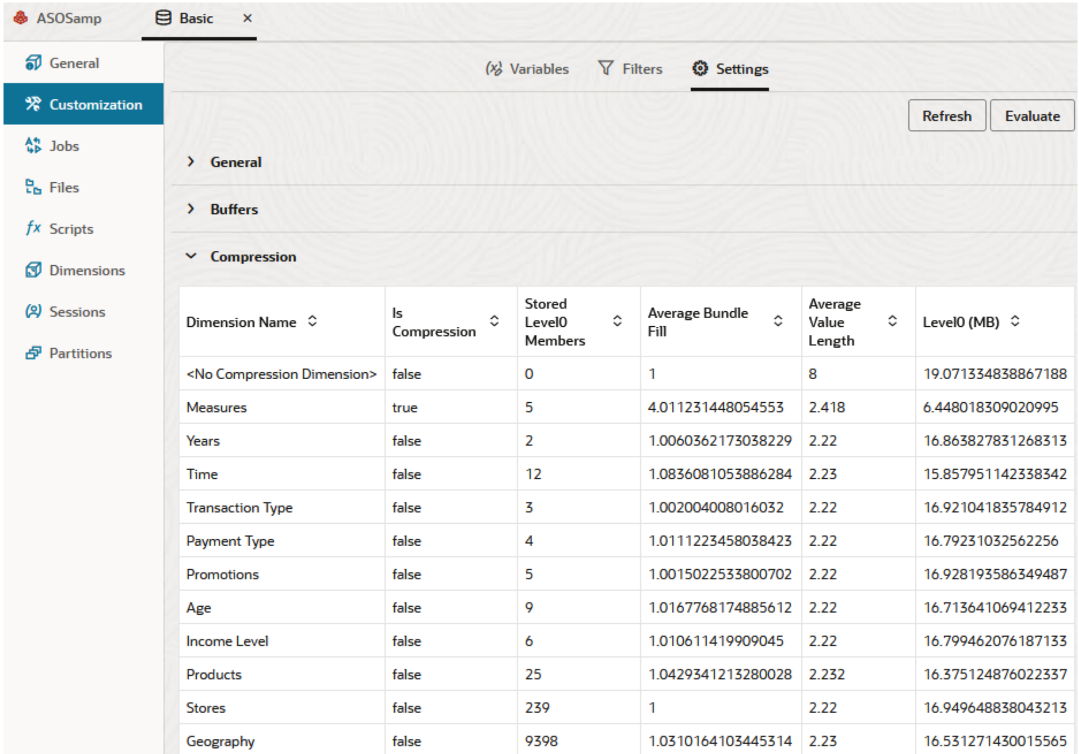
If a dimension with a large number of level 0 members is tagged as compression, upper-level queries take longer because they require many level 0 members to be retrieved. If users will be doing many upper-level retrievals on a large dimension, it is not a good candidate for a compression dimension.

View Compression Estimation Statistics

You can view detailed compression and query statistics in the Essbase web interface, MaxL, or REST API. These estimates can help you choose the best dimension to use as the compression dimension. For each dimension you can view the number of stored level 0 members, which affects retrieval performance; the average bundle fill and average value length, which affect compression; and the level 0 size.

To view the statistics, select one of the following methods:

- The Essbase web interface. Example:



Dimension Name	Is Compression	Stored Level0 Members	Average Bundle Fill	Average Value Length	Level0 (MB)
<No Compression Dimension>	false	0	1	8	19.071334838867188
Measures	true	5	4.011231448054553	2.418	6.448018309020995
Years	false	2	1.0060362173038229	2.22	16.863827831268313
Time	false	12	1.0836081053886284	2.23	15.857951142338342
Transaction Type	false	3	1.002004008016032	2.22	16.921041835784912
Payment Type	false	4	1.0111223458038423	2.22	16.79231032562256
Promotions	false	5	1.0015022533800702	2.22	16.928193586349487
Age	false	9	1.0167768174885612	2.22	16.713641069412233
Income Level	false	6	1.010611419909045	2.22	16.799462076187133
Products	false	25	1.0429341213280028	2.232	16.375124876022337
Stores	false	239	1	2.22	16.949648838043213
Geography	false	9398	1.0310164103445314	2.23	16.531271430015565

- The aggregate storage version of the query database MaxL statement. Example:

```
query database ASOSamp.Basic list aggregate_storage compression_info;
```

- The REST API endpoint **Get ASO Compression Info**.

See the following descriptions of each of the compression and query related statistics.

Stored level 0 members

Dimensions with a large number of stored level 0 members do not perform well if tagged Compression. As with any dynamically calculated dimension, upper-level retrievals from compression dimensions generally are slow.

Average bundle fill

Compression is more effective if values are grouped together in consecutive members on dimensions or hierarchies rather than spread throughout the outline with lots of `#MISSING` data between values. Essbase saves memory by storing information about the location and contents of the groups rather than storing it separately for each of the members. The average bundle fill is the average number of values stored in the groups. It can vary between 1 and 16, with 16 being the best. Choosing a compression dimension that has a higher average bundle fill means that the database compresses better.

In some outlines, you can improve compression by ordering the numbers in the compression dimension so that members that are frequently populated are grouped together. When populated members are grouped together, more values fit into each bundle, increasing the average bundle fill and improving compression.

Average value length

The average value length is the average storage size, in bytes, required for the stored values in the cells. It can vary between 2 bytes and 8 bytes with 2 bytes being the best. Without compression, it takes 8 bytes to store a value in a cell. With compression, it can take fewer bytes, depending on the value length. For example, 10.050001 might take 8 bytes to store even when compressed, but 10.05 may only take 2 bytes (4 bytes to store when compressed). Dimensions with a smaller average value length compress the database better.

Rounding the data values to no more that two digits after the decimal point can reduce the average value length, improving compression.

Expected level 0 size

This field indicates the estimated size of the compressed database. A smaller expected level 0 size indicates that choosing this dimension is expected to enable better compression.

Verifying Aggregate Storage Outlines

Aggregate storage outline files have the same file extension (`.otl`) as block storage database outline files and are stored in an equivalent directory structure.

When you save an outline, Essbase verifies it for errors. You can also verify the accuracy of an outline before you save it. Some block storage database features do not apply to aggregate storage databases, and the verification process considers the rules for aggregate storage databases. See Comparison of Aggregate and Block Storage.

Outline Paging in Aggregate Storage Cubes

Aggregate storage (ASO) cube outlines are pageable, significantly reducing memory usage for very large cubes. Essbase preloads part of the outline into memory. During data retrieval, Essbase pages other parts of the outline into memory as needed.

When you create an aggregate storage cube, the outline is created in a pageable format.

Paging an outline into memory enables Essbase to handle very large outlines (for example, 10 million or more members), but potentially increases data retrieval time.

> **Note:**
>
> Aggregate storage cubes that have pageable outlines contain memory pages, and therefore their outline files may be larger than binary block storage outline files.

Outline Paging Limits

The maximum size of a buildable outline (the number of members) depends on several factors:

- The available memory for Essbase
- The amount of memory in Essbase allocated for other uses
- The amount of memory required for each member (and aliases for each member)

Essbase uses about 40 MB of memory on startup. In addition, the various caches require the following memory allocations:

- Outline paging cache: 8 MB
- Aggregate storage data cache: 32 MB
- Aggregate storage aggregation cache: 10 MB

Therefore, the initial memory footprint for Essbase is about 90 MB. In addition, memory must be allocated to process incoming query requests. Typical memory to reserve for this purpose is about 300 MB. The total memory allocated for Essbase is therefore 390 MB.

On a Windows system with 1.85 GB of addressable memory, the amount available to build and load the outline is about 1.46 GB (1.85 GB - 390 MB = 1.46 GB).

The maximum outline size depends on whether it is built using a dimension build or from an outline already loaded into Essbase.

Dimension Build Outline Paging Limit

To build the outline by using a dimension build, Essbase allocates about 100 bytes per member, plus the size of the member name, plus the size of all alias names for the member (up to 10 aliases are allowed).

For a sample outline (using a single byte codepage) where the average member name is 15 characters and there is one alias (of 20 characters) per member, the memory requirement for each member that is added:

100 + 15 + 20 bytes = 135 bytes

The total number of members that can be added in a dimension build is the available memory (1.46 GB, or 153,092,060 bytes) divided by the number of bytes per member (135), approximately 11 million members.

On systems with more than 2 GB of addressable memory, the outline can be larger in proportion to the extra memory that is available.

When the dimension build is complete, a *dbname*`.otn` file is saved in the cube directory. The `.otn` file is used as input for the outline restructuring process, which replaces the old outline with the new one. During restructuring, two copies of the outline are loaded into memory, the old one (potentially empty), and the new one, so the maximum size of an outline that can be restructured depends on the size of the old outline.

In a dimension build, which starts with an empty outline, only one outline is loaded into memory.

Loaded Outline Paging Limit

The memory requirement for an outline loaded into Essbase at runtime or during restructuring is different from the memory requirements for a dimension build. Essbase allocates about 60 bytes per member, plus the size of the member name plus 5 bytes, plus the size of all alias names for the member (up to 10 aliases are allowed) plus 5 bytes. For a sample outline where the average member name is 15 characters and there is one alias (of 20 characters) per member, the memory requirement for each member that is added:

60 + 15 + 5 + 20 + 5 bytes = 105 bytes per member

Assuming 1.46 GB of available memory, the maximum size of an outline that can be loaded is one with 14 million members (1.46 GB/105 bytes).

The 14 million members are the sum of two outlines that are loaded during restructuring. For example, if an existing outline has 5 million members, the new outline can have a maximum of 9 million members. In an incremental dimension build, it is recommended to build the smaller dimensions first and the larger ones last to allow for a maximum outline size.

Compacting the Aggregate Storage Outline File

The outline file (`.otl`) for an aggregated storage outline increases in size as the outline changes; for example, when members are added or deleted. You can decrease the outline file size by compacting it. After the outline file is compacted, the file continues to grow as before, when members are added or deleted.

Compacting the outline file causes Essbase to restructure the outline and can take place only when no other users or processes are actively using the database. Compacting the outline does not cause Essbase to clear the data.

> **Note:**
>
> When a member is deleted from the outline, the corresponding record of that member in the outline file is marked as deleted but the record remains in the outline file. Compacting the outline file does not remove the records of deleted members.

To compact an outline file, you can use the alter database MaxL statement.

Developing Formulas on Aggregate Storage Outlines

Formulas calculate relationships between members in an Essbase outline. Aggregate storage (ASO) formulas are different than block storage formulas. You write aggregate storage (ASO) formulas in MDX as numeric value expressions, and apply them to the cube outline, where they are dynamically calculated upon data retrieval.

DIfferences Between ASO and BSO Formulas

If you are familiar with formulas on block storage (BSO) outlines, note the differences with formulas in aggregate storage (ASO).

- Essbase provides a native calculation language to write formulas on block storage outlines. To write formulas for aggregate storage outlines, use the MDX (Multidimensional Expressions) language.
- Apply formulas directly to members in the outline. For block storage cubes, formulas can be placed in a calculation script. For aggregate storage cubes, you cannot place formulas in a calculation script.

Formula Calculation for ASO Cubes

Essbase calculates formulas in aggregate storage outlines only when data is retrieved. Calculation order may affect calculation results. Whenever you use MDX formulas on multiple dimensions in an aggregate storage outline, it is good practice to set the solve order for each member or dimension. See Calculation Order and Solve Order in ASO Cubes.

> **Note:**
>
> When designing an aggregate storage calculation, consider that aggregate storage members with MDX formulas are dynamically calculated. The dynamically calculated members have a value of #MISSING until they are queried.

Using MDX Formulas

An MDX formula must always be an MDX numeric value expression. In MDX, a numeric value expression is any combination of functions, operators, and member names that does one of the following actions:

- Calculates a value
- Tests for a condition
- Performs a mathematical operation

A numeric value expression is different from a set expression. A set expression is used on query axes and describes members and member combinations. A numeric value expression specifies a value.

A numeric value expression is used in queries to build calculated members, which are logical members created for analytical purposes in the WITH section of the query, but which do not exist in the outline.

The following query defines a calculated member and uses a numeric value expression to provide a value for it:

```
WITH MEMBER
 [Measures].[Prod Count]
AS
 'Count (
    Crossjoin (
     {[Units]},
     {[Products].children}
    )
  )', SOLVE_ORDER=1
SELECT
 {[Geography].children}
ON COLUMNS,
 {
  Crossjoin (
     {[Units]},
     {[Products].children}
    ),
   ([Measures].[Prod Count], [Products])
 }
ON ROWS
FROM
 ASOsamp.Sample
```

In the sample query, the WITH clause defines a calculated member, Product Count, in the Measures dimension, as follows:

```
WITH MEMBER
 [Measures].[Prod Count]
```

The numeric value expression follows the WITH clause and is enclosed in single quotation marks. In the sample query, the numeric value expression is specified as follows:

```
'Count (
    Crossjoin (
     {[Units]},
     {[Products].children}
    )
  )'
```

The SOLVE_ORDER property specifies the order in which members and formulas are evaluated. See Calculation Order and Solve Order in ASO Cubes.

> **Note:**
>
> For an explanation of the syntax rules used to build the numeric value expression in the example, see the documentation for the Count, CrossJoin, and Children functions.

A numeric value expression also can be used as an MDX formula to calculate the value of an existing outline member.

Therefore, rather than creating the example query, you can create an outline member on the Measures dimension called Prod Count that is calculated in the outline in the same way that the hypothetical Prod Count was calculated in the sample query.

To create a calculated member with a formula:

1. Create a member.
2. Attach an MDX formula to the member.

   Assuming that you created the example Prod Count member, you would use the following formula, which is the equivalent of the numeric value expression used to create the calculated member in the example query:

   ```
   Count(Crossjoin ( {[Units]}, {[Products].children}))
   ```

3. Verify the formula by verifying the outline.

   When you retrieve data from the aggregate storage cube, the formula is used to calculate the member value.

   You can use substitution variables within formulas. For example, you could define a substitution variable named “EstimatedPercent” and provide different percentages as substitution variable values. See Using Substitution Variables.

Before applying formulas to members in the outline, you can write MDX queries that contain calculated members. When you can write an MDX query that returns the calculated member results that you want, you are ready to apply the logic of the numeric value expression to an outline member and validate and test the expression. See Writing MDX Queries.

Formula Syntax for Aggregate Storage Databases

When you create member formulas for aggregate storage outlines, observe the following rules:

- Enclose member names in brackets (`[]`) if they meet any of the following conditions:
  - Start with a number or contains spaces; for example, `[100]`. Brackets are recommended for all member names, for clarity and code readability.
  - Are the same as an operator or function name.
  - Include a nonalphanumeric character; for example, a hyphen (`-`), an asterisk (`*`), or a slash (`/`).

> **Note:**
>
> In formulas, member names starting with `$` or `&` must be enclosed in quotation marks as well as brackets. For example, `$testmember` would be expressed in the formula as `["$testmember"]/100`

- Use the IIF function to write conditional tests with a single else condition. The syntax for the IIF function does not require an ELSEIF keyword to identify the else condition nor an ENDIF keyword to terminate the statement. You can nest IIF functions to create a more complex formula.
- Use the Case, WHEN, THEN construct to write conditional tests with multiple conditions.
- Be certain that tuples are specified correctly. A tuple is a collection of members with the restriction that no two members can be from the same dimension. Enclose tuples in parentheses; for example, `(Actual, Sales)`.
- Be certain that sets are specified correctly. A set is an ordered collection of one or more tuples. When a set has multiple tuples, the following rule applies: In each tuple of the set, members must represent the same dimensions as do the members of other tuples of the set. Additionally, the dimensions must be represented in the same order. In other words, all tuples of the set must have the same *dimensionality*.

  See also MDX Sets and Tuples.

  Enclose sets in braces, for example:

  ```
  { [Year].[Qtr1], [Year].[Qtr2], [Year].[Qtr3], [Year].[Qtr4] }
  ```

Writing Formulas on Aggregate Storage Outlines

Write MDX formulas on Essbase aggregate storage (ASO) outlines using the formula editor. Formulas enable you to apply basic mathematical equations, calculate members across dimensions, perform conditional tests, and filter based on user defined attributes (UDAs).

Composing Formulas on Aggregate Storage Outlines

Formulas are plain text. You can type the formulas directly into the formula editor, or you can create a formula in any text editor and paste the text into the formula editor.

MDX-based syntax checking tells you about syntax errors in formulas; for example, a mistyped function name or a nonexistent member. Unknown member names can be validated against a list of function names. If you are not connected to the service instance or to the application associated with the outline, Essbase may connect you to validate unknown names.

Syntax checking occurs when validate or save a formula. Errors are displayed in the user interface. If an error occurs, you choose to save or not save the formula. If you save a formula with errors, you are warned when you save the outline. When you calculate a formula with errors, the formula is ignored and the member is given a value of $MISSING.

A syntax checker cannot warn you of semantic errors in a formula. Semantic errors occur when a formula does not work as you expect. One way to find semantic errors in a formula is to place the numeric value expression that defines the formula into an MDX query and run it, to verify that the results are as you expect.

You can include formulas in a dimension build data source. See Set Dimension Build Field Type Information.

Basic Equations for Aggregate Storage Outlines

You can apply a mathematical operation to a formula to create a basic equation. For example, the following formula is applied to the Avg Units/Transaction member in the ASOsamp.Sample cube:

```
[Units]/[Transactions]
```

The formula in Avg Units/Transaction divides the number of units by the number of transactions to arrive at the average number of units per transaction.

In aggregate storage outlines, members cannot be tagged as expense items. Therefore, functions in Calc, such as @VAR and @VARPER, which determine the difference between two members by considering expense tags, are not relevant in aggregate storage outlines.

The MDX subtraction operator can be used to calculate the difference between two members. For example, the following formula can be applied to a new member, called Price Diff, in ASOsamp.Sample, to calculate the difference between the price paid and the original price:

```
[Price Paid]-[Original Price]
```

Members Across Dimensions in Aggregate Storage Outlines

ASOsamp.Sample provides a formula on a member called % of Total. This member formula identifies the percentage of the Measures total that is produced by Transactions. The formula for % of Total:

```
Transactions/
(Transactions,Years,Months,[Transaction Type],[Payment Type],
Promotions,Age,[Income Level],Products,Stores,Geography)
```

The formula specifies a member (Transactions) divided by a tuple (Transactions, Years, ...). The formula lists a top member from every dimension to account for all Transaction data in the cube; that is, not Transaction data for the Curr Year member but Transaction data for all members of the Years dimension, not Transaction data for months in the first two quarters but Transaction for all months, and so on. In this way, the value of % of Total represents the percentage of the Measures total that are produced by Transactions.

Conditional Tests in Formulas for Aggregate Storage Outlines

You can define a formula that uses a conditional test or a series of conditional tests to determine the value for a member. Use the IIF function to perform a test with one else condition. You can nest IIF functions to create a more complex query.

The example specifies a formula for a member that represents the price the company must pay for credit card transactions, which includes a 5% charge. The following example assumes that the Credit Price member has been added to the Measures dimension of the ASOsamp.Sample cube. Credit Price has the following formula, which adds 5% to Price Paid when the payment type is a credit card.

```
IIF (
   [Payment Type].CurrentMember=[Credit Card],
   [Price Paid] * 1.05, [Price Paid]
)
```

Use the CASE, WHEN, THEN construct to create formulas with multiple tests and else conditions.

The Filter function returns the tuples of the input set that meet the criteria of the specified search condition. For example, to establish a baseline (100) for all products, you can add a Baseline member and create a formula for it, as follows:

```
Count(Filter(Descendants([PersonalElectronics],
[Products].Levels(0)),[Qtr1] > 100.00))
```

Specifying UDAs in Formulas in Aggregate Storage Outlines

UDAs are words or phrases that you create for a member. For example, in Sample.Basic, top-level members of the Market dimension have the UDA Small Market or the UDA Major Market.

The Major Market example used in this topic shows how to create a formula for a member that shows the sum of sales for all major market members. The example assumes that a new member (Major Market Total) has been added to Sample.Basic.

1. MDX provides a Boolean function, IsUDA, which Returns TRUE if a member has the associated UDA tag. The following syntax returns TRUE if the current member of the Market dimension has the UDA “Major Market”:

   ```
   IsUda([Market].CurrentMember, "Major Market")
   ```

2. A Filter function, when used with IsUDA (as shown in the following syntax), cycles through each member of the Market dimension and returns a value for each member that has the Major Market UDA:

   ```
   Filter([Market].Members, IsUda([Market].CurrentMember, "Major Market"))
   ```

3. The Sum function adds the values returned by the Filter function; for the Major Market example, the following formula is produced:

   ```
   Sum (Filter([Market].Members, IsUda([Market].CurrentMember, "Major
   Market")))
   ```

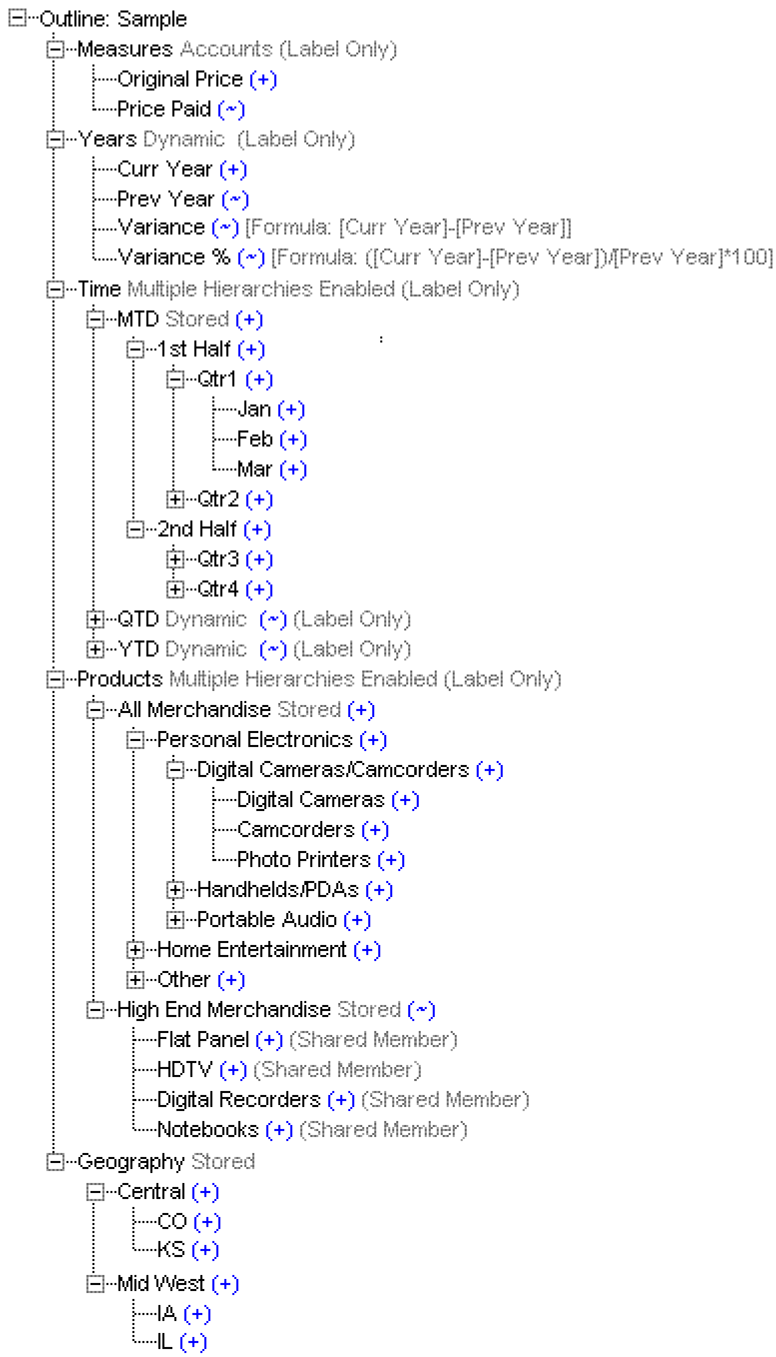
   This formula is attached to the Major Market Total member.

38 Load, Calculate, and Retrieve Aggregate Storage Data

The most common tasks for Essbase aggregate storage (ASO) cubes involve maintaining the outline, loading and calculating data values, and retrieval. These tasks are different for ASO cubes than for block storage cubes.

Examples in this chapter refer to the aggregate storage outline for the ASOSamp.Basic cube, illustrated below.

Figure 38-1 ASOSamp.Basic Aggregate Storage Outline



The simplified aggregate storage outline is not completely expanded. A plus sign (+) node at the left of a member name indicates that the member has children that are not displayed.

Build Dimensions in Aggregate Storage Cubes

When you build dimensions in Essbase aggregate storage (ASO) cubes, be aware that changes to hierarchies will trigger a restructure, and can lead to clearing of aggregate views and of of data values. Incremental dimension building is recommended to save time.

Aggregate storage dimension build changes to the outline can result in all aggregate views or all data values being cleared from the cube when the dimension build is finished. Aggregate Storage Cube Restructuring describes the results of outline changes.

If you use multiple data sources to build dimensions, you can save processing time by performing an incremental dimension build. Incremental dimension builds enable you to defer restructuring until all data sources have been processed.

For information about incremental dimension build, see Incremental Dimension Builds.

Differences between outline characteristics of block storage outlines and aggregate storage outlines affect data sources and rules files. For example, defining a dimension as sparse or dense is not relevant to aggregate storage outlines.

When using the aggregate storage data load buffer, you can combine data sources and rules files to add members to the outline and to load data values to the level 0 cells. Regardless of the order in which you specify the files, Essbase makes the outline changes and then loads the data values.

> **Note:**
>
> Oracle recommends that you separate dimension build operations from data load operations, because some dimension builds clear data, which could lead to data loss. See Build Dimensions in Aggregate Storage Cubes.

Rules File Differences for Aggregate Storage Dimension Builds

Rules files for building aggregate storage outlines must define only outline properties and field types that apply to aggregate storage outlines.

After converting a block storage outline to aggregate storage, update the rules files by associating them to the aggregate storage version of the outline.

Field Type Differences

The field contains a number (0–127) that specifies the order in which the member is evaluated in the outline. Values less than 0 or greater than 127 are reset to 0 and 127, respectively. No warning message is displayed. For information on the solve order property, see Calculation Order and Solve Order in ASO Cubes.

Valid build methods for solve order:

- Generation
- Level
- Parent-child references

Data Source Differences for Aggregate Storage Dimension Builds

The following table lists the member codes that are recognized in dimension build data sources as properties for members of aggregate storage database outlines. Any other consolidation code is ignored and + (add) is assumed. (Data sources for modifying aggregate storage outlines should not include field values that apply only to block storage outlines.)

Table 38-1 Consolidation Codes for Members of Aggregate Storage Outlines

Code	Description
%	Expresses as a percentage of the current total in a consolidation (applies only to members of a dynamic hierarchy)
*	Multiplies by the current total in a consolidation (applies only to members of a dynamic hierarchy)
+	Adds to the current total in a consolidation (applies only to members of a dynamic hierarchy)
-	Subtracts from the current total in a consolidation (applies only to members of a dynamic hierarchy)
/	Divides by the current total in a consolidation (applies only to members of a dynamic hierarchy)
~	Excludes from the consolidation (applies only to members of a dynamic hierarchy or members beneath Label Only members in a stored hierarchy)
C	Set member as top of a stored hierarchy (applies to dimension member or generation 2 member)
D	Set member as top of a dynamic hierarchy (applies to dimension member or generation 2 member)
H	Set dimension member as multiple hierarchies enabled (applies to dimension member only)
K	Reset the time balance property to NONE (applies to accounts dimensions only).
N	Never allow data sharing
O	Tag as label only
P	Reset the time balance skip option to NONE (applies to accounts dimensions only).

Currency name and currency category field types are not supported for aggregate storage outlines.

In aggregate storage outlines, formulas must be specified in the same format as MDX numeric value expressions.

Building Alternate Hierarchies in Aggregate Storage Databases

To build shared members in an aggregate storage outline, make these selections in the Rules Editor:

- On the Properties tab, select **Auto Configuration**
- On the Dimensions tab, under **Dimension Properties**, select **Share** on the General tab

When auto configuration and sharing are enabled, duplicate members are automatically created under a new parent as shared members.

> **Note:**
>
> There are restrictions on using the duplicate generation (DUPGEN) method to build alternate hierarchies in aggregate storage outlines.

> **Caution:**
>
> In alternate hierarchies in aggregate storage databases, you can associate attributes only with level-0 members.

Exclusive Operations on Aggregate Storage Cubes

Some operations on Essbase aggregate storage (ASO) cubes are exclusive, meaning they cannot run concurrently with others of the same type of operations. Queries against the cube can continue while exclusive operations are in progress.

On aggregate storage cubes, multiple exclusive operations cannot be performed simultaneously on the same cube. If one exclusive operation is running and a second exclusive operation is attempted simultaneously, the second operation is rejected with a message indicating that the operation cannot proceed while the first operation is in progress. For example, when performing a partial data clear, a second partial data clear cannot run concurrently, even if the operations are clearing different regions. While most exclusive operations are mutually exclusive, there are some exceptions.

Exclusive operations and exceptions:

- Export

  Multiple export operations can run at the same time because export is a read-only operation. Export operations can run at the same time as build aggregations and backup, both of which are exclusive operations; however, export is not compatible with any other exclusive operation.
- Build aggregations
- Backup (putting the cube in read-only archive mode)
- Data load (committing the contents of a load buffer to the cube)

  Creating an aggregate storage load buffer and loading data into the load buffer are not exclusive operations. These operations can run concurrently with any other operations. However, committing the data in the load buffer to the cube is an exclusive operation.
- Spreadsheet send operations (for example, updating cells)

  If a send operation is running while another exclusive operation is attempted, the new operation waits for the send operation to finish before proceeding. Even though not compatible with the send operation, the new operation does not error out, because send operations are assumed to be small and fast (<5 seconds). This means it is possible for many users to perform spreadsheet send operations at the same time without those operations being rejected because they are incompatible with each other.

> **Note:**
>
> In the case where multiple exclusive operations are attempted while a send operation is running, the order in which the new exclusive operations execute after the send operation completes is random; the order is not based on the sequence in which the new exclusive operations were attempted. For example, if a send operation is running and an exclusive operation of a different type is attempted, the new exclusive operation waits for the send operation to finish before proceeding. If, in the meantime, more send operations are attempted by other users, those send operations might be executed before the other exclusive operation, even though they were attempted afterward. Therefore, the exclusive operation might wait indefinitely as long as there is at least one send operation waiting to be executed.

- Merge slices
- Custom calculations, allocations, and other write operations
- Data clear operations (full, aggregates only, and partial)

Queries are allowed to run concurrently with all exclusive operations. However, if an operation adds, changes, or removes any data in the cube, the following sequence takes place at the end of the operation, when the changes are made visible to queries:

1. Any new queries are temporarily blocked from starting (the queries wait).
2. Existing queries finish running.
3. Data changes from the exclusive operation are committed to the cube.
4. Queries that are waiting proceed.

Queries are never rejected or canceled because of an operation that changes data on an aggregate storage cube.

Load Data into Aggregate Storage Cubes

Essbase aggregate storage (ASO) cubes facilitate analysis of very large dimensions containing up to a million or more members. To help you load data to such large outlines, you can load incrementally, manage the data load buffers, and merge/replace slices of data.

To efficiently support loading data values into such large cubes, Essbase:

- Allows processing multiple sources of data through temporary data load buffers
- Allows you to control the percentage of resources a data load buffer uses
- Allows an aggregate storage cube to contain multiple slices of data (a query to the database accesses each slice, collecting all of the data cells)
- Provides an incremental data load process that completes in a length of time that is proportional to the size of the incremental data

To load values to aggregate storage cubes, you can use the Jobs page in the Essbase web interface, or you can use the alter database and import data statements in MaxL. Examples in this document are based on using MaxL.

> **Note:**
>
> If values have been calculated and stored through an aggregation, Essbase automatically updates higher-level stored values when data values are changed. No additional calculation step is necessary. The existence and size of an aggregation can affect the time it takes to perform a data load.
>
> You cannot export data while loading data into a cube.
>
> When copying an ASO application, to retain all of the data in the cube, you must merge all incremental data slices into the main database slice before you copy the application. Data in unmerged incremental data slices is not copied.

Load Data Incrementally through a Data Load Buffer

If you use incremental data load to load data values to an Essbase aggregate storage (ASO) cube, you can improve performance. Essbase loads the values to a temporary data load buffer first, with a final write to storage after all data sources have been read.

Using the import data MaxL statement to load data values from a single data source does not involve the aggregate storage data load buffer.

If you use multiple **import database data** MaxL statements to load data values to aggregate storage cubes, Essbase can utilize a temporary data load buffer, completing the final write to storage after all data sources have been read. Using the aggregate storage data load buffer can significantly improve overall data load performance.

In the aggregate storage data load buffer, Essbase sorts and commits the values after all data sources have been read. If multiple (or duplicate) records are encountered for any specific data cell, the values are accumulated. Essbase then stores the accumulated values—replacing, adding to, or subtracting from existing data values in the cube.

> **Note:**
>
> When using the aggregate storage data load buffer, the choice for replacing, adding, or subtracting values is specified for the entire set of data sources when loading the data buffer contents to the cube.

While the data load buffer exists in memory, you cannot build aggregations or merge slices, because these operations are resource-intensive. You can, however, load data to other data load buffers, and perform queries and other operations on the cube. There might be a brief wait for queries, until the full data set is committed and aggregations are created.

The data load buffer exists in memory until the buffer contents are committed to the cube or the application is restarted, at which time the buffer is destroyed. Even if the commit operation fails, the buffer is destroyed and the data is not loaded into the cube. You can manually destroy a data load buffer by using the alter database MaxL statement.

> **Note:**
>
> Stopping the application before committing the buffer contents destroys the buffer. In this situation, after restarting the application, you must initialize a new buffer and load the data to it.

To use the data load buffer for aggregate storage cubes:

1. Prepare the data load buffer, where data values are sorted and accumulated by using the alter database MaxL statement to initialize an aggregate storage data load buffer. For example:

   ```
   alter database ASOsamp.Basic
      initialize load_buffer with buffer_id 1;
   ```

2. Load data from your data sources into the data load buffer using the **import database** MaxL statement. Use multiple statements to load data from multiple data sources. You can include any combination of data sources. Specify a rules file if the data source requires one.

   The following example loads two data sources, one of which uses a rules file, into the same data load buffer:

   ```
   import database ASOsamp.Basic data
      from server data_file 'file_1.txt'
      to load_buffer with buffer_id 1
      on error abort;
   import database ASOsamp.Basic data
      from server data_file 'file_2'
      using server rules_file ‘rule’
      to load_buffer with buffer_id 1;
      on error abort;
   ```

   To load data into multiple load buffers simultaneously, see Multiple Data Loads in Parallel.

3. Use the import data MaxL statement to commit the data load buffer contents to the cube. For example:

   ```
   import database ASOsamp.Basic data
      from load_buffer with buffer_id 1;
   ```

   To commit the contents of multiple data load buffers into the cube with one MaxL statement, see Multiple Data Loads in Parallel.

The following incremental data load example provides optimal performance when new data values do not intersect with existing values:

1. Create a single data load buffer using the ignore_missing_values and ignore_zero_values properties. For example:

   ```
   alter database ASOsamp.Basic
      initialize load_buffer with buffer_id 1
      property ignore_missing_values, ignore_zero_values;
   ```

If the cube must be available for send data requests while being updated, initialize the data load buffer with the **resource_usage** grammar set for 80%. For example:

```
alter database ASOsamp.Basic
   initialize load_buffer with buffer_id 1
   resource_usage 0.8 property
   ignore_missing_values, ignore_zero_values;
```

2. Load the data into the buffer. For example:

```
import database ASOsamp.Basic data
   from server data_file 'file_1.txt'
   to load_buffer with buffer_id 1
   on error abort;
import database ASOsamp.Basic data
   from server data_file 'file_2'
   to load_buffer with buffer_id 1;
   on error abort;
```

3. Commit the contents of the data load buffer to the cube by creating a slice and adding values. For example:

```
import database ASOsamp.Basic data
   from load_buffer with buffer_id 1
   add values create slice;
```

Data Load Buffer Resource and Disk Space Usage

When you use incremental data load to load data values to an Essbase aggregate storage (ASO) cube, you can put constraints on the allowed resource usage and wait time for the temporary data load buffer. You can reduce disk space usage by managing the tablespace.

Controlling Data Load Buffer Resource Usage

When performing an incremental data load, Essbase uses the aggregate storage cache for sorting data. You can control the amount of the cache a data load buffer can use by specifying the percentage. The percentage is a number between .01 and 1.0 inclusive; only two digits after the decimal point are significant—for example, 0.029 is interpreted as 0.02. By default, the resource usage of a data load buffer is set to 1.0, and the total resource usage of all data load buffers created on a database cannot exceed 1.0. For example, if a buffer of size 0.9 exists, you cannot create another buffer of a size greater than 0.1.

> **Note:**
>
> Send operations internally create load buffers of size 0.2; therefore, a load buffer of the default size of 1.0 will cause send operations to fail because of insufficient data load buffer resources.

To set the amount of resources the buffer is allowed to use, specify the percentage when you initiate the data load in the Essbase web interface. If using MaxL, use the alter database MaxL statement with the **resource_usage** grammar.

For example, to set the resource_usage to 50% of the total cache, use this statement:

```
alter database ASOsamp.Basic
   initialize load_buffer with buffer_id 1
   resource_usage .5;
```

If you plan to run concurrent send operations, use the ASOLOADBUFFERWAIT configuration setting and the **alter database** MaxL statement with the **wait_for_resources** grammar. ASOLOADBUFFERWAIT applies to the creation of aggregate storage data load buffers with the **wait_for_resources** option, and applies to allocations, custom calculations, data update operations.

Managing Disk Space For Incremental Data Loads

Incremental data loads on aggregate storage cubes may use disk space up to two times the size of your current data files. For example, assume that the size of a cube's data is 1 GB, and the size of the incremental data load is 200 MB, for a total size of 1.2 GB. During the incremental data load process, Essbase might use up to 2.4 GB of disk space.

In cases where databases are larger than 2 GB, you can reduce disk space utilization by setting the maximum file size of the default tablespace to no more than 2 GB.

To set the maximum file size of the default tablespace, you can use the **alter tablespace** MaxL statement.

Related Links

alter database

ASOLOADBUFFERWAIT

alter tablespace

Data Load Buffer Properties

When incrementally load data values to an aggregate storage (ASO) load buffer, you can tell Essbase to ignore missing and zero values in the source data, and you can resolve cell conflicts (eliminate invalid aggregations by combining duplicate cells).

The data load buffer properties you can set are:

- ignore_missing_values: Ignores #MI values in the incoming data stream
- ignore_zero_values: Ignores zeros in the incoming data stream
- aggregate_use_last: Combines duplicate cells by using the value of the cell that was loaded last into the load buffer

> **Note:**
>
> When loading text and date values into an aggregate storage database, use the **aggregate_use_last** property to help eliminate invalid aggregations. For other guidelines, see Loading, Clearing, and Exporting Text and Date Measures.

If you use multiple properties in the command and any conflict, the last property listed takes precedence.

Handling Missing and Zeroes in the Data Stream

When loading data incrementally, you can specify how missing and zero values in the source data are treated when loading the data into the data load buffer.

To set data load buffer properties, use the **alter database** MaxL statement with the **property** grammar.

For example:

```
alter database ASOsamp.Basic
   initialize load_buffer with buffer_id 1
   property ignore_missing_values, ignore_zero_values;
```

Resolving Cell Conflicts

For resolving cell conflicts for duplicate cells, you can specify whether to use the last cell loaded into the load buffer.

By default, when cells with identical keys are loaded into the same data load buffer, Essbase resolves the cell conflict by adding the values together.

To create a data load buffer that combines duplicate cells by accepting the value of the cell that was loaded last into the load buffer, use the **alter database** MaxL statement with the **aggregate_use_last** grammar.

For example:

```
alter database ASOsamp.Basic
   initialize load_buffer with buffer_id 1
   property aggregate_use_last;
```

> **Note:**
>
> When using data load buffers with the **aggregate_use_last** grammar, data loads are significantly slower, even if there are not any duplicate keys.

Multiple Data Loads in Parallel

Multiple data load buffers can exist on an Essbase aggregate storage (ASO) cube. Although only one commit operation can be active at a time, you can commit multiple data load buffers in the same commit, which is faster than committing buffers individually.

To load data into multiple data load buffers simultaneously, use separate MaxL Shell sessions. For example, in one MaxL Shell session, load data into a buffer with an ID of 1:

```
alter database ASOsamp.Basic
   initialize load_buffer with buffer_id 1 resource_usage 0.5;
import database ASOsamp.Basic data
   from data_file "dataload1.txt"
   to load_buffer with buffer_id 1
   on error abort;
```

Simultaneously, in another MaxL Shell session, load data into a buffer with an ID of 2:

```
alter database ASOsamp.Basic
   initialize load_buffer with buffer_id 2 resource_usage 0.5;
import database ASOsamp.Basic data
   from data_file "dataload2.txt"
   to load_buffer with buffer_id 2
   on error abort;
```

When the data is fully loaded into the data load buffers, use one MaxL statement to commit the contents of both buffers into the database by using a comma-separated list of buffer IDs:

For example, this statement loads the contents of buffers 1 and 2:

```
import database ASOsamp.Basic data
   from load_buffer with buffer_id 1, 2;
```

> **Note:**
>
> When loading SQL data into aggregate storage cubes, you can use up to eight rules files to load data in parallel. This functionality is different than the process described above. When preforming multiple SQL data loads in parallel, you can use one **import database** MaxL statement with the **using multiple rules_file** grammar. Essbase initializes multiple temporary aggregate storage data load buffers (one for each rules file) and commits the contents of all buffers into the cube in one operation.

List Data Load Buffers for an Aggregate Storage Cube

Multiple data load buffers can exist on an Essbase aggregate storage (ASO) cube. For a list and description of the data load buffers that exist on the cube, use the **query database** MaxL statement with the **list load_buffers** grammar.

The syntax of the MaxL statement to list ASO data load buffers is:

```
query database appname.dbname list load_buffers;
```

This statement returns the following information about each existing data load buffer:

Table 38-2 Data Load Buffer Information

Field	Description
buffer_id	ID of a data load buffer (a number between 1 and 4,294,967,296).
internal	A Boolean that specifies whether the data load buffer was created internally by Essbase (TRUE) or by a user (FALSE).
active	A Boolean that specifies whether the data load buffer is currently in use by a data load operation.

Table 38-2 (Cont.) Data Load Buffer Information

Field	Description
resource_usage	The percentage (a number between .01 and 1.0 inclusive) of the aggregate storage cache that the data load buffer is allowed to use.
aggregation method	One of the methods used to combine multiple values for the same cell within the buffer: • AGGREGATE_SUM: Add values when the buffer contains multiple values for the same cell. • AGGREGATE_USE_LAST: Combine duplicate cells by using the value of the cell that was loaded last into the load buffer.
ignore_missings	A Boolean that specifies whether to ignore #MI values in the incoming data stream.
ignore_zeros	A Boolean that specifies whether to ignore zeros in the incoming data stream.

See: Query Database (Aggregate Storage)

Create a Data Slice

You can incrementally commit the data load buffer to an Essbase aggregate storage (ASO) cube to create a slice. After loading the new slice into the cube, Essbase creates all necessary views on the slice (such as aggregate views) before the new data is visible to queries.

Creating a data slice is useful because it improves the performance of incremental data loads. The amount of time an incremental data load takes is proportional to the amount of new data; the size of the cube is not a factor.

To create a data slice, use the **import database** MaxL statement with the **create slice** grammar.

For example, to create a slice by overriding values (the default), use this statement:

```
import database ASOsamp.Basic data
   from load_buffer with buffer_id 1
   override values create slice;
```

> **Note:**
>
> If you use override values when creating a slice, #MISSING values are replaced with zeros. Using this option is significantly slower than using the add values or subtract values options.

See: Import Data (Aggregate Storage)

Merge Incremental Data Slices

When loading data incrementally to an aggregate storage (ASO) cube, you can manually merge the incremental data slices to the main cube slice, or you can use AUTOMERGE to configure Essbase to automatically merge slices during the data load.

Automatically Merging Incremental Data Slices During a Data Load to an Aggregate Storage Cube

Using the AUTOMERGE and AUTOMERGEMAXSLICENUMBER configuration settings, you can specify whether Essbase automatically merges incremental data slices during a data load to an aggregate storage cube.

AUTOMERGE configuration setting options:

- ALWAYS—Specifies to automatically merge incremental data slices during a data load to an aggregate storage cube. By default, merges are executed once for every four consecutive incremental data slices. If, however, the AUTOMERGEMAXSLICENUMBER configuration setting is used, the auto-merge process is activated when the AUTOMERGEMAXSLICENUMBER value is exceeded. The size of the incremental data slices is not a factor in selecting which ones are merged.

  The default value is ALWAYS.
- NEVER—Specifies to never automatically merge incremental data slices during a data load to an aggregate storage cube. To manually merge incremental data slices, use the **alter database** MaxL statement with the **merge** grammar.
- SELECTIVE—Specifies to activate the incremental data slice auto-merge process when the number of incremental data slices specified in the AUTOMERGEMAXSLICENUMBER configuration setting is exceeded. If the number of incremental data slices in the data load does not exceed the value of AUTOMERGEMAXSLICENUMBER, the auto-merge process is not activated.

Manually Merging Incremental Data Slices

You can merge all incremental data slices into the main slice or merge all incremental data slices into a single data slice while leaving the main slice unchanged. To merge slices, you must have the same privileges as for loading data (Database Update Permission or higher).

After the new input view is written to the cube, Essbase creates the aggregate views for the slice. The views created for the new slice are a subset of the views that exist on the main slice.

> **Note:**
>
> You cannot export data when performing a merge.

If you cleared data from a region using the logical clear region operation, which results in a value of zero for the cells you cleared, you can elect to remove zero value cells during the merge operation.

To perform merging operations, use the alter database MaxL statement with the **merge** grammar.

For example, to merge all incremental data slices into the main slice, use this statement:

```
alter database ASOsamp.Basic
   merge all data;
```

To merge all incremental data slices into the main slice and remove zero-value cells, use this statement:

```
alter database ASOsamp.Basic
   merge all data remove_zero_cells;
```

To merge all incremental data slices into a single data slice, use this statement:

```
alter database ASOsamp.Basic
   merge incremental data;
```

> **Note:**
>
> Before you copy an aggregate storage application, you must merge all incremental data slices into the main slice. Data in unmerged incremental data slices is not copied.

Related Links

AUTOMERGE

AUTOMERGEMAXSLICENUMBER

Alter Database (Aggregate Storage)

Replace Data Using Incremental Data Slice Contents

For aggregate storage (ASO) data sets that are small enough to reload completely while maintaining low data latency, Essbase can remove the current contents of an aggregate storage cube and replace the cube with the contents of a specified data load buffer.

The atomic replacement functionality transitions querying the old contents of the cube to the new contents without interrupting service. The newly loaded data set is aggregated to create the same set of views that existed for the replaced data set.

Essbase also allows for atomically replacing the contents of all incremental data slices in a cube. Consider a situation in which data can be separated into a relatively large, static data set that is never updated and a relatively small, volatile data set for which the individual updates are difficult to identify but are confined to the volatile data set. For example, the large, static data set consists of historical transaction data for the last three years; however, for the transaction data for the past two months, users can change a characteristic of a transaction in the source database. Tracking these changes can be prohibitively complex. You can load the static data set as the main slice in a cube and the volatile data set as one or more incremental slices.

When committing slices during ASO incremental data load, Essbase removes the current contents of all incremental data slices and creates a new slice (using the **add values** grammar in the buffer commit specification of the **import database** MaxL statement) with the contents of a specified data load buffer. The newly loaded data set is augmented with aggregated views based on the set of views that exist on the main slice.

> **Note:**
>
> To use the **override** grammar, create a data load buffer with the ignore_missing_values property for optimal performance. Additionally, you must ensure that there are not any conflicts between the static and volatile data sets (for example, there should not be a value in each data set for the same cell).

To replace the contents of a database or the incremental data slices in a cube, use the **import database** MaxL statement with the **override** grammar.

For example, to replace the contents of a cube, use this statement:

```
import database ASOsamp.Basic data
   from load_buffer with buffer_id 1
   override all data;
```

To replace the contents of all incremental data slices with a new slice, use this statement:

```
import database ASOsamp.Basic data
   from load_buffer with buffer_id 1
   override incremental data;
```

> **Note:**
>
> If the override replacement fails, Essbase continues to serve the old data set.

In Smart View, the submit command is equivalent to using the incremental data load functionality with the **override** grammar.

While performing a send operation, new requests for lock, unlock, and retrieve and lock will wait until the send operation is completed.

See: Import Data (Aggregate Storage)

View Incremental Data Slices Statistics

Essbase provides statistics on the size and number of incremental aggregate storage (ASO) data slices, and the cost of querying the incremental data slices.

The time it takes for a query to access all of the incremental data slices is expressed as a percentage (between .01 and 1.0 inclusive). If a cube has a main slice and multiple incremental data slices, a query statistic of 0.66 means that two-thirds of the query time was spent querying the incremental data slices and one-third was spent querying the main data slice. If the cost of querying the incremental data slices is too high, you can merge the slices.

To view the information about slices, use the **list aggregate_storage slice_info** grammar in the **query database** MaxL statement. For example,

```
query database ASOsamp.Basic list aggregate_storage slice_info;
```

Query Database (Aggregate Storage)

Renegade Members in Aggregate Storage Data Loads

Renegade members enable continuation of an Essbase aggregate storage (ASO) data load even if a specified member combination has missing or invalid members.

When a data load encounters a missing or invalid member, the data load continues, with the data value of the missing or invalid member stored under the member that is tagged as the renegade member in the dimension. If a renegade member is not set in the dimension, the record is rejected. If data already exists for the renegade member, the behavior depends on whether you selected to add values or to overwrite values when creating the data load rules file.

Each dimension can have only one member assigned as a renegade member, and the renegade member must be a level-0 member.

The following data load file includes a member named SC:

```
Product  Measures   *Data*
NY,      Sales       100
SA,      Sales       200
SC,      Sales       300
```

In the following outline, no member is named SC; however, the member named SA is set as the renegade member in the Products dimension:

```
Products (+)
   NY (+)
   SA (+)
Measures (+)
   Sales (+)
   COGS (+)
```

During the data load, the data value for the member combination SC and Sales, which is 300, is loaded into renegade member SA and Sales.

In the following data load file, two records exist for SC and Sales, each with different values:

```
Product    Measures    *Data*
NY,        Sales        100
SA,        Sales        200
SC,        Sales        250
SC,        Sales        300
```

Both values for SC and Sales (250 and 300) are loaded into SA and Sales. If you selected to add values, the value in the cell is 550 (250 + 300). If you selected to overwrite values, then the value in the cell is the last one loaded; in this case, 300.

The following examples illustrate the behavior of renegade members using the following data load file:

```
Months     Transaction Type     Customer     Product     Price
Jan,       Sale,                Discard1,    Product1    300
Jan,       Sale,                Discard1,    Discard2    300
Jan,       Sale,                Customer1,   Discard2    300
```

Discard1 and Discard2 do not exist in the outline.

- **Example 1:**

  If the Customer dimension has the **Customer1** member tagged as renegade, and the other dimensions do not have renegade members, only the first record is loaded into the following intersection:

  ```
  Jan     Sale     Customer1(Ren)     Product1     300
  ```

  The other two records are rejected because the Product dimension does not have a renegade member. The rejected records are logged in the renegade member log file.

- **Example 2:**

  If the Product dimension has the **Product1** member tagged as renegade, and the other dimensions do not have renegade members, only the last record is loaded into the following intersection:

  ```
  Jan     Sale     Customer1     Product1(Ren)     300
  ```

  The other two records are rejected, because the Customer dimension does not have a renegade member. The rejected records are logged in the renegade member log file.

- **Example 3:**

  If the Customer and Product dimensions both have renegade members (Customer1 and Product1), all records are loaded into the following intersection:

  ```
  Jan     Sale     Customer1(Ren)     Product1(Ren)     900  (or 300 if overwrite
  is enabled)
  ```

Example 4:

In example 4, the Customer dimension has **RenMember1** tagged as renegade, and the Product dimension has **RenMember2** tagged as renegade. Using the following data load file, all records are loaded because both the Customer and Product dimensions have renegade members.

Customer1 and **Product1** are not renegade members. "Discard1" and "Discard2"do not exist in the outline.

Data load file:

```
Months   Transaction Type  Customer     Product    *Data*
Jan,      Sale,            Discard1,    Product1    300
Jan,      Sale,            Discard1,    Discard2    300
Jan,      Sale,            Customer1,   Discard2    300
```

The values specified in the data load file for discard members are instead automatically loaded into the designated renegade members:

Loaded data:

```
Months  Measures   Customer          Product           Price
Jan     Sale       RenMember1(ren)   ProductR          300
Jan     Sale       RenMember1(ren)   RenMember2(ren)   300
Jan     Sale       CustomerR         RenMember2(ren)   300
```

Logging for renegade members is not enabled by default. To enable logging, use the RENEGADELOG configuration setting, which, when set to TRUE, enables logging of members loaded into a renegade member intersection.

> **Note:**
>
> Renegade members can be referenced in calculation and report scripts. Renegade members are not supported in tabular data loads or spreadsheet update operations.

Source Data Differences for Aggregate Storage Data Loads

While processing records in the source data in preparation for loading values into aggregate storage (ASO) cubes, Essbase processes records only for the level 0 dimension intersections where the member does not have a formula.

The following example shows a source of data that has records for only level 0 intersections. The last field contains data values, and the other fields are level 0 members of their respective dimensions.

```
Jan, Curr Year, Digital Cameras, CO, Original Price, 10784
Jan, Prev Year, Camcorders, CO, Original Price, 13573
```

Essbase ignores records that specify upper-level members and, at the end of the data load, displays the number of skipped records.

For example, the following record would be skipped because member **Mid West** is a level 1 member:

```
Jan, Curr Year, Digital Cameras, Mid West, Original Price, 121301
```

Sorting through data is unnecessary, because Essbase reads and sorts records internally before committing values to the cube.

Clear Data from Aggregate Storage Cubes

You can clear all data from an Essbase aggregate storage (ASO) cube, or you can clear data from a specified region, retaining the data located in other regions. You can do physical or logical clears.

To clear data, you must have the same privileges as for loading data (Database Update Permission or higher).

Clearing Data from Specific Regions

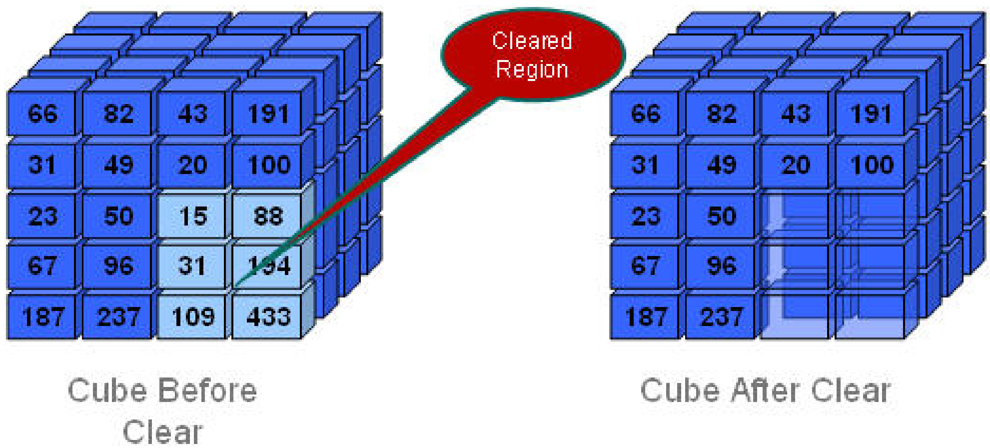
This method is useful when you want to delete volatile data (such as data corresponding to the last month), while retaining historical data.

Methods for clearing data from a region:

- **Physical**

  The input cells in the specified region are physically removed from the aggregate storage cube, as illustrated below.

  Figure 38-2   Physically Clearing a Region of Data

  If there are multiple data slices in the cube, the physical clear region operation automatically merges all data slices into the main data slice. After data for the specified region is cleared, Essbase materializes all aggregate views that were present in the main data slice before the clear region operation took place.

  The process for physically clearing data completes in a length of time proportional to the size of the input data, not to the size of the data being cleared. Therefore, you might use this method only when removing large slices of data.

  To physically clear data, use the alter database MaxL statement with the **clear data in region** grammar and the **physical** keyword:

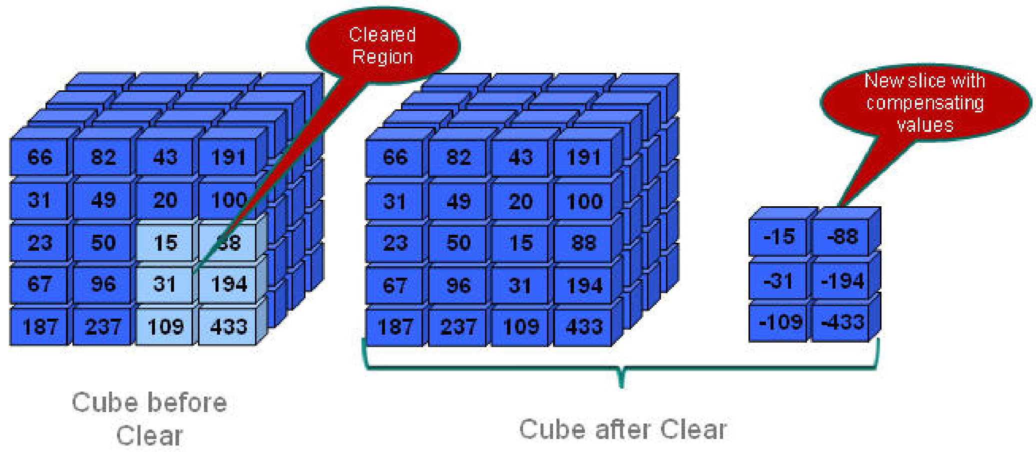
  ```
  alter database appname.dbname clear data in region 'MDX set expression'
  physical;
  ```

  To save time, you can use a comma-separated list of *MDX set expressions* to clear from multiple physical regions.

- **Logical**

  The input cells in the specified region are written to a new data slice with negative, compensating values that result in a value of zero for the cells you want to clear, as illustrated below.

Figure 38-3 Logically Clearing a Region of Data



The logical clear region operation automatically merges only the data slice with zero values into the main data slice; other data slices in the cube are not merged. After data for the specified region is cleared, Essbase materializes aggregate views only in the new data slice.

The process for logically clearing data completes in a length of time that is proportional to the size of the data being cleared. Because compensating cells are created, this option increases the size of the cube.

To logically clear data, use the alter database MaxL statement with the **clear data in region** grammar but without the **physical** keyword:

```
alter database appname.dbname clear data in region 'MDX set expression';
```

Queries to the logically cleared region return zero values instead of #MISSING values. You may need to update formulas that rely on #MISSING values for empty cells.

To remove cells with a value of zero, use the alter database MaxL statement with the **merge** grammar and the **remove_zero_cells** keyword.

> **Note:**
>
> Oracle does not recommend performing a second logical clear region operation on the same region, because the second operation does not clear the compensating cells created in the first operation and does not create new compensating cells.

In specifying the region to be cleared, follow these guidelines:

- The region must be symmetrical.
  - {(Jan, Budget)} is a valid symmetrical region that clears all Budget data for Jan.
  - {(Jan, Forecast1),(Feb, Forecast2)} is an invalid region because it consists of two asymmetrical regions (Jan, Forecast1 and Feb, Forecast2).
- Individual members in any dimension in the region specification must be stored members.
- Members in the region cannot be:

  – Dynamic members (members with implicit or explicit MDX formulas)
  – From attribute dimensions

    If you need to clear cells by an attribute, use the Attribute MDX function.

- Members in the region can be upper-level members in stored hierarchies, which is a convenient way to specify multiple level 0 members.

  For example, you can specify Qrt1, which is the same as specifying Jan, Feb, and Mar (the level 0 children of Qrt1):

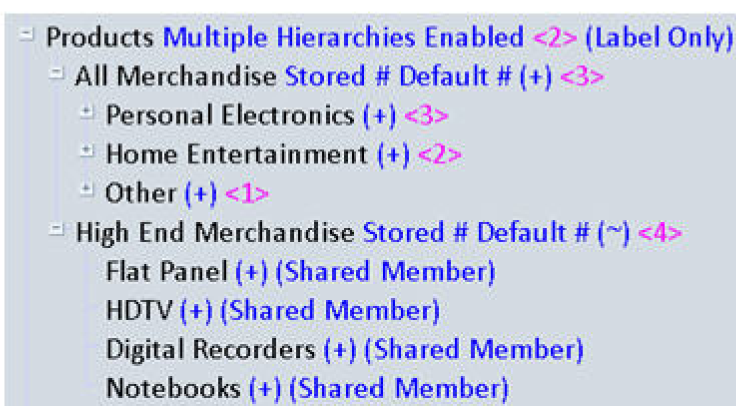
  The following two MaxL statements produce the same results:

```
alter database appname.dbname clear data in region '{Qtr1}';
```

```
alter database appname.dbname clear data in region '{Jan, Feb, Mar}';
```

- (Physically clearing data only) Members in the region can be upper-level members in alternate hierarchies.

  For example, you can specify High End Merchandise, which is the same as specifying Flat Panel, HDTV, Digital Recorders, and Notebooks (the shared, level 0 children of High End Merchandise):



The following two MaxL statements produce the same results:

```
alter database appname.dbname clear data in region '{High End
Merchandise}';
```

```
alter database appname.dbname clear data in region '{[Flat Panel],[HDTV],
[Digital Recorders],[Notebooks]}';
```

To specify members in alternate hierarchies when logically clearing data, use the Descendants MDX function.

> **Note:**
>
> When the region contains upper-level members from alternate hierarchies, you may experience a decrease in performance. In this case, consider using only level 0 members.

- The MDX set expression must be enclosed with single quotation marks.

For example, to clear all January data for Forecast1 and Forecast2 scenarios, use this statement:

```
alter database ASOSamp.Basic clear data in region 'CrossJoin({Jan},
{Forecast1, Forecast2})';
```

During the clear region operation, you cannot perform operations that update the cube (such as loading data, merging data slices, or clearing data from another region), nor export data. You can query the cube; however, the query results are based on the data set before the clear region operation.

The **clear data in region** grammar cannot clear data from the entire cube.

Clearing All Data

Clearing all data from an aggregate storage cube is the same as for a block storage cube. To clear the entire cube, use the **alter database** MaxL statement with the **reset** grammar:

```
alter database appname.dbname reset;
```

Related Topics

Clear Aggregated Data from the Cube

alter database

Block Storage Calculation Features That Do Not Apply to Aggregate Storage

Calculation of Essbase aggregate storage (ASO) cubes differs from that of block storage cubes. The following characteristics of calculating block storage databases do not apply to ASO.

- Block storage calculation scripts written with Essbase calculation commands and functions. However, you can write aggregation scripts and custom ASO calculations.
- Dynamic Calc member storage property
- Block storage formula syntax and predefined Essbase functions in formulas
- Custom-defined calculation functions and macros developed in Java
- Formulas on members of dimensions that are not aggregate storage dynamic hierarchies
- Preloaded values for member intersections above level 0
- Two-pass calculation tags
- Block-storage Intelligent Calculation

Related Links

Calculation Differences for ASO

Aggregation of Data in an ASO Cube

Custom Calculations and Allocations on Aggregate Storage Cubes

Aggregation of Data in an ASO Cube

To prepare an Essbase aggregate storage (ASO) cube for retrieval, you load the level 0 values, and calculate the cube by aggregating. The remaining values are calculated when data is retrieved. Values calculated for retrievals are not stored.

Though ASO cubes require no calculation after data values are loaded into level 0 cells, you can precalculate data values as aggregate views, to optimize aggregation of data.

Essbase calculates ASO values through outline consolidation and MDX formulas on stored and dynamic hierarchies. When a data load is complete, the cube is ready to calculate. When retrieval requests are made, Essbase consolidates the values loaded for level 0 members, and calculates formulas.

To improve retrieval performance, Essbase can aggregate values and store them ahead of time. However, aggregating and storing all values can be a lengthy process that requires disk space for storage. Essbase provides an intelligent aggregation process that balances time and storage resources.

As Essbase cubes grow, retrievals must process more data values to satisfy the calculation needs of the queries. For faster retrieval, Essbase enables you to precalculate data values and store those values in aggregations. If cube size nears one million aggregate cells, you should strongly consider performing an aggregation. Depending on the usage environment, you can achieve performance benefits by precalculating smaller cubes as well. Use MaxL, or the Build Aggregations job in the Essbase web interface.

Aggregation-Related Essbase Terms

To understand how Essbase aggregate storage (ASO) cubes are calculated, learn the terminology about aggregation, including: aggregate cells, input cells, aggregate views, consolidation, aggregations, and aggregation scripts.

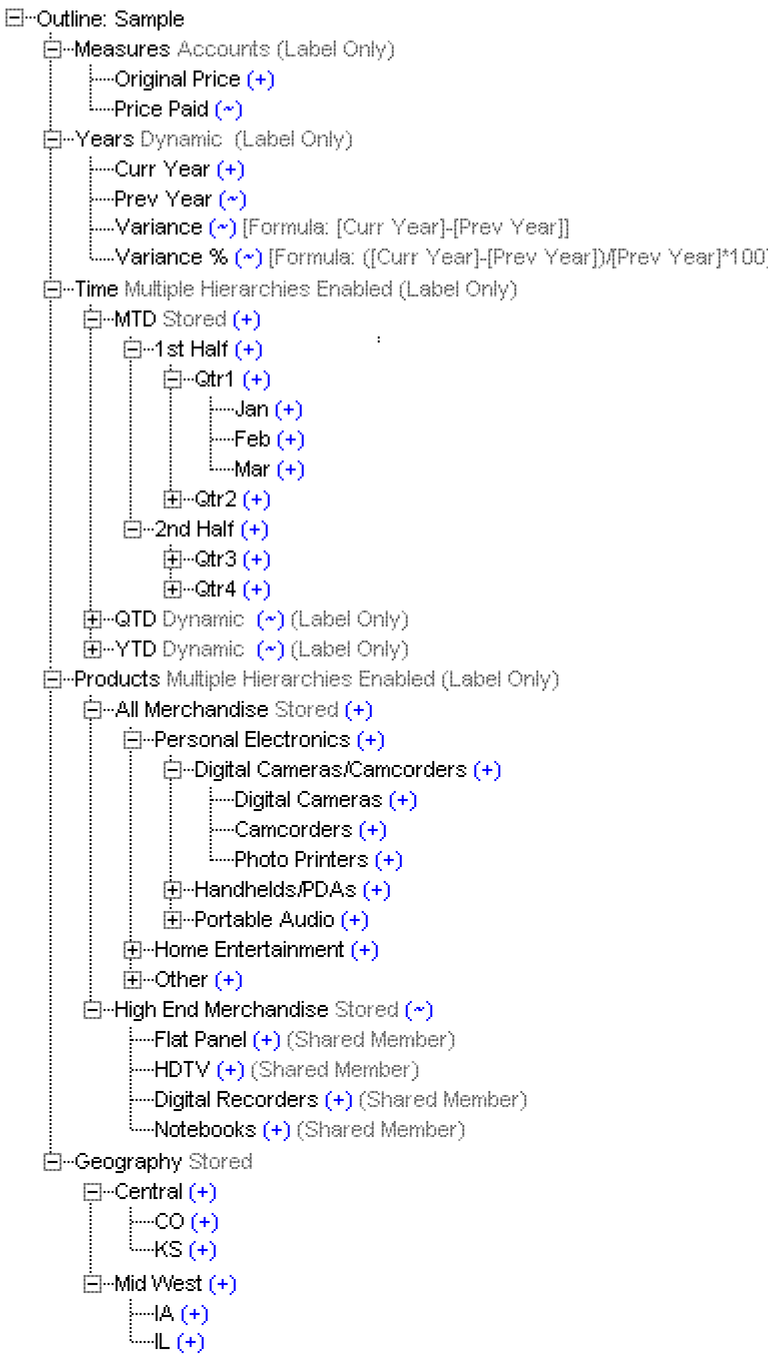
Aggregate Cells

Cells for level 0 intersections across dimensions, without formulas, are called input cells, meaning that data values can be loaded to them. Higher-level cells involving members of the accounts dimension or dynamic hierarchies are always calculated at retrieval time.

All other higher-level intersections across dimensions are *aggregate cells*. Values for aggregate cells must be rolled up (consolidated) from lower-level values.

For example, in the ASOSamp.Basic outline, Price Paid > Curr Year > 1st Half > Portable Audio > CO is an aggregate cell; Original Price > Curr Year > Jan > Camcorders > CO is another aggregate cell.

Figure 38-4   Sample Aggregate Storage Outline



Aggregate cell values are calculated for each request, or they can be precalculated and stored on disk.

Aggregate Views

When Essbase defines which aggregate cells to precalculate and store, it does so by creating aggregate views. An *aggregate view* is a collection of aggregate cells. The collection is based on the levels of the members within each dimension.

For example, consider one aggregate view for the outline. This aggregate view includes aggregate cells for the following dimension levels:

- Measures dimension, level 0
- Years dimension, level 0
- Time dimension, level 1 of hierarchy 0
- Product dimension, level 2 of hierarchy 0
- Geography dimensions, level 0

The example aggregate view is shown as 0, 0, 1/0, 2/0, 0.

Each dimension is shown, left to right, in its sequence in the outline. If a dimension contains hierarchies, the notation specifies the member level within its hierarchy. Hierarchies within a dimension are numbered top-down, starting with hierarchy 0.

The 0, 0, 1/0, 2/0, 0 aggregate view contains aggregate cells that include the following member intersections:

```
Original Price, Curr Year, Qtr1, Personal Electronics, CO
Original Price, Curr Year, Qtr1, Personal Electronics, KS
Original Price, Curr Year, Qtr1, Home Entertainment,   CO
Original Price, Curr Year, Qtr1, Home Entertainment,   KS
Original Price, Curr Year, Qtr2, Personal Electronics, CO
Original Price, Curr Year, Qtr2, Personal Electronics, KS
Original Price, Curr Year, Qtr2, Home Entertainment,   CO
Original Price, Curr Year, Qtr2, Home Entertainment,   KS
Original Price, Curr Year, Qtr3, Personal Electronics, CO
Original Price, Curr Year, Qtr3, Personal Electronics, KS
Original Price, Curr Year, Qtr3, Home Entertainment,   CO
Original Price, Curr Year, Qtr3, Home Entertainment,   KS
Original Price, Curr Year, Qtr4, Personal Electronics, CO
Original Price, Curr Year, Qtr4, Personal Electronics, KS
Original Price, Curr Year, Qtr4, Home Entertainment,   CO
Original Price, Curr Year, Qtr4, Home Entertainment,   KS
Original Price, Prev Year, Qtr1, Personal Electronics, CO
Original Price, Prev Year, Qtr1, Personal Electronics, KS
Original Price, Prev Year, Qtr1, Home Entertainment,   CO
Original Price, Prev Year, Qtr1, Home Entertainment,   KS
and so on...
```

Aggregations

Aggregations are consolidations, based on outline hierarchy, of level 0 data values. An aggregation contains one or more aggregate views that are rolled up (consolidated). Essbase provides an intelligent aggregation process that selects aggregate views to be rolled up, aggregates them, and then stores the values for the cells in the selected views. If an aggregation includes aggregate cells dependent on level 0 values that are changed through a data load, the higher-level values are automatically updated at the end of the data load process.

The term *aggregation* is used for the aggregation process and the set of values stored as a result of the process.

How ASO Data Values are Rolled Up

The hierarchical structure of an aggregate storage outline determines how values are rolled up. Level 0 member values roll up to level 1 member values, level 1 member values roll up to level 2 member values, and so on.

Consolidation operators assigned to members of dynamic hierarchies define the operations used in the roll-up: add (+), subtract (-), multiply (*), divide (/), percent (%), no operation (~), and (^) never consolidate.

Members of stored hierarchies can have only the addition (+) or the no-consolidation (~) operator.

For more complex operations, you can provide MDX formulas on members of dynamic hierarchies. MDX formulas are written in the same format as MDX numeric value expressions.

Aggregation Scripts

Each *aggregation script* is a file that defines a particular selection of aggregate views to be materialized. See Aggregation Scripts for Essbase ASO Cubes.

Perform ASO Aggregations

An aggregation involves view selection and materialization. For aggregate view selections that prove useful, you can save them as aggregation scripts, bypassing view selection next time they are needed. When you perform an aggregation, the selected views are calculated.

You can optionally store precalculated values as aggregations on an Essbase aggregate storage cube. Use MaxL or Jobs to perform aggregations.

The aggregation process has two phases:

- Aggregate view selection.
- Calculation and storage of values for the selected aggregate views. This phase is also called the materialization of the aggregation.

During the aggregate view selection phase, Essbase analyzes how calculating and storing various combinations of aggregate views might affect average query response time. As input to the analysis, you can define physical storage and performance requirements. You can also track data usage and provide the information to the analysis process, as described in Select Views Based on Usage.

Based on their usefulness and resource requirements, Essbase creates a list of aggregate views. Included with each view in the list is storage and performance information that applies when that aggregate view plus all other aggregate views listed above it are stored. You can choose to aggregate the listed views, select and aggregate a subset of the listed views, or rerun the selection process with different input criteria. You can also add to an aggregation the materialization of new views that are based on new selection criteria. Refer to Optimization for Aggregate View Selection.

Whether or not you materialize the selection, you can save the selection of aggregate views as an aggregation script. Aggregation scripts provide flexibility and can save time because they enable you to bypass the selection process if the same selection is needed again. See Aggregation Scripts for Essbase ASO Cubes.

After the selection process is finished, the selected aggregate views are calculated when you materialize the selected aggregate views into an aggregation.

The following process is recommended for defining and materializing aggregations:

1. After the outline is created or changed, load data values.

2. Perform the default aggregation.

   Optional: Specify a storage stopping point.

3. Materialize the suggested aggregate views and save the default selection in an aggregation script.
4. Run the types of queries for which the aggregation is designed.
5. If query time or aggregation time is too long, consider optimizing the aggregation.
6. (Optional) Save the aggregation selection as an aggregation script.

To perform an ASO cube aggregation selection or materialization, you can use the Essbase web interface or these MaxL statements:

- execute aggregate process
- execute aggregate selection
- execute aggregate build

You can also configure Essbase to Generate Aggregate Views Automatically when needed.

Optimization for Aggregate View Selection

Essbase's default view selection for aggregate storage (ASO) cubes provides excellent performance. However, accepting all aggregate views in the selection list does not guarantee optimum performance. You can optimize the view selection for your cube's environment and retrieval patterns.

For its default selection of aggregate views, Essbase analyzes stored hierarchies and assumes an equal chance that any aggregate cell will be retrieved. Essbase cannot account for external factors such as the amount of available memory at the time of a query. Available memory can be affected by such factors as the cache memory definition at retrieval time, or the memory other concurrent processes require.

If you want to track which data is most queried and include the results and alternate views in the aggregate view selection process, refer to Select Views Based on Usage.

To improve performance of aggregate storage (ASO) cubes in Essbase 21c, you can configure Essbase to automate the creation and maintenance of default aggregate views based on metadata analysis, and you can control the aggregate view size. For more information, refer to Generate Aggregate Views Automatically.

As you tune and test aggregations, consider the following points:

- Improving retrieval performance can increase disk storage costs and the time it takes to materialize the aggregation.
- Tracking queries may result in a set of proposed aggregate views that provide better performance for some queries than for others. Selecting proposed aggregate views can considerably improve performance time of some queries with others experiencing little improvement—but never worse—as long as query type and frequency are close to the type and frequency of queries performed during the tracking period.
- Optimizing aggregations may require an iterative, fine-tuning process.

To help Essbase estimate the size of aggregate views, you can adjust the ASOSAMPLESIZEPERCENT configuration setting, which changes the number of input cells from which Essbase samples. The sample size is specified as a percentage of input-level data. The default, and minimum, sample size is 1 million (1,000,000) cells.

Essbase provides information to help you select and store the right balance of aggregate views for your ASO cube. Weigh this information against what you know about your retrieval requirements and environment. Use the following information to help you select aggregate views for an aggregation:

- The maximum storage requirement

  You can specify a storage limit for selecting aggregate views in two ways:

  – When the aggregation selection is initiated, you specify a maximum storage stopping value. Aggregate views are selected until the specified storage limit is reached or there are no more views to select.

    When using the execute aggregate process MaxL statement with the **stopping when total_size exceeds** grammar, you can specify the maximum disk space of the resulting data files, as a ratio of the current cube size. For example, if the size of a cube is 1 GB, specifying the total size as 1.2 means that the size of the resulting data cannot exceed 20% of 1 GB, for a total of 1.2 GB.

  – After each analysis of the cube, Essbase displays information about the level 0 input cell view followed by a list of suggested aggregate views. Displayed by each aggregate view is a storage number that includes that aggregate view and all other aggregate views it depends on. You can consider this storage number as you select the aggregate views to be included in the aggregation.

- The relative “Query Cost” performance improvement

  The Query Cost number that is displayed by each aggregate view in the list projects an average retrieval time for retrieving values from the associated aggregate view. The default view selection estimates the cost as the average of all possible queries. When using query tracking, the estimated cost is the average for all tracked queries. The cost number for a specific aggregate view can be different in different selection lists; for example, aggregate view 0, 0, 1/0, 2/0, 0 can show a different query cost in the default selection list than it would show in a selection that includes tracked queries in the analysis.

  To compute the percentage improvement, divide the query cost value for the aggregate view into the query cost value shown for storing only level 0 input cells.

- Tracked usage

  Before running an aggregate view selection, you can turn on query tracking to determine which data is retrieved most often. After some period of cube activity, you can have Essbase include the usage statistics in the aggregation analysis process.

- Aggregation time

  The time it takes to perform an aggregation after the selection process completes increases for each aggregate view materialized. To determine actual aggregation time, you must perform the aggregation.

The following process is recommended for fine-tuning aggregations:

1. Perform the default aggregations described in Perform ASO Aggregations.
2. Save the default selection in an aggregation script. See Aggregation Scripts for Essbase ASO Cubes.
3. Turn on query tracking. See Select Views Based on Usage.
4. Have users perform their usual queries against the cube or perform the batch query operations for which the aggregation is being designed. Queries from all query tools are tracked.
5. After sufficient time to capture data retrieval requirements, perform another aggregation including tracked data.

6. Analyze the proposed list of aggregate views to be stored, and select the aggregate views that you determine provide the best balance of system resources and retrieval performance.
7. Materialize the selected aggregate views and, if desired, save the selection in an aggregation script.
8. Working with aggregation scripts and various selection criteria, repeat the process until you think you have the optimum selection of aggregate views for your situation.

> **Note:**
>
> To optimize aggregations for different retrieval situations, such as for generating reports or user queries, you may need to repeat the tuning process, creating an aggregation script for each situation.

Generate Aggregate Views Automatically

To improve performance of aggregate storage (ASO) cubes in Essbase 21c, you can configure METADATABASEDAGGVIEWSBUILD to let Essbase automate the creation and maintenance of default aggregate views, based on metadata analysis. You can control the aggregate view size using DEFAULTVIEWBUILDSIZE.

In releases prior to Essbase 19c, default aggregate views were created by Essbase using internal analysis based on data sampling. Starting in Essbase 19c, Essbase can use metadata analysis for selecting the default views.

If you use the view-selection algorithm based on metadata analysis, it enables Essbase to perform automatic default aggregate views selection, build, and maintenance. When you select to use automatically generated aggregation views, query performance can improve. It also impacts data load time and increases the amount of disk space used by data, because right after a data load, aggregate views will be built (or updated, in the case of existing views).

To automate default aggregate view generation and management, set the METADATABASEDAGGVIEWSBUILD configuration to AUTO (in Essbase 21c or later), or set the DEFAULTVIEWBUILD configuration to TRUE (in Essbase 19c). If enabled, aggregation views will be generated automatically based on qualifying criteria, or on-demand (when you run the execute aggregate selection MaxL statement).

For details about the qualifying criteria, refer to METADATABASEDAGGVIEWSBUILD (for Essbase 21c) or DEFAULTVIEWBUILD (for earlier releases).

To control the size of the resulting aggregate views, add the additional application configuration setting DEFAULTVIEWBUILDSIZE, and set its value to the desired total size ratio. For example, `DEFAULTVIEWBUILDSIZE AsoSamp 1.2` limits the resulting growth of the aggregated cube to no more than 20% of its size prior to the aggregation.

Select Views Based on Usage

Capture retrieval statistics for your Essbase aggregate storage (ASO) cube, and use these statistics to build aggregations tailored to retrieval patterns in your company. Essbase includes alternate hierarchies in its analysis of the cube when you enable query tracking to inform the aggregate view selection process.

Your organization's Essbase cube usage required for periodic report generation may be different than for ongoing user retrievals. To optimize for different retrieval situations, consider tracking situational usage patterns and creating aggregation scripts for each situation.

Before you begin the aggregation selection process, ensure that query tracking is on, and that it has been on long enough to capture representative usage. To enable it, use the MaxL statement alter database with the **enable query_tracking** grammar.

Query tracking holds query usage information in memory. Performing any of the following operations clears query usage information.

- Loading or clearing data
- Materializing or clearing an aggregation
- Turning off query tracking

Query tracking remains on until you turn it off, stop the application, or change the outline.

> **Note:**
>
> Query tracking and query tracing are different.
>
> Query *tracking* enables you to capture user retrieval statistics against an aggregate storage cube, so that Essbase can make view-based optimizations to improve the performance of aggregations. It is on by default. Related MaxL statements include:
>
> ```
> import query_tracking
> export query_tracking
> alter database enable query_tracking
> query database appname.dbname get cube_size_info
> ```
>
> Query *tracing* helps you monitor Essbase query performance metrics for block storage cubes (including hybrid mode). It is off by default. If you enable it, Essbase logs metrics in a trace report. Related configuration parameters: TRACE_REPORT, QUERYTRACE, QUERYTRACETHRESHOLD, LONGQUERYTIMETHRESHOLD.

View Selection Based on Aggregate Level Usage

If you manage an aggregate storage (ASO) cube, you can apply view selection properties to stored hierarchies to restrict Essbase from choosing certain levels for aggregation. In the Essbase web interface, you apply these properties using the outline editor, in the options group labeled **Aggregate level usage**.

By default, Essbase uses internal mechanisms to decide how to create aggregations. User-defined view selection provides a way for you to influence default view selection and view selection based on query data.

Database managers may apply view selection properties to stored hierarchies to restrict Essbase from choosing certain levels for aggregation.

- In the Classic Web Interface, you apply these properties using the outline editor, in the options group labeled **Aggregate level usage**.

- In the REST API, you can apply view selection properties using the Run Batch Outline Edit endpoint, using the `aggLevelUsage` property.

> **Note:**
>
> Secondary (alternate) hierarchies are either shared or attribute hierarchies.

Table 38-3 View Selection Properties

Property	Effect
Default	On primary hierarchies, Essbase considers all levels. It does not aggregate on secondary hierarchies unless alternative roll-ups are enabled.
Consider all levels	Considers all levels of the hierarchy as potential candidates for aggregation. This is the default for primary hierarchies, but not for secondary hierarchies.
Do not aggregate	Does not aggregate along this hierarchy. All views selected by Essbase are at the input level.
Consider bottom level only	Applies only to secondary hierarchies. Essbase considers only the bottom level of this hierarchy for aggregation.
Consider top level only	Applies only to primary hierarchies. Considers only top level of this hierarchy for aggregation.
Never aggregate to intermediate levels	Applies to primary hierarchies. Selects top and bottom levels only.

> **Note:**
>
> The bottom level of an attribute dimension consists of the zero-level attribute members. When a secondary hierarchy is formed using shared members, the bottom level comprises the immediate parents of the shared members.

Essbase considers only views that satisfy the selected view selection properties.

You should be familiar with the dominant query patterns of cubes before changing default properties; preventing selection of certain views will make queries to those views slower while improving the speed of other queries. Similarly, enabling Consider All Levels on a secondary hierarchy may speed queries to that hierarchy while making other queries slower.

To define view selection properties, you can use these MaxL statements:

- execute aggregate process
- execute aggregate selection

Aggregation Scripts for Essbase ASO Cubes

Aggregation scripts represent specific aggregate view selections on an Essbase aggregate storage (ASO) cube. They enable you the flexibility to skip the aggregate view selection process that might otherwise be required after a data load.

About Aggregation Scripts

Each aggregation script represents a specific aggregate view selection against the cube.

Aggregation scripts can save you time. For example, after loading new data values you need not perform another aggregate view selection. You can speed the aggregation process by using the selection stored in an aggregation script to materialize the aggregation.

Aggregation scripts also give you flexibility. You can use them to save aggregate view selections optimized for different retrieval situations; for example, you can use one script to optimize retrievals in month-end reporting and another for daily retrieval requirements.

Aggregation scripts for an ASO cube become invalid when the selection it contains is invalid for the cube. Create aggregation scripts when you create aggregations. Do not manually modify aggregation script files, which may cause unpredictable results.

Creating Aggregation Scripts

Saved aggregation scripts enable you to split up the total aggregation process. You can materialize an aggregation at a different time than when the aggregate views for the aggregation are selected. The aggregation script contains information derived during the aggregate view selection phase.

To create an aggregation script, you can use these MaxL statements:

- query database
- execute aggregate selection

Aggregation scripts are stored in the cube directory as text files with the `.csc` extension, and are valid as long as the dimension level structure in the outline has not changed.

To avoid the potential clutter of invalid aggregation script files, manually delete aggregation scripts when they are no longer useful.

Executing Aggregation Scripts

Executing an aggregation script materializes the aggregate views specified within it. Although you can create multiple aggregation scripts, only one aggregation can be materialized at a time.

To execute an aggregation script, you can use the execute aggregate build MaxL statement.

Clear Aggregated Data from the Cube

Clearing aggregations from your Essbase aggregate storage (ASO) cube removes non level-0 data. After aggregations are cleared, user queries calculate retrieved values dynamically from the level 0 values.

At times you might want to manually clear aggregations from the disk; for example, to make the disk space available for disk-intensive operations. Clearing aggregations clears all data, except level 0 values, from the cube, releasing the disk area for other use.

To clear aggregations, you can use the alter database MaxL statement.

Replace Aggregated Data in the Cube

You can replace an aggregation of your Essbase ASO cube data by clearing the existing aggregation and materializing a different selection of aggregate views. You can perform a new aggregate view selection and materialization process, or you can run an aggregation script.

Consider replacing the aggregation in the following situations:

- To optimize the selection of aggregate views to improve performance.
- To create aggregations optimized for different retrieval situations, such as for generating reports or user queries.
- To optimize an aggregation after significant growth in cube size. Gradually, as the size of a cube increases, an aggregation can become less efficient. Consider replacing an aggregation when performance degradation is noticeable or when the cube size increases to about 150% of its original size.
- To optimize aggregations for new or different operational environments, such as memory and disk resource availability changes.

You must replace an aggregation and associated aggregation scripts after the number of levels in a dimension has been changed or one or more dimensions have been added or removed from the outline.

Calculation Order and Solve Order in ASO Cubes

Calculation order for Essbase aggregate storage (ASO) cubes differs from that of block storage cubes. Stored hierarchies and attribute dimensions are consolidated, and then dynamic hierarchies are evaluated based on a defined solve order.

For aggregate storage cubes, Essbase calculates data in the following order:

1. **Aggregates members of stored hierarchies and attribute dimensions.** The order in which members and dimensions are aggregated is optimized internally and changes according to the nature of the cube outline and the existing aggregations. Because the aggregation is additive, the order in which Essbase aggregates the dimensions and members does not affect the results.

   Because the internal aggregation order for an aggregate storage cube is not predictable, any inherent rounding errors are also not predictable. These rounding errors are expected behavior in computer calculation and are extremely small in relation to the data values concerned.
2. **Calculates dynamic hierarchy dimension members and formulas.** The order in which members and formulas are evaluated is defined by the solve order property, which you can set for each member or dimension. Calculation order may affect calculation results.

Solve Order Property

The concept of solve order applies to query execution. When a cell is evaluated in a multidimensional query, the order in which the calculations should be resolved may be ambiguous. To remove ambiguity, you can use the solve order property to specify the required calculation priority.

> **Note:**
>
> It is good practice to specify the solve order by setting the solve order property at the member or dimension level. Members without formulas that do not have a specified solve order inherit the solve order of their dimension. Members with formulas that do not have a specified solve order have a solve order of zero.

To change the solve order, use the outline editor in the Essbase web interface, or use Smart View. To specify the solve order for a calculated member, you can use the **solve_order** parameter in the With Section of an MDX query.

The value of the solve order property determines the priority with which Essbase calculates the formulas. The formulas on the members that have a specified solve order are calculated in order from the lowest solve order to the highest. You can specify a solve order between 0 and 127. The default is 0.

You can specify the solve order at the member level or at the dimension level. Essbase uses the following information to define calculation precedence:

1. Member solve order
2. Dimension solve order (members without formulas for which you do not specify a member solve order inherit the solve order of their dimension. Members with formulas for which you do not specify a member solve order have a solve order of zero.)

   If multiple members have the same solve order, the members are evaluated in the reverse order in which their dimensions occur in the outline. The member that occurs later in the outline takes precedence.

   The tie situation calculation order is different for calculated members defined in an MDX query for block storage cubes.

   > **Note:**
   >
   > When a member formula is dependent on the value of another member, the member with the formula must have a higher solve order than the member or members on which it depends. For example, in the ASOSamp.Basic outline, Avg Units/Transaction depends on the value of Units and of Transactions. Avg Units/Transaction must have a higher solve order than Units and Transactions.

Example Using the Solve Order Property

The following example is based on the ASOSamp.Basic cube. To remove ambiguity in query results, the example uses the solve order property to specify the required calculation priority.

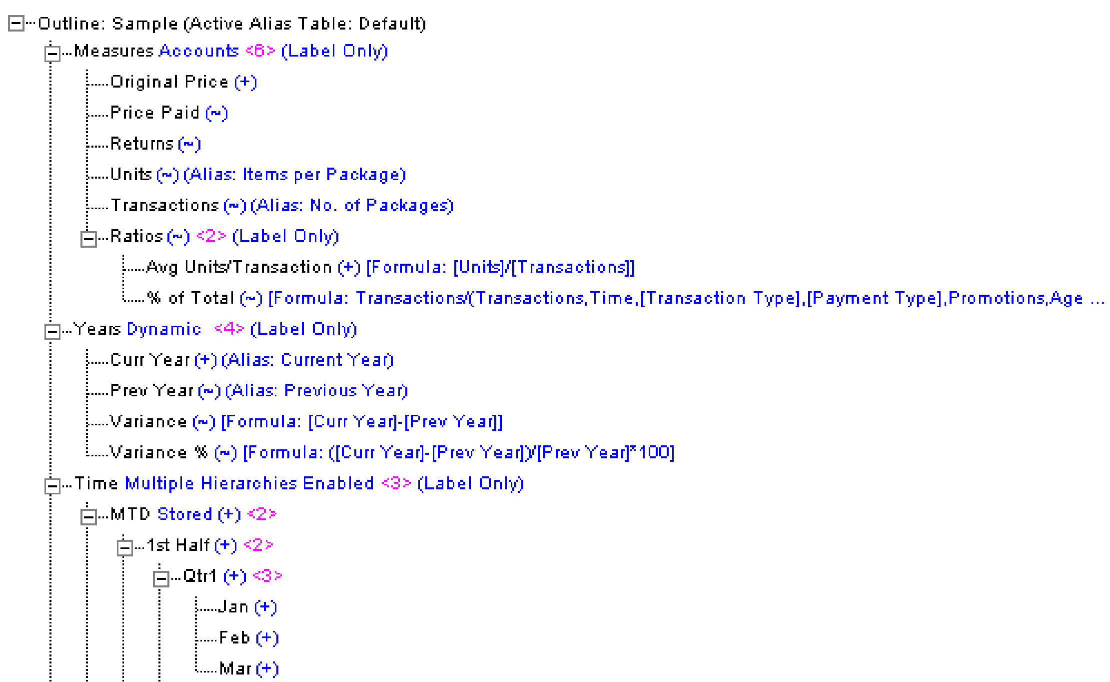
The spreadsheet query shown below retrieves data for the number of units sold and the number of transactions for January of the current year and for January of the previous year. The Variance member shows the difference between the current year and the previous year. The Avg Units/Transaction member shows a ratio of the number of units sold per transaction.

Figure 38-5 Results from Spreadsheet Query of ASOSamp.Basic cube Showing the Variance Between Two Ratios (C12)

	A	B	C
1			
2			
3			Jan
4	Curr Year	Units	42228
5		Transactions	44500
6		Avg Units/Transaction	0.94894382
7	Prev Year	Units	31643
8		Transactions	33160
9		Avg Units/Transaction	0.954252111
10	Variance	Units	10585
11		Transactions	11340
12		Avg Units/Transaction	-0.005308291
13	Variance %	Units	33.45131625
14		Transactions	34.19782871
15		Avg Units/Transaction	-0.556277601
16			

The following image shows the cube outline for these members, and the formulas applied to the Variance and Avg Units/Transaction members.

Figure 38-6 ASOSamp.Basic Cube Showing the Measures, Years, and Time Dimensions



When calculating the variance of the average units per transaction (cell C12 in the spreadsheet example), the result could be the variance between the two ratios, or the result could be the ratio of the two variances. The result depends on whether Essbase gives precedence to the formula on Variance or the formula on Avg Units/Transaction.

The value of the solve order property, which is attached to the members in the outline, determines the priority with which Essbase evaluates the formulas. The higher the solve order setting, the later in the order the member is calculated. For example, a formula with a solve order of 1 is solved before a member with a solve order of 2.

In the example, if the Variance member has a lower solve order than the Avg Units/Transaction member, then the formula on the Variance member takes precedence and the result is the variance between two ratios. This is the case in the ASOSamp.Basic cube, because the solve order of the Variance member is 10 and the solve order of the Avg Units/Transaction member is 20. The formula on Variance takes precedence, because the Variance member has the lower solve order. The result for cell C12 of the query in the spreadsheet example is the variance between the two ratios, as shown in the table below:

Table 38-4 Using the Solve Order Property to Specify the Variance Between Two Ratios

Member	Solve Order	Formula	Result of Intersection of Variance and Avg Units/ Transaction (cell C12)
Variance Avg Units/Transaction	10 20	Curr Year - Prev Year Units/Transactions	Current year average units/ transaction - previous year average units/transaction 0.94894382 (cell C6) - 0.954252111 (cell C9) = -0.005308291 (cell C12)

Alternatively, if you change the ASOSamp.Basic cube, and you give the Avg Units/Transaction member a lower solve order than the Variance member, then the formula on the Avg Units/ Transaction member takes precedence, and the result is the ratio of two variances, as shown in the table and spreadsheet example below:

Table 38-5 Using the Solve Order Property to Specify the Ratio of Two Variances

Member	Solve Order	Formula	Result of Intersection of Variance and Avg Units/ Transaction (cell C12)
Variance Avg Units/Transaction	20 10	Curr Year - Prev Year Units/Transactions	Variance (current year to previous year) of units / variance of transactions 10585 (cell C10) / 11340 (cell C11) = 0.933421517 (cell C12)

Figure 38-7 Results from Spreadsheet Query of ASOSamp.Basic Cube Showing the Ratio of Two Variances (C12)

	A	B	C
1			
2			
3			Jan
4	Curr Year	Units	42228
5		Transactions	44500
6		Avg Units/Transaction	0.94894382
7	Prev Year	Units	31643
8		Transactions	33160
9		Avg Units/Transaction	0.954252111
10	Variance	Units	10585
11		Transactions	11340
12		Avg Units/Transaction	0.933421517
13	Variance %	Units	33.45131625
14		Transactions	34.19782871
15		Avg Units/Transaction	0.978170764
16			

Time Balance and Flow Metrics in ASO Accounts Dimensions

Time Balance properties on Essbase aggregate storage (ASO) cubes provide built-in calculations along the Time dimension. Flow tags help you optimize calculation of time-related accounts. A flow metric is an unsigned Accounts dimension member that stores periodic and year-to-date values.

Use Time Balance Tags in ASO Accounts Dimensions

You can set Time Balance properties on aggregate storage Accounts dimensions to provide built-in calculations along the Time dimension. This saves time and performance overhead of using member formulas to achieve time-balance functionality.

The following time-balance properties are supported on stored or formula-bearing Accounts dimension members:

- TB First, TB Last, TB Average
- SKIP NONE, SKIP MISSING

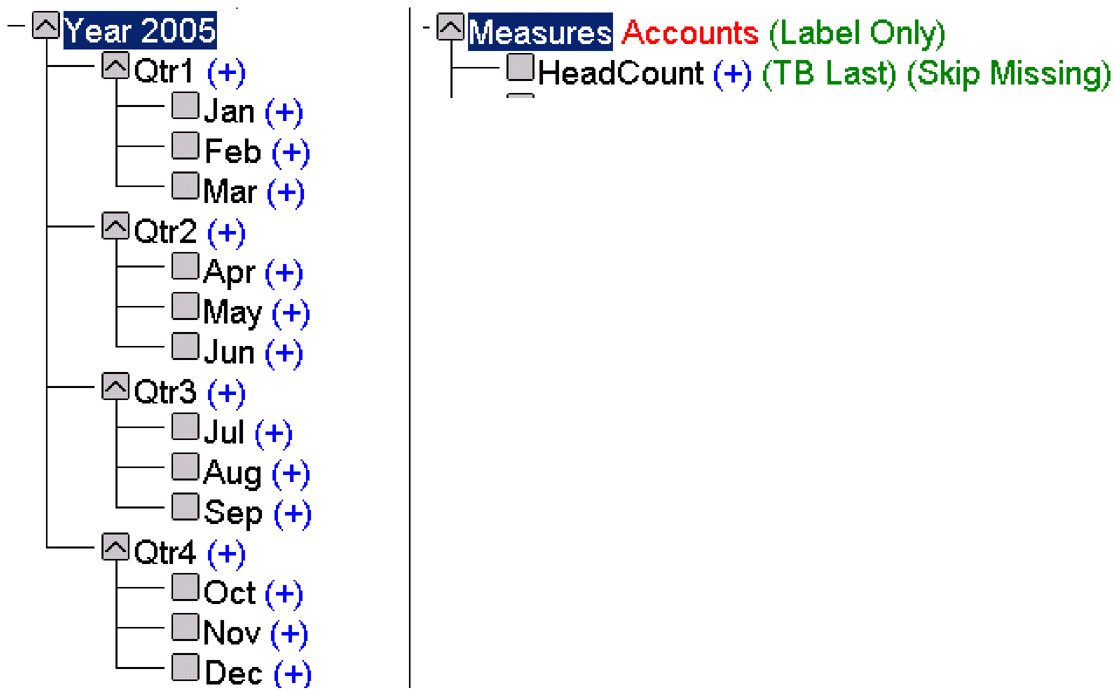
Consider a stored measure such as Headcount in a human-resources application. Within a Year-Quarter-Months hierarchy, Headcount data is loaded at the month level.

The desired yearly or quarterly Headcount value is not the sum of its months; rather, it should be the last recorded value within a time period.

Tagging Headcount as TB Last with SKIPMISSING means that, for Year 2005, its value is the last nonempty value of the headcount for its months. If Dec has a nonmissing Headcount value, then that value will be returned; otherwise, the Nov value will be checked and returned if nonmissing.

If a formula-bearing member has a time balance tag, the formula is executed only for level 0 Time members, and the Time dimension is aggregated according to the time balance tag.

The time balance tags provide a built-in calculation along the Time dimension. To perform other time-based calculations using formulas, such as period-to-date and rolling averages, you can create a dimension called TimeView and write all time-based formulas on that dimension. Doing so enables you to use Time Balance calculation functionality without losing the ability to perform other time-based calculations.



Use Flow Tags in ASO Accounts Dimensions

You can use flow metrics to optimize calculation of time-based accounts in an aggregate storage cube. A flow metric is an unsigned Accounts dimension member that stores periodic and year-to-date values in the outline.

The following example describes the problem to be solved with flow metrics. Assume you have Sales and Additions figures for all 12 months. You want to perform an aggregation to populate each month's beginning inventory.

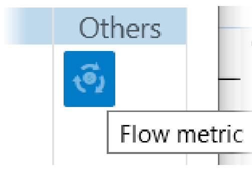
Table 38-6 Inventory Calculation

	Sales	Additions	Inventory
Jan	5	1	50
Feb	6	3	46
Mar	4	2	43
Apr	7	0	41
...	...	...	...

You would use an MDX formula on the Beginning Inventory member in order to calculate its value. Without flow metrics, to obtain each month's beginning inventory, the calculator engine would have to reiterate the MDX formula exponentially.

```
Inventory = SUM(MemberRange(Jan:Time.CurrentMember), (Additions - Sales)) +
Beg_Inventory
```

To optimize the illustrated example, assign the Inventory member the formula **(Addition – Sales)**, and tag the member as Flow, using the outline editor in the Essbase web interface.



Before you tag a member as Flow, ensure that it:

- is a member of the Accounts dimension
- has a formula
- has a Time Balance tag

Restrictions on Alternate Hierarchies for Time Balance and Flow Metrics

If alternate hierarchies are used in the aggregate storage time dimension, the following restrictions apply when using Flow and TB tags on the Accounts dimension:

1. The shared level among alternate time hierarchies must be level 0.
2. The order of members at shared level among alternate time hierarchies must be the same in all alternate hierarchies.
3. The number of shares for a stored member must not exceed that of its previous sibling. Previous sibling means the prior stored member in the outline, found at the same generation.

   For example, consider the following Year dimension. The alternate hierarchies are not valid, because the number of shares for April (3) is greater than the number of shares for March (1).

Aggregate Time-Balance Tagged Measures

Use the MDX Aggregate function to aggregate measures tagged with time balance tags.

Effect of Attribute Calculations on Time Balance Measures in ASO Cubes

The following calculation logic applies if

1. The aggregate storage outline contains a time dimension or date-time dimension with one or more attribute or linked-attribute dimensions.
2. You perform queries on time balance tagged measures.

If the above cases are both true, MDX Aggregate() semantics are used to evaluate the cells.

For example, consider a scenario in which:

- Year is a time-date dimension with a day level hierarchy.
- Holiday is an attribute dimension on Year, with each date tagged with Holiday_TRUE or Holiday_FALSE.
- Opening Inventory is tagged as TBFirst.

The value of (Year, Holiday_TRUE, [Opening Inventory]) is evaluated according to the following MDX logic:

```
Aggregate( {Set of dates that are holidays in Year}, [Opening Inventory])
```

Retrieve Aggregate Storage Data

Data retrieval (query) from Essbase aggregate storage (ASO) cubes, as with block storage (BSO) cubes, can be done using Smart View, MDX, MaxL, Report Writer, or other grid clients. Some differences should be expected between ASO and BSO queries.

The following programs and tools enable data retrieval from aggregate storage cubes:

- Smart View
- MDX queries
- The **export data using report_file** MaxL statement

> **Note:**
>
> Commands that support block-storage-only features (for example, the <SPARSE Report Writer command) cannot be used with aggregate storage databases. MDX queries fully support aggregate storage features.

Attribute Calculation Retrievals

Aggregate storage applications support only the Sum member of the Attribute Calculations dimension. If you specify any other member name from the Attribute Calculations dimension, such as Min or Avg, an error is returned.

Related Topics

Query Differences with ASO

Understanding the Attribute Calculations Dimension

Export Data (Aggregate Storage)

Overview of MDX

39 Custom Calculations and Allocations on Aggregate Storage Cubes

Custom calculations extend the analytical capabilities of Essbase by enabling the execution of recurring calculations on aggregate storage (ASO) cubes. You can also use custom allocations in the budgeting process to distribute revenues or costs.

The information in this section applies only to aggregate storage cubes.

Custom calculations and custom allocations share the following functionality:

- Aggregate storage data load buffers
- Credit and debit processing
- Offset handling

You can use MDX Insert to perform custom calculations and allocations. Enable this feature at the application level using the CUSTOMCALCANDALLOCTHRUINSERT configuration setting. Enable it for a calculation script using the USE_MDX_INSERT command.

Essbase also provides API functions and structures for performing custom calculations and custom allocations. The following APIs validate the syntax of MDX expressions used in performing custom calculations and allocations:

- Java API: IEssCube.verifyMDXExpression
- C API: EssVerifyMDXExpression

Topics in this section:

Custom Calculations on Aggregate Storage Cubes

Custom calculations extend the analytical capabilities of Essbase by enabling the execution of recurring calculations on aggregate storage (ASO) cubes. You can target certain cube areas to calculate, providing debit, credit, and offset information for accounting.

You can write custom calculations for aggregate storage cubes that update target level 0 cells. Custom calculation scripts are expressed in MDX.

Using custom calculations, you can do basic math on account balances in a general ledger and write the results to targeted level 0 members of an Essbase aggregate storage cube. You can perform calculations on account balances or on fixed amounts and can be scheduled to repeat every accounting period.

Custom calculations can be useful when the cube is used for general ledger reporting, where double-entry accounting is in effect. Debit items, such as assets and expenses, must balance with credit items, such as equity and revenue.

Use the following workflow to create and execute custom calculations:

- Create a calculation script expressed in MDX.
- Select an area of the cube where the calculation will be executed. You provide the area at execution time using the target and POV (point of view) parameters.
- If you use debit and credit processing, select the debit and credit members in the outline to write the positive and negative values. You provide these parameters at execution time.
- If you use offsetting entries, select the area where offsetting entries should be made. You provide this parameter at execution time using an MDX tuple. If an offset is not specified or is empty, the offset calculation is not performed.

  > **Note:**
  >
  > In general ledger bookkeeping, an offsetting entry is a counterbalancing measure on the opposite side of the ledger; for example, a $100 credit in January may have a $100 offset added to the debit side of the ledger, so the ledger can be balanced in preparation for an upcoming expense of that amount.

- Execute the custom calculation script.

List of Custom Calculations Criteria

When you design custom calculations for Essbase aggregate storage (ASO) cubes, the critera to consider include: POV (context region), the MDX calculation script, the target (where results are written), the source region (referenced by formulas), an optional offset value, and optional credit and debit members.

Custom calculation functionality depends on a variety of specified criteria. Review these terms before continuing.

Table 39-1 Description of Custom Calculations Criteria

Criteria	Description
POV	A symmetric region in the database that describes the context in which custom calculations are performed. Attribute members cannot be used for this argument.
Calculation script	A calculation script expressed in MDX. Attribute members cannot be used in the left side of the equation.
Target	A tuple argument expressed in MDX that defines the region in the cube where calculation results are written. This argument is combined with left side of each formula and the offset to determine where the results and offset values are written. Attribute members cannot be used for this argument.

Table 39-1 (Cont.) Description of Custom Calculations Criteria

Criteria	Description
(Optional) Offset	The location in the database where an offsetting value for each source amount is written. Attribute members cannot be used for this argument.
(Optional) Credit and debit members	In double-entry accounting, balancing journal entries for one transaction. Both are MDX member expressions. The debit member indicates a member to which positive result values are written, and the credit member indicates a member to which negative result values are written. Attribute members cannot be used for this argument.
Source region	An MDX set expression specifying the region of the cube referred to by the formulas in the script.

Write and Run ASO Custom Calculations

Write and execute custom calculations for Essbase aggregate storage cubes to make updates to target level 0 cells. Custom calculation scripts are expressed in MDX. To run the scripts, use MaxL **execute calculation**.

Writing Custom Calculations

A custom calculation script is a file that you create and store with a `.csc` extension. Create the custom calculation script with one or a series of tuple-expression pairs in MDX, terminated by semicolons. The syntax:

```
tuple := numeric_value_expression;
```

The *tuple* is an MDX specification of one or more members where no two members can be from the same dimension. The tuple must be on the left side of the equation and is the primary factor in determining where results of the custom calculation are written.

Only member names are allowed in the tuple expression. The use MDX functions is not supported for custom calculation scripts.

> **Note:**
>
> The secondary factor determining the target for results is the *target* parameter, and the third factor is the *POV* parameter. You specify the second and third parameters at calculation execution time, rather than as part of the calculation script.

The *numeric_value_expression* is a simple MDX numeric value expression, such as a number or an arithmetic operation. The expression must be on the right side of the equation. Only arithmetic operators are permitted. An error is returned if non arithmetic operators (such as AND, OR, or IF statements) are used.

Member names can be used in the numeric value expression, but the use of member functions is not supported for custom calculation scripts.

Attribute members cannot be used on the left side of the equation in a custom calculation script.

You must also define the source region, which serves as a performance hint for Essbase. Essbase pre-fetches the data specified in the source region, and uses that to perform the calculation specified in the script.

Executing Custom Calculations

You can run custom calculations using the MaxL execute calculation (aggregate storage) statement.

You can also run custom calculation scripts using the API, by calling the Java API method IEssPerformCustomCalc.performCustomCalc, or the C API function EssPerformCustomCalcASO.

You can also use Oracle Hyperion Calculation Manager to design a custom calculation and deploy it for execution to Enterprise Scheduling Services.

Sample Use Case for Custom Calculations

This sample use case for a custom calculation on an Essbase aggregate storage (ASO) cube utilizes a credit member, a debit member, and an offset to perform business rental expense allocations.

Consider an outline with the following dimensions:

- **Company**, containing CompanyA, CompanyB, and other children.
- **Department**, containing numbered departments such as 101, 102, 103.
- **Account**, in which Account 5740 is a rent expense account and SQFT is a statistical account used to record square footage for each department.
- **Scenario**, in which the Actual member is where data is posted, and the Allocation member is where allocations and custom calculations are stored. The Scenario member is a parent that aggregates the child members Actual and Allocation.
- **Year**, a time dimension organized by months and quarters.
- **Geography**, a dimension organized by states and cities.
- **AmountType**, in which Debit is the target, and Credit is the offset.
- **Project**, a dimension containing projects such as Proj1, Proj2.

The POV is an MDX set expression indicating where the custom calculation should be executed. It is specified as follows:

```
CrossJoin( { ( [Company], [101], [Jan], [Scenario] ) },
           Descendants( Geography, Geography.Levels(0)) )
```

The DebitMember is an MDX member expression indicating a debit member to which positive result values should be written. It is specified as `[BeginningBalance_Debit]`.

The CreditMember is an MDX member expression indicating a credit member to which negative and offsetting result values should be written. It is specified as `[BeginningBalance_Credit]`.

> **Note:**
>
> The offset is written to the debit member in the case that the sum of all result values is negative.

The offset is an MDX tuple expression indicating where offsetting entries should be made. It is specified as `([Account_NA], [Project_NA])`.

The offset expression is combined with Target and POV to determine the location where offsetting entries are made. If dimensions overlap, the order for resolving the offset location is the offset, the target, and the POV, in that order.

The target is an MDX tuple expression indicating where to write the results of the custom calculation. It is specified as `(Allocation)`.

The target expression is combined with POV, and the left side of each line in the custom calculation script, to determine the location where results are written. If dimensions overlap, the order for resolving the target location is the left side of the equations, the target, and the POV, in that order. In this example, results are written to the Allocation member, because the target overrides the Scenario member specified in the POV.

The following is an example of a custom calculation script:

```
(AccountA,Proj1) := 100;
([AccountB], [Proj1]) := ([AccountB], [Proj1]) * 1.1;
(AccountC,Proj1) :=
    ((AccountB,Proj1,2007) + (AccountB, Proj1)) / 2;
(AccountA,Proj2) :=
    ((AccountD,Proj1) +
      (AccountB,Proj2)) / 2;
```

For each combination in the POV,

- The calculation script is executed in the context of the current POV combination.
- One offset value is written to the target location.

> **Note:**
>
> Each formula (line in the calculation script) is executed simultaneously, rather than sequentially. Therefore, you cannot use the result of one formula in a subsequent formula.

To define the source region, examine the custom calculation script and determine which members are referenced on the right sides of equations. At a minimum, the source region should include all members from the right sides of the assignment statements in the custom calculation script.

Define the source region as a single MDX set. If the members on the right sides of the equations are from more than one dimension, you can use CrossJoin to create the set from two sets. CrossJoin only accepts two sets, so you may have to use nested CrossJoins.

The source region for the above custom calculation script is:

```
Crossjoin(
 {[AccountB], [AccountD]},
   Crossjoin(
    {[Proj1], [Proj2]}, {[2007]}
   )
)
```

It is not necessary to include any members in the source region that are not assigned in the script. For example, if you added to the source region an `[AccountC]`, which is not used in the script, then it would be ignored, and could cause a slight detriment to performance.

It is not necessary to account for numbers in the source region. For example, the following assignment in a custom calculation script requires nothing to be added to source region: `([Bud Var]):=10`.

Optimize Custom Calculations with NONEMPTYTUPLE

Using the NONEMPTYTUPLE property, you can optimize an Essbase aggregate storage (ASO) custom calculation script to conserve memory resources by skipping empty tuples when calculating over large, sparse data sets.

Because large data sets can be very sparse, using the NONEMPTYTUPLE property in custom calculation scripts can optimize your script to conserve memory resources.

You create a custom calculation script with one or a series of tuple-expression pairs in MDX, terminated by semicolons. You can optionally filter out empty result sets from being calculated, by including the NONEMPTYTUPLE property clause in the custom calculation script.

Using the NONEMPTYTUPLE property clause in a custom calculation script indicates to Essbase that the cell value being calculated for a tuple is empty whenever the given *nonempty_member_list* is empty.

Syntax

```
use_optimized_way;
tuple := [NONEMPTYTUPLE (nonempty_member_list)] numeric_value_expression;
```

Where

- use_optimized_way—a literal keyword, required to enable the use of NONEMPTYTUPLE property in the calculation script. If omitted, NONEMPTYTUPLE directives are ignored.
- *tuple*—an MDX specification of one or more members, where no two members can be from the same dimension.
- NONEMPTYTUPLE—an optional property you can use to optimize calculation performance. If used, then you must follow this literal property with *nonempty_member_list*.
- *nonempty_member_list*—one or more comma-separated member names from different dimensions.
- *numeric_value_expression*—a simple MDX numeric value expression, such as a number or an arithmetic operation. The expression must be on the right side of the equation. Only

arithmetic operators are permitted. An error is returned if non arithmetic operators (such as AND, OR, or IF statements) are used.

Examples

The following custom calculation script examples include a NONEMPTYTUPLE property clause to filter out empty tuples from being included in the calculation pass.

```
use_optimized_way;
([Balance].[Net Balance].[Net Change].[Allocation Out]):= NONEMPTYTUPLE
([Balance].[Remainder],[Rule]) -(([Balance].[Remainder],
[Rule])*(20.24000/100));


([2014], [August], [Actual]):= NONEMPTYTUPLE ([2014], [January], [Actual])
([2014], [January], [Actual]);
```

Further Information

MDX Optimization Properties

Custom Allocations on Aggregate Storage Cubes

Custom allocations enable you to allocate a given source amount to a target range of cells in an Essbase aggregate storage (ASO) cube. Allocations are used in the budgeting process to distribute revenues or costs.

The source amount can be allocated to the target proportionately, based on a given basis, or the source amount can be spread evenly to the target.

You can perform ASO allocations using the MaxL execute allocation statement.

You can also perform ASO allocations by using the using the API, by calling the Java API method IEssPerformAllocation.performAllocation or the C API function EssPerformAllocationAso.

A single allocation has its own *POV* (point of view), *range*, *amount*, *basis*, *target*, and, optionally, *offset*. To perform allocations with different sets of values for these parameters, you must make a sequence of individual API calls.

Allocations are performed in the allocation engine and then written back to the cube using temporary data load buffers that Essbase creates.

List of Allocation Criteria

When you design custom allocations for an Essbase aggregate storage (ASO) cube, you base them on many required and optional criteria, including POV, range, amount, basis, target, allocation method, and rounding method.

Allocations are based on a variety of specified criteria. This topic provides a brief description of the criteria used to define allocations. Review these terms before continuing.

Table 39-2 Description of Allocation Criteria

Criteria	Description	See
POV	A symmetric region in the database that describes the context in which allocations are performed	Setting the POV
Range	A symmetric region in the database in which allocated values are calculated and written	Setting the Range
(Optional) Excluded range	Locations in the range where you do not want allocation values written	Setting the Range
Amount	The amount to be allocated	Setting the Amount
(Optional) Amount context	Additional context, or specificity, for the amount	Setting the Amount
(Optional) Amount time span	One or more time periods to be considered for the amount	Setting the Amount
(Optional) Zero amount options	Treatment of zero or #MISSING amount values	Setting the Amount
Basis	When combined with the range, defines the location of basis values that determine how the amount is allocated	Setting the Basis
(Optional) Basis time span	One or more time periods to be considered for the basis	Setting the Basis
(Required if basis time span is set) Basis time span option	Method for calculating the basis across the basis time span: • Combine • Split	Setting the Basis
Zero basis options	Treatment of zero basis values	Setting the Basis
(Optional) Negative basis options	Treatment of negative basis values	Setting the Basis
Target	When combined with the range, defines the region in the database where allocation values are written	Setting the Target
(Optional) Target time span	One or more time periods to be considered for the target	Setting the Target
(Required if target time span is set) Target time span option	Method for allocating values across the target time span: • Divide the amount • Repeat the amount	Setting the Target
Allocation method	Method for allocating the amount: • Share: Allocates the amount proportionately to the basis values • Spread: Allocates the amount evenly	Setting the Allocation Method
(Optional) Spread skip options	For spread allocation method, whether to skip basis values in the range that are zero, #MISSING, or negative	Setting the Allocation Method

Table 39-2 (Cont.) Description of Allocation Criteria

Criteria	Description	See
Rounding method	Whether to round allocated values. If you choose to round, specifies the method for handling rounding errors: • Discard rounding errors • Add the total rounding error to: – The greatest allocated value – The lowest allocated value – A specific location	Setting the Rounding Method
(Optional) Round digits	The number of decimal places to which allocation values are rounded: • To the nearest integer • To a specified number of decimal places • To a power of 10	Setting the Rounding Method
(Required if rounding method is set to a specific location) Round to location	The location to which to add the total rounding error	Setting the Rounding Method
(Optional) Offset	The location in the database where an offsetting value for each source amount is written	Setting the Offset
(Optional) Credit and debit members	In double-entry accounting, balancing journal entries for one transaction	Balancing Allocations

Regions in ASO Allocations

Essbase uses source, target, basis, and offset regions in the aggregate storage (ASO) cube when performing custom allocations. Each region consists of at least one member from each dimension defined in the region.

Table 39-3 List of Regions Used in Allocations

Region Name	Region Definition	Description
Source	(*POV* X *amount* [X *amount context*] X [*amount time span*])	The region containing the amount values that are to be allocated. The source region and target region cannot overlap.
Target	(*POV* X *target* X *debit member/credit member* X *range* X [*target time span*])	The region containing the locations to which allocated values are written. The source region and target region cannot overlap. The target region need not need be empty before performing an allocation. Essbase overwrites non-empty cells either with allocation data or with zeros. For cells with #MISSING, the cells remain #MISSING unless Essbase writes allocation data to those cells.

Table 39-3 (Cont.) List of Regions Used in Allocations

Region Name	Region Definition	Description
Basis	(*POV* X *basis* X *range* X [*basis time span*])	The region containing the basis values that are used to determine how the source amount is allocated. Basis might override part of the POV.
Offset	(*POV* X *offset* X *debitMember/ creditMember*)	The region containing the locations to which offset values are written.

Specifying Allocation Criteria

Use MDX syntax when you specify criteria for custom allocations in an Essbase aggregate storage (ASO) cube.

Allocation parameter values can be expressed in the following ways:

- MDX member expression
- MDX set expression
- MDX tuple expression (where no two members can be from the same dimension)
- Constant

For more information about how to express allocation parameter values, see the the MaxL execute allocation statement or *Oracle Essbase and Provider Services Java API Reference*.

Shared Members in Allocation Parameters

When shared members are specified in allocation parameters, Essbase maps the shared members to their prototype members before performing the allocation.

Duplicate Members in Allocation Parameters

When duplicate members are specified in allocation parameters, either because a member name is repeated or because a member and its shared member are both specified, Essbase removes the duplicate members and issues a warning.

Setting the POV

Essbase custom allocations for aggregate storage (ASO) cubes require a POV to be given as one of the allocation criteria. *POV* specifies a symmetric region in the cube that describes the context in which allocations are performed.

The POV can consist of only level 0 members. The dimensions defined in the POV cannot be used in other parameters, except for the basis and the basis time span.

The allocation is repeated for every combination of members in the POV set. The number of POV combinations is the product of the number of members from dimensions with multiple members. (Dimensions with only one member are not used to calculate the number of combinations.)

For example, assume that the POV consists of two dimensions (CostCenter and Project), and allocations are to be made to two cost centers (CostCenter1 and CostCenter2) and three projects (Project1, Project2, and Project3). The number of POV combinations is six:

```
Project1,CostCenter1
Project1,CostCenter2
Project2,CostCenter1
Project2,CostCenter2
Project3,CostCenter1
Project3,CostCenter2
```

The allocation is repeated six times by successively setting the allocation context to each combination.

Values considered as the basis for the allocation are dependent on the POV combination.

Note:

If time periods are specified in the POV, you cannot use the amount time span and the target time span options.

Setting the Range

Essbase custom allocations for aggregate storage (ASO) cubes require a range to be given as one of the allocation criteria. *Range* specifies a symmetric region in the cube in which allocated values are calculated and written.

If you do not want allocation values written to certain cells within the range, use the *excluded range* parameter to express a symmetrical subset of the range. Even when excluding a subset of the range, Essbase uses all cells in the range to calculate allocated values.

When you exclude cells from the range, the sum of allocated values might be less than the value of the amount.

The following examples assume that the range consists of six member combinations, the amount is 6, and the allocation method is spread, in which Essbase evenly allocates the amount across the range. The allocation spread amount is 1 (6/6 = 1).

As illustrated below, Essbase writes 1 in each cell in the range.

Table 39-4 Example: Allocating the Amount to Each Member in the Range

-	CostCtr1	CostCtr2
Project1	1	1
Project2	1	1
Project3	1	1

As illustrated below, if the excluded range is set to the member combination of (Project2,CostCtr2), Essbase does not write the allocation spread amount to that cell. Therefore, the sum of allocated values (5) is less than the amount (6). The value of the excluded cell after the allocation process is either #MISSING or zero.

Table 39-5 Example: Allocating the Amount to Only Some Members in the Range

-	CostCtr1	CostCtr2
Project1	1	1
Project2	1	
Project3	1	1

The range and excluded range can consist of only level 0 members.

Setting the Amount

Essbase custom allocations for aggregate storage (ASO) cubes require an amount to be given as one of the allocation criteria. *Amount* specifies the source of the allocation. The amount value is allocated to cells in the target region.

The amount, which can consist of upper-level or level 0 members, can be expressed as a numeric value expression, a tuple, or a constant.

How you express the amount determines certain requirements:

- Numeric value expression:
  - All members in the expression must be from the same dimension.
  - Tuples cannot be used in the expression.
  - Only arithmetic expressions (+, -, /, and *) can be used in the expression.
  - MDX functions (such as Avg and Parent) are not allowed.

  For example:

  ```
  (Acc_1000 + Acc_2000)/2
  ```

  ```
  AccA + AcctB
  ```

  ```
  Balance * 1.1
  ```

- Tuple:
  - The tuple must use one member from every dimension that is not specified in the POV.
  - The amount context must be empty.

  For example:

  ```
  (Balance,Cost_Center_00,Project_00)
  ```

  ```
  (Balance,Cost_Center_00,Actual)
  ```

- Constant:
  - The amount context must be empty.
  - The amount time span must be empty.

For example:

```
100
```

You can use these parameters to further define the amount:

- **(Optional)** *Amount context* provides additional context, or specificity, for the amount. The amount context, which can consist of upper-level or level 0 members, can be expressed as a tuple. By specifying the amount context, you can include a member from a dimension that is not specified in the POV.

  When using amount context, these requirements apply to the amount and the amount context:

  – The parameters cannot refer to members in the same dimensions.

  – Together, the parameters must use members from every dimension not specified in the POV.

- **(Optional)** *Amount time span* specifies one or more time periods to be considered for the amount. The amount value is aggregated over the specified time periods, and the aggregated amount value is allocated. Time periods must be level 0 members in a Time dimension.

  When amount is specified using an arithmetic expression, and amount time span is used, amount time span takes precedence over any formulas in the amount or any formula members used in the amount. For example, assume that the amount is specified as Dept_A/Dept_B and amount time span is set to Jan, Feb, Mar, and Apr for each department, as shown below. The amount to be allocated for the POV is calculated by dividing the amount time span value for Dept_A (10) by the amount time span value for Dept_B (20), which is 0.5.

  Table 39-6 Example: Amount Time Span Takes Precedence Over Formulas in Amount

Members in Amount Time Span	Dept_A	Dept_B
Jan	1	2
Feb	2	4
Mar	3	6
Apr	4	8
Total	10	20

- **(Optional)** *Zero amount options* specifies how to treat the amount if the value is zero or #MISSING. You can choose to allocate zero values (the default), skip to the next nonzero or non-#MISSING amount value, or cancel the entire allocation operation.

You can use amount context and amount time span to achieve the same result, as shown in following example. The amount is the value of Dept_A, but amount time span is used to focus only on the months of Jan, Feb, Mar, and Apr for Dept_A. As shown below, the aggregated value of the members included in the amount time span (10) is the amount value that is allocated across the cells in the range.

Table 39-7 Example: Amount Time Span

Members in Amount Time Span	Dept_A
Jan	1

Table 39-7 (Cont.) Example: Amount Time Span

Members in Amount Time Span	Dept_A
Feb	2
Mar	3
Apr	4
Total	10

You can achieve the same amount value by specifying amount as an arithmetic expression of Jan + Feb + Mar + Apr and setting the amount context as Dept_A, as shown below:

Table 39-8 Example: Amount Context

Amount Context	Jan	Feb	Mar	Apr	Total
Dept_A	1	2	3	4	10

Setting the Basis

Essbase custom allocations for aggregate storage (ASO) cubes usually require a basis to be given as one of the allocation criteria. *Basis*, when combined with the range, defines the location of basis values that determine how the amount is allocated. The basis can consist of upper-level or level 0 members.

Basis is optional if the allocation method used is spread, and no values are skipped. Basis must be omitted when the allocation method spread is used without skip options.

You can use these parameters to further define the basis:

- **(Optional)** *Basis time span* specifies one or more time periods to be considered for the basis. Time periods must be level 0 members in a Time dimension.
- (Required if basis time span is set) *Basis time span option* specifies how the basis is calculated across the time periods specified by the basis time span. You can choose to use the basis value for each time period individually (*split*) or use the sum of the basis values across the time periods specified by the basis time span (*combine*).
  – If basis time span specifies multiple time periods and the target time span specifies one time period or is empty, you must set the basis time span option to combine. Essbase ignores the target time span option.
  – If basis time span and target time span specifies multiple time periods, and you set the basis time span option to split, the periods specified by basis time span and target time span must be identical. Essbase ignores the target time span option.
- *Zero basis options* specifies how to treat a zero basis value. You can choose to skip to the next nonzero or non-#MISSING amount value or cancel the entire allocation operation. Essbase processes the zero basis options setting based on the allocation method.
- **(Optional)** *Negative basis options* specifies how to treat a negative basis value. The options available for the negative basis options depend on the allocation method used.

> **Note:**
>
> The basis is ignored when using the spread allocation method and you have not set any *spread skip options*.

Setting the Target

Essbase custom allocations for aggregate storage cubes require a target to be given as one of the allocation criteria. *Target*, when combined with the range, defines the region in the ASO cube where allocation values are written.

The target can consist of only level 0 members.

You can use these parameters to further define *target*:

- **(Optional)** *Target time span* specifies one or more time periods to be considered for the target. Time periods must be level 0 members in a Time dimension.
- (Required if target time span is set) *Target time span option* specifies the method for allocating values across the time periods specified in target time span. You can choose to divide the amount value or repeat the amount value across the specified time periods.
  - If basis time span specifies multiple time periods, and the target time span specifies one time period or is empty, you must set the basis time span option to combine. Essbase ignores the target time span option.
  - If basis time span and target time span specifies multiple time periods, and you set the basis time span option to split, the periods specified by the basis time span and target time span must be identical. Essbase ignores the target time span option.

Setting the Allocation Method

When designing custom allocations for aggregate storage (ASO) cubes, you can specify an allocation method to tell Essbase whether to allocate the amount evenly (spread) or proportionally (share).

- The *share* method allocates a percentage of the amount (`alloc_share_amt`) by dividing the basis value for the current member in the range (`basis_mbr_value)` by the sum of the basis across the range (`basis_range_sum`). The allocated amounts are based on the number of valid basis values in the range. The algorithm for calculating the allocation share amount:

  ```
  alloc_share_amt = (basis_mbr_value/basis_range_sum) * amount
  ```

  Basis values and Essbase action:

  - Zero, Essbase writes a zero to the corresponding target cell.

    If the sum of all basis values is zero (which would result in a division-by-zero error), Essbase uses the zero basis options setting.
  - #MISSING, Essbase either leaves the target cell as #MISSING, or, if the target cell already has a value, overwrites the existing value with zero.
  - A negative number, Essbase uses the negative basis options setting. You can choose to use the negative basis value (the default), skip to the next amount value (no data is

allocated for the current amount value, and Essbase skips to the next POV combination), or cancel the entire operation.

The following examples illustrate the share allocation method. In both examples, the amount to allocate is 10.

In the following example, assume that the amount (10) represents the rent expense for a building, and the basis represents the head count of each department in the range. Essbase uses the basis values for departments with non-#MISSING head count (Dept_A through Dept_D) to calculate the allocation share amounts, which is the rent allocation.

The rent allocation for Dept_A is the basis value of Dept_A (3), divided by the sum of valid basis values across the range (3 + 2 = 5), multiplied by the amount (10): 3/5 * 10 = 6. For Dept_D, the rent allocation is 2/5 * 10 = 4. The total of the target cells in the range equals 10.

Table 39-9 Share Allocation Method Example

Members in Range	Basis (Head Count)	Target (Rent Allocation)
Dept_A	3	6
Dept_B		
Dept_C	0	0
Dept_D	2	4

In the following example, assume that all basis values are to be considered in calculating the share allocation amounts. The allocation for Mbr1 is the basis value of Mbr1 (3), divided by the sum of valid basis values across the range (3 + -1 + 2 = 4), multiplied by the amount (10): 3/4 * 10 = 7.5. For Mbr3, the allocation is -1/4 * 10 = -2.5; for Mbr4, the allocation is 2/4 *10 = 5. The total of the target cells in the range equals 10.

Table 39-10 Share Allocation Method Example: Negative Basis Options — Default

Members in Range	Basis	Target
Mbr1	3	7.5
Mbr2	#MISSING	
Mbr3	-1	-2.5
Mbr4	2	5.0

- The *spread* method allocates the amount evenly across the range (`alloc_spread_amt`). The number used to divide the amount and, therefore, the number of target cells where the allocation spread amount is to be written, is based on the number of valid basis values in the range (`#_valid_basis_values`). The algorithm for calculating the allocation spread amount:

  ```
  alloc_spread_amt = amount/#_valid_basis_values
  ```

  When using the spread allocation method, you can use the optional *spread skip options* parameter to skip all basis values in the range that are zero, #MISSING, or negative. You can specify multiple options.

  Basis values and Essbase action:

  – Zero, Essbase writes a zero to the corresponding target cell.

    If spread skip options is set to skip zero, no data is allocated.

  - #MISSING, Essbase either leaves the target cell as #MISSING; or, if the target cell already has a value, Essbase overwrites the existing value with zeros.

    If spread skip options is set to skip #MISSING, no data is allocated.
  - A negative number, Essbase uses the negative basis options setting (which takes precedence over the spread skip options setting of skip negative). You can choose one of the following actions:
    * Use the negative basis value (the default)
    * Skip to the next amount value (no data is allocated for the current amount value)
    * Use the absolute value of the negative number
    * Treat the negative number as $MISSING (no value is allocated to the target cell)
    * Treat the negative number as a zero (zero is allocated to the target cell)
    * Cancel the entire operation

If all basis values have been skipped (which would make the denominator in the allocation zero), Essbase uses the zero basis options setting. See Setting the Basis.

The following examples illustrate the spread allocation method. In both examples, the amount to allocate is 10.

In the following example, assume that the spread skip options parameter is not specified. Therefore, Essbase considers all four basis members in the range. Essbase divides the amount (10), by the number of valid basis members in the range (4), and spreads that value (2.5) to each target cell in the range: 10/4 = 2.5.

Table 39-11 Spread Allocation Method Example: Do Not Skip Basis Values

Members in Range	Basis	Target
Mbr1	2	2.5
Mbr2	#MISSING	2.5
Mbr3	3	2.5
Mbr4	-6	2.5

In the following example, assume that the spread skip options parameter is set to ignore #MISSING and negative numbers. Therefore, Essbase considers only the two basis members with positive values (Mbr1 and Mbr3). Essbase divides the amount (10), by the number of valid basis members in the range (2), and spreads that value (5) to the Mbr1 and Mbr3 target cells: 10/2 = 5.

Table 39-12 Spread Allocation Method Example: Skip #MISSING and Negative Basis Values

Members in Range	Basis	Target
Mbr1	2	5
Mbr2	#MISSING	
Mbr3	3	5
Mbr4	-6	

Setting the Rounding Method

When designing custom allocations for aggregate storage (ASO) cubes, you can specify a rounding method to tell Essbase whether to round allocated values (the default is not to round).

If you choose to round values, the rounding method specifies how to handle rounding errors. You can choose to discard rounding errors, or to round all allocated values and add the total rounding error to the highest allocated value, the lowest allocated value, or to a specific cell. If you choose to add the rounding error to the highest or lowest allocated value, and there are multiple highest or lowest allocated values, Essbase chooses one of the highest or lowest values to which to add the rounding error.

If you choose to round allocation values, you can use these parameters to further define the rounding method:

- **(Required if rounding allocated values)** *Round digits* specifies the number of decimal places to which allocated values are rounded. You can choose to round to the nearest integer (the default), to a specified number of decimal places, or to a power of 10.

  Round digits must be a number from -100 to 100 and can be expressed as an integer, an MDX numeric value expression, or a tuple.

  Using an MDX numeric value expression is helpful when the setting for round digits is based on the currency of the allocated value. For example, assume that the database contains a dimension named Currency, which is part of the POV, and an associated attribute dimension named NumCurrencyDigits, which specifies how to round allocated values based on the currency of the allocated values. You can express round digits as:

  ```
  Currency.currentMember.NumCurrencyDigits
  ```

  > **Note:**
  >
  > If, for the rounding method, you choose not to round allocated values, the round digits parameter value must be 0 (which is the default). If you want to round to 0 digits, the parameter value must be -1.

- **(Required if rounding method is set to a specific location)** *Round to location* specifies a cell to which to add the total rounding error. Expressed as a tuple, the cell must be in the range and have the same dimensionality as the range. Round to location can consist of only level 0 members.

  > **Note:**
  >
  > If, for the rounding method, you choose an option other than to round to a specific cell, the round to location parameter must be empty.

Setting the Offset

When designing custom allocations for Essbase aggregate storage (ASO) cubes, you can optionally specify an offset. *Offset* specifies the location in the cube where an offsetting value for each source amount is written.

Offset works the same for allocations and custom calculations.

Balancing Allocations

When designing custom allocations for Essbase aggregate storage (ASO) cubes, the optional *debitMember* and *creditMember* can consist only of level 0 members.

The debit member and the credit member are optional. If used, they must be two different members from the same dimension.

debitMember and *creditMember* work the same for allocations and custom calculations.

Basis and Target Time Span

When designing custom allocations for Essbase aggregate storage (ASO) cubes, you can specify one or more time periods to be considered for the basis, and one or more time periods to be considered for the target.

The number of members that are specified for the basis time span and target time span affect how Essbase treats the basis time span option and target time span option.

In situations where the basis or target time span is empty, or set to a single time period, Essbase ignores any setting that you might have set for the respective basis or target time span option. In situations where one or both of the basis or target time spans are set to multiple time periods, Essbase requires a particular setting for the respective basis or target time span option.

Table 39-13 Summary: Basis and Target Time Span, and Basis and Time Span Option

Basis Time Span	Target Time Span	Basis Time Span Option	Target Time Span Option	See
Empty or single member	Empty or single member	Ignored	Ignored	Example 1
Empty or single member	Multiple members	Ignored	Divide or repeat	Example 2
Multiple members	Empty or single member	Combine	Ignored	Example 3
Multiple members	Multiple members	Split	Ignored	Example 4
Multiple members	Multiple members	Combine	Divide or repeat	Example 5

Example 1: Basis and Target Time Span—Empty or Single Member

In this example, the basis time span and target time span are not set or are set for only one time period. Essbase ignores any setting you might have chosen for the basis time span option or target time span option.

Example 2: Basis Time Span—Empty or Single Member; Target Time Span—Multiple Members

In this example, the basis time span is not set, and multiple time periods are specified for the target time span. The basis time span option is ignored. For the target time span option, you can select either *divide* or *repeat*.

Assume the amount is 1000. The example below shows the basis for each department (Dept_1 = 1) and the total basis for the range (21):

Table 39-14 Example 2: Basis Values

	Range						Total
	Dept_1	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Basis	1[1]	2	3	4	5	6	21[2]

1 Basis for the member
2 Total basis for the range

The setting for target time span option determines how the allocation is calculated.

- **Repeat the allocated amount across the specified target time span periods:**

  In this scenario, Essbase performs the allocation for a single period and copies the allocated amount value to all members in the target time span.

  The algorithm Essbase uses:

  ```
  alloc_amt = (basis_mbr_value/basis_total_range) * amount
  ```

  As shown below, for Dec 07,Dept_1, the member basis value (1) is divided by the total basis across the range (21), and the result (0.04762) is multiplied by amount (1000): (1/21) * 1000 = 47.62. Essbase copies 47.62 into the cells for Jan 08, Feb 08, Mar 08, and Apr 08. Essbase continues to perform allocations for Dec 07 for each department. For each target time span, the sum of the allocated values across the range equals the amount (1000).

Table 39-15 Example 2: Allocation Using Target Time Span Option Set to Repeat

	Range						Total
Members in Target Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	47.62	95.24	142.86	190.48	238.10	285.71	1000
Jan 08	47.62	95.24	142.86	190.48	238.10	285.71	1000
Feb 08	47.62	95.24	142.86	190.48	238.10	285.71	1000
Mar 08	47.62	95.24	142.86	190.48	238.10	285.71	1000
Apr 08	47.62	95.24	142.86	190.48	238.10	285.71	1000
							5000[1]

[1] Total allocated values

The total allocated amount is the original amount value (1000) multiplied by the number of target time span members (5): 1000 * 5 = 5000.

- **Divide the allocated amount across the specified target time span periods**

  In this scenario, Essbase performs the allocation for one period and evenly divides the allocated amount across all members in the target time span.

  The algorithm Essbase uses:

```
alloc_amt = ((basis_mbr_value/basis_total_range) * amount)/
#_target_time_span_periods
```

  As shown below, for Dec 07,Dept_1, Essbase performs the same calculation as described for the repeat target time span option scenario to arrive at 47.62. However, this amount is evenly divided across all five target time span periods for Dept_1; therefore, 9.52 is written in each target cell: 47.62/5 = 9.52. Essbase continues to perform allocations for each department. For each target time span, the sum of the allocated values across the range equals (200).

Table 39-16 Example 2: Allocation Using Target Time Span Option Set to Divide

	Range						Total
Members in Target Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	9.52	19.05	28.57	38.10	47.62	57.14	200
Jan 08	9.52	19.05	28.57	38.10	47.62	57.14	200
Feb 08	9.52	19.05	28.57	38.10	47.62	57.14	200
Mar 08	9.52	19.05	28.57	38.10	47.62	57.14	200
Apr 08	9.52	19.05	28.57	38.10	47.62	57.14	200
							1000[1]

[1] Total allocated values

The total allocated values across the range is the original amount value (1000): 200 * 5 = 1000.

Example 3: Basis Time Span—Multiple Members; Target Time Span—Empty or Single Member

In this example, multiple time periods are specified for the basis time span, but the target time span is not set. The target time span option is ignored. The only valid choice for the basis time span option is *combine*.

Assume the amount is 1000. As shown below, the basis to be used for each department is the sum of the basis values for the basis time span (Dept_1 = 15); the total basis for the range is the sum of all department basis values (147):

Table 39-17 Example 3: Basis Values

	Range						Total
Members in Basis Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	1	2	3	4	5	6	
Jan 08	2	3	4	5	6	7	
Feb 08	3	4	5	0	7	8	
Mar 08	4	5	6	1	8	9	
Apr 08	5	6	7	2	9	10	
Total	15[1]	20	25	12	35	40	147[2]

1 Basis for each range member summed across the basis time span periods
2 Total basis for the range

The allocation is calculated using the basis time span setting of combine, which uses the sum of the basis values across the basis time span periods.

The algorithm Essbase uses for each range member:

```
alloc_amt = (sum_across_basis_time_span/basis_total_range) * amount
```

As shown below, the allocated value for each department is written to one target location, because the target time span is not set to multiple periods. For the allocated amount for Dept_1, the sum of the basis time span (15) is divided by the total basis for the range (147), and the result (0.10204) is multiplied by amount (1000): (15/147) * 1000 = 102.04. Essbase continues to perform allocations for each department in the range.

Table 39-18 Example 3: Allocation Using Basis Time Span Set to Combine

	Range						Total
	Dept_1	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Target	102.04	136.05	170.07	81.63	238.10	272.11	1000[1]

1 Total allocated values

The total allocated values across the range is the original amount value (1000).

Example 4: Basis and Target Time Span—Multiple Members; Basis Time Span Option—Split

In this example, multiple time periods are specified for the basis time span and the target time span, and the basis time span option is set to *split*. When using the split basis time span option, the periods specified by the basis time span and target time span must be identical.

Assume the amount is 1000. As shown below, the total basis for the range is the sum of all department basis values (165):

Table 39-19 Example 4: Basis Values

	Range						Total
Members in Basis Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	1	2	3	4	5	6	21[1]
Jan 08	2	3	4	5	6	7	27
Feb 08	3	4	5	6	7	8	33
Mar 08	4	5	6	1	8	9	39
Apr 08	5	6	7	2	9	10	45
							165[2]

1 Total basis for each basis time span period
2 Total basis for the range

The allocation is calculated using the basis time span setting of split, which uses the basis value for each time period individually.

The algorithm Essbase uses:

```
alloc_amt = (basis_mbr_value/basis_total_range) * amount
```

As shown below, for Dec 07,Dept_1, the member basis value (1) is divided by the total basis for the range (165), and the result (0.00606) is multiplied by amount (1000): (1/165) * 1000 = 6.06. Essbase continues to perform allocations for each time period for each department.

Table 39-20 Example 4: Allocation Using Basis Time Span Set to Split

	Range						Total
Members in Target Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	6.06	12.12	18.18	24.24	30.30	36.36	127.27
Jan 08	12.12	18.18	24.24	30.30	36.36	42.42	163.64
Feb 08	18.18	24.24	30.30	36.36	42.42	48.48	200.00
Mar 08	24.24	30.30	36.36	42.42	48.48	54.55	236.36
Apr 08	30.30	36.36	42.42	48.48	54.55	60.61	272.73
							1000[1]

1 Total allocated values

The total allocated values across the range is the original amount value (1000).

Example 5: Basis and Target Time Span—Multiple Members; Basis Time Span Option—Combine

In this example, multiple time periods are specified for the basis and target time spans; however, because the basis time span option is set to *combine*, the basis and target time spans need not contain the same member set.

Assume the amount is 1000. As shown below, the basis to be used for each department is the sum of the basis values across the basis time span (Dept_1 = 10); the basis for the range is the sum of all department basis values (113):

Table 39-21 Example 5: Basis Values

	Range						Total
Members in Basis Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	1	2	3	4	5	6	
Jan 08	2	3	4	5	6	7	
Feb 08	3	4	5	6	7	8	
Mar 08	4	5	6	0	8	9	
Total	10[1]	14	18	15	26	30	113[2]

1 Basis for each range member summed across the basis time span periods
2 Total basis for the range

The setting for target time span option determines how the allocation is calculated.

- **Repeat the allocated amount across the specified target time periods:**

  In this scenario, Essbase performs the allocation for a single period and copies the allocated amount value to all members in the target time span.

  The algorithm Essbase uses for each range member:

  ```
  alloc_amt = (sum_across_basis_time_span/basis_total_range) * amount
  ```

  As shown below, for Dec 07,Dept_1, the basis for Dept_1 (10) is divided by the total basis for the range (113), and the result (0.0885) is multiplied by amount (1000): (10/113) * 1000 = 88.50. Essbase copies 88.50 into the cells for Jan 08, Feb 08, Mar 08, and Apr 08. Essbase continues to perform allocations for Dec 07 for each department. For each target time span, the sum of the allocated values across the range equals the amount (1000).

Table 39-22 Example 5: Allocation Using Target Time Span Option Set to Repeat

	Range						Total
Members in Target Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	88.50	123.89	159.29	132.74	230.09	265.49	1000

Table 39-22 (Cont.) Example 5: Allocation Using Target Time Span Option Set to Repeat

	Range						Total
Members in Target Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Jan 08	88.50	123.89	159.29	132.74	230.09	265.49	1000
Feb 08	88.50	123.89	159.29	132.74	230.09	265.49	1000
Mar 08	88.50	123.89	159.29	132.74	230.09	265.49	1000
Apr 08	88.50	123.89	159.29	132.74	230.09	265.49	1000
							5000[1]

1 Total allocated values

The total allocated values is the original amount value (1000) multiplied by the number of target time span members (5): 1000 * 5 = 5000.

- **Divide the allocated amount across the specified target time periods:**

  In this scenario, Essbase performs the allocation for a single period and evenly divides the allocated amount across all members in the target time span.

  The algorithm Essbase uses:

```
alloc_amt = ((basis_time_span/basis_total_range) * amount)/
#_target_time_span_periods
```

  As shown below, Essbase performs the same calculation as described for the repeat target time span option scenario to arrive at 88.50. However, this amount is evenly divided across all five target time span periods for Dept_1; therefore, 17.70 is written in each target cell: 88.50/5 = 17.70. Essbase continues to perform allocations for each department. For each target time span, the sum of the allocated values across the range equals (200).

Table 39-23 Example 5: Allocation Using Target Time Span Option Set to Divide

	Range						Total
Members in Target Time Span	**Dept_1**	**Dept_2**	**Dept_3**	**Dept_4**	**Dept_5**	**Dept_6**	
Dec 07	17.70	24.78	31.86	26.55	46.02	53.10	200
Jan 08	17.70	24.78	31.86	26.55	46.02	53.10	200
Feb 08	17.70	24.78	31.86	26.55	46.02	53.10	200
Mar 08	17.70	24.78	31.86	26.55	46.02	53.10	200
Apr 08	17.70	24.78	31.86	26.55	46.02	53.10	200
							1000[1]

1 Total allocated values

The total allocated values across the range is the original amount value (1000): 200 * 5 = 1000.

Examples of Aggregate Storage Allocations

The following example illustrates how changing the POV combination affects the values considered as the basis for the allocation. The example, which uses the share allocation method, allocates the total rent from the previous year to all cost centers in the current year, based on each cost center's head count. Assume that the aggregate storage database has four dimensions—Departments, Time, CostCenter, and Measures—and the allocation criteria is specified as shown in the following example:

Table 39-24 POV Example: Allocation Criteria

Criteria	Definition
POV	Dept_A, Dept_B
Amount	2007, CCNA, TotalRent Assume that the amount values are: • Dept_A = 1000 • Dept_B = 2000
Basis	Jan 2008, Head count
Target	Jan 2008, RentalAllocation
Range	Level 0 descendants of CostCenter Assume that the range evaluates to the following cost centers: • CostCenter1 • CostCenter2 • CostCenter3 • CostCenter4

The allocation is performed for each of the POV combinations:

- Dept_A
- Dept_B

Each POV combination has its own set of basis values that are used in calculating the allocation: the head count for each cost center in the range and the total Jan 2008 head count, as shown in the following example:

Table 39-25 POV Example: Basis Values for Each POV Combination

	Member Basis Values				**Range Basis Value**
POV	**CostCenter1**	**CostCenter2**	**CostCenter3**	**CostCenter4**	**2008 Head Count Total**
Dept_A	1	2	3	5	11
Dept_B	5	0		10	15

For each POV, Essbase divides the head count of each cost center (the basis value of each member) by the total head count of the range (the basis value of the range), and then multiplies that value by the total rental amount for each department (amount). For example, for Dept_A,CostCenter1, the member basis value (1) is divided by the basis of the range (11), and the result (0.09090909) is multiplied by amount (1000): (1/11) * 1000 = 90.90909. For Dept_B,CostCenter1, the allocated amount is 666.6667: (5/15) * 2000 = 666.6667. The following example shows the allocated share amount for each cost center:

Table 39-26 POV Example: Target Values for Each POV Combination

	Member Target Values				Amount Value
POV	CostCenter1	CostCenter2	CostCenter3	CostCenter4	Rental Allocation Total
Dept_A	90.90909	181.8182	272.7273	454.5455	1000
Dept_B	666.6667	0		1333.333	2000

Sample Use Case for Aggregate Storage Allocations

This sample use case for a custom allocation on an Essbase aggregate storage (ASO) cube uses the share allocation method to proportionally redistribute the total monthly rent expense across departments, based on the square footage each department occupies.

Consider an outline with the following dimensions:

- **Company**: Contains multiple ledgers. The rent expense allocations take place in the Vision US ledger.
- **Department**: Contains the following members:
  - 100, which stores the total monthly rent expense, of $100,000, for Vision US. This amount is proportionally allocated to the children of department 999.
  - 999, which is the parent of the following departments:
    - 101, which receives 45% of the rent allocation
    - 102, which receives 30% of the rent allocation
    - 103, which receives 25% of the rent allocation
- **Account**: Contains the following members:
  - 5740, which is a rent expense account
  - SQFT, which is a statistical account used to record square footage for each department
- **AmountType**: Contains PeriodActivty, which is the parent of the following members:
  - PeriodActivityDebit, which is the target location
  - PeriodActivityCredit, which is the offset location

You can accomplish the rent expense allocation in several ways, each with the same result. Two scenarios are presented. For each scenario, assume that the following parameters are defined as follows:

- **Allocation method**: Share
- **Range**: The descendents of department 999:
  - 101
  - 102
  - 103

  No cells in the range are excluded.

- **Basis**: The square footage of each range member for the period of activity (which is monthly).

  ```
  SQFT,PeriodActivity
  ```

- **Zero amount option**: (Default) Allocate zero amount values.
- **Zero basis option**: If the basis value is zero, cancel the allocation operation.
- **Basis time span option**: (Default) Split, use the basis value for each time period individually.
- **Rounding method**: Round allocation values to the nearest 1,000 and add the total rounding error to department 101.
- **Debit member**: If the sum is positive, write the value to PeriodActivityDebit.
- **Credit member**: Write the value to PeriodActivityCredit.

Scenario 1: Aggregate Storage Allocations

For scenario 1, assume the following parameters are defined as follows:

- **POV**: Consists of one member, Vision US, from the Company dimension.

  Because only one member from one dimension is specified, the POV does not change and, therefore, the allocation is performed only once.
- **Amount**: The source value of the allocation is from the following cross-dimensional member:

  ```
  5740,100,Beginning Balance
  ```

- **Target**: Write allocated values to account 5740 for each department.
- **Offset**: Write the offsetting entry to member 5740,100.

Scenario 2: Aggregate Storage Allocations

For scenario 2, assume the following parameters are defined as follows:

- **POV**: Consists of one member each from two dimensions:
  - Vision US, from the Company dimension
  - 5740, from the Amount dimension

    In this scenario, account 5740 is a part of the POV. In the basis, account 5740 is overridden with member SQFT.

  Because only one member from each dimension is specified, the POV does not change and, therefore, the allocation is performed only once.
- **Amount**: The source value of the allocation is from the following cross-dimensional member:

  ```
  100,Beginning Balance
  ```

- **Target**: Not set. Because the combination of POV, target, debit member/credit member, and range uses members from all dimensions, the target can be empty.
- **Offset**: Write the offset entry to department 100.

Data Consistency and Formulas

When designing custom allocations for Essbase aggregate storage (ASO) cubes, you can use formulas in the amount and the basis, but not in the target. However, because ASO cubes do not support transaction semantics, you might experience data inconsistency issues when using formula members in the amount or the basis.

In the following example, User 1 posts revenue values for a set of departments and User 2 performs an allocation of bonus money, for which the year-to-date revenue for each department is basis of the allocation. The order in which these operations are performed affects the result:

- Scenario 1: User 1 posts revenue before User 2 runs the allocation.

  The allocation results are based on the updated revenue values.
- Scenario 2: User 2 runs the allocation before User 1 posts revenue.

  The allocation results are based on prior revenue values, not on the updated revenue values.
- Scenario 3: User 1 posts revenue and User 2 runs the allocation concurrently.

  The allocation results are based on the updated or prior revenue values, depending on which user operation started first.

  Oracle does not recommend running these operations concurrently when using formula members in the amount or the basis.

Also, assume that an MDX formula is used to calculate the year-to-date revenue for the allocation. The complexity of the formula can affect the result:

- Scenario 4: The year-to-date revenue formula involves members from one dimension and uses only the following arithmetic expressions: +, -, /, and *.

  The allocation results are based entirely on either the updated or prior revenue values.

  Oracle recommends using simple MDX formulas, as described in scenario 4.
- Scenario 5: The year-to-date revenue formula is more complicated than the formula in scenario 4.

  It is possible that some of the allocation results are based on the updated revenue values and some are based on the prior revenue values.

Data Load Buffers for Custom Calculations and Allocations

When performing allocations or custom calculations on an aggregate storage cube, Essbase uses temporary data load buffers. If there are insufficient resources in the aggregate storage cache to create the data load buffers, Essbase waits until resources are available.

Multiple data load buffers can exist on a single aggregate storage cube. The data load buffers that Essbase creates for allocations and custom calculations are not configurable. You can, however, configure the data load buffers that you create for data loads and postings.

If you want to perform allocations and custom calculations concurrently with data loads and postings, set the resource usage for the data load buffers that you create for data loads and postings to a maximum of 0.8 (80%). The lower you set the resource usage setting, the greater the number of allocations and custom calculations that can run concurrently with data loads and postings. You can also configure the amount of time Essbase waits for resources to become available in order to process load buffer operations.

To configure data load buffers, use the **alter database** MaxL statement with the **initialize load_buffer** grammar. You can also optimize aggregate storage data loads using configuration settings.

Offset Handling for Custom Calculations and Allocations

When designing custom allocations for an Essbase aggregate storage (ASO) cube, understanding the principles of double-entry accounting will help you know the purpose of an offset as one of the optional allocation criteria.

In general ledger bookkeeping, an offsetting entry is a counterbalancing measure on the opposite side of the ledger from a transaction of equal value. In this document, an offsetting entry is referred to as an offset.

Specification of an offset is optional. An offset might be needed in case the sum of credits and the sum of debits are not equal. If the sum of credits and debits are not equal, the ledger is unbalanced. In such a case, an offset would serve to balance the ledger.

For example, a $100 credit in January may need a $100 offset added to the debit side of the ledger, so that the ledger can be balanced in preparation for a known upcoming expense of that amount.

An offset is a location you specify in the form of a tuple, to which Essbase writes a value offsetting the result of the custom calculation script.

In the following examples, assume the POV is **Prod1, Prod2, AcctA, AcctB, Jan**.

The following custom calculation script has a sum of 13.

```
mbr1 := 7;
mbr2 := -4;
mbr3 := 0;
mbr4 := 10;
```

Therefore, if an offset is required, it must also be 13. Assume that an offset is written to a member called “Offset_Member.”

```
          Debit     Credit
mbr1          7
mbr2                    4
mbr3          0
mbr4         10
mbr_offset             13
Total        17        17
```

When an offset is used, credit and debit processing is reversed. The following calculation sequence occurs when an offset is used with credit and debit processing:

1. For the given POV, get the sum of results written by the calculation script (in this case, 13).
2. If the sum is positive, write it to the credit member in the target cube.
3. If the sum is negative, change it to a positive and write it to the debit member in the target cube.

Credit and Debit Processing for Custom Calculations and Allocations

When designing custom allocations for an Essbase aggregate storage (ASO) cube, understanding the principles of double-entry accounting will help you know the purpose of debit members and credit members as optional allocation criteria.

Oracle General Ledger uses double-entry accounting, in which every transaction has two journal entries: a debit entry and a credit entry.

Thus, for every transaction, there are two accounts, represented as columns. The two accounts must balance; in other words, the sum of debit column must equal the sum of the credit column.

A debit member can be specified, to which the custom calculation writes positive result values, and a credit member can be specified, to which the custom calculation writes negative and offsetting result values. The debit member and the credit member must be two different members from the same dimension. For example, a dimension called “AmountType” may have two level 0 children named “Credit” and “Debit.”

Whenever the calculation would result in writing a positive number to a level 0 cell in the target cube, the positive value is written to the debit member.

Whenever the calculation would result in writing a negative number to a level 0 cell in the target cube, the sign is changed to a positive and is written to the credit member.

40 Manage Aggregate Storage Applications and Cubes

Review the topics in this chapter to better understand how to manage Essbase aggregate storage (ASO) applications and cubes. You can improve the performance of aggregations, reduce restructuring overhead, export data, monitor the ASO cache, and list statistics.

- Aggregate Storage Cache
- Aggregate View Build Optimization
- Aggregate Storage Cube Restructuring
- Export Aggregate Storage Cubes
- Stored Levels in an Aggregate Storage Outline

The information in this chapter applies only to aggregate storage cubes and is not relevant to block storage cubes.

Defining and executing aggregations requires Application Manager Permission or higher. Dimension builds that clear cube values require Database Manager Permission.

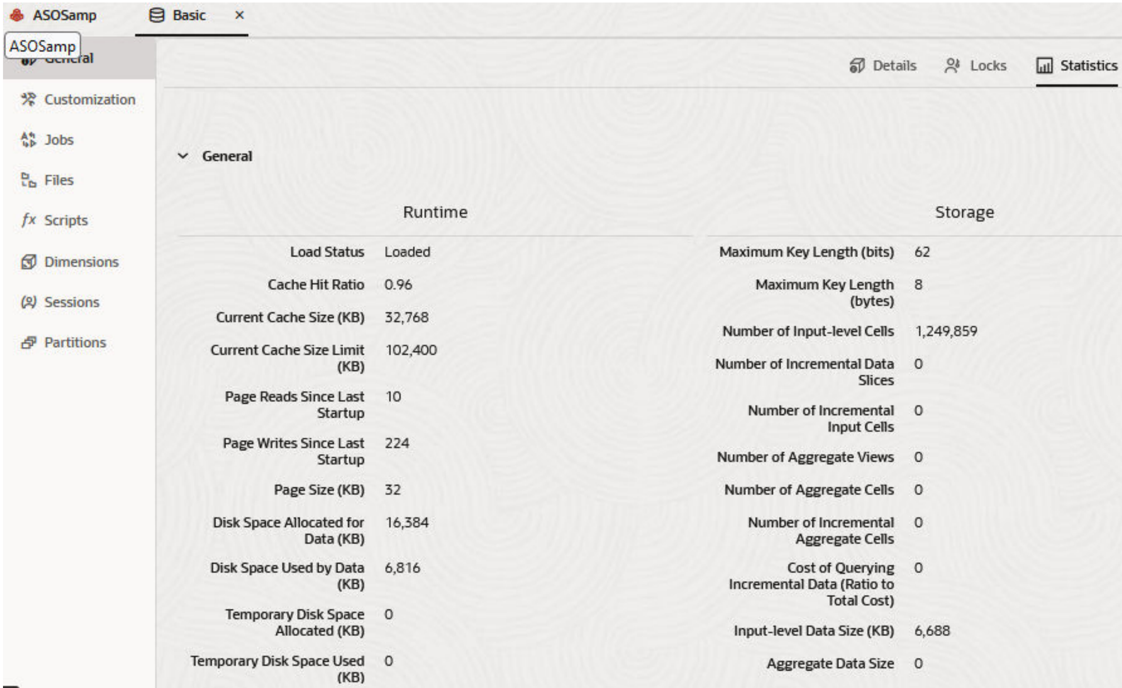
Aggregate Storage Cache

Monitor the Essbase aggregate storage cache size as needed. The cache facilitates memory usage during data loads, aggregations, and retrievals.

When an aggregate storage outline is started, a small area in memory is allocated as the aggregate storage cache for the relevant application. As additional cache area is needed, the cache size incrementally increases until the maximum cache size is used or the operating system denies additional allocations.

> **Note:**
>
> Denial of aggregate cache memory allocations does not deny increased use of existing memory.

You can view the current aggregate storage cache memory allocation and the maximum aggregate cache size setting.



Changing the cache size may optimize memory use. The default maximum cache size, 32 MB, is its minimal setting.

You can use the size of input-level data to determine when to increase the maximum size for the cache. In the Essbase web interface, the statistic is called **Input-level Data Size**. In MaxL you can get the same information from the statement query database (with keywords **get cube_size_info**).

A 32 MB cache setting supports a database with approximately 2 GB of input-level data. If the input-level data size is greater than 2 GB by some factor, the aggregate storage cache can be increased by the square root of the factor. For example, if the input-level data size is 3 GB (2 GB * 1.5), multiply the aggregate storage cache size of 32 MB by the square root of 1.5, and set the aggregate cache size to the result: 39.04 MB.

For aggregation materialization performance, consider the number of threads set for parallel calculation. The aggregation materialization process uses multiple threads that divide the aggregate storage cache. Increasing the number of threads for aggregate storage applications or cubes may require an increase in aggregate storage cache size.

To view aggregate storage cache size, you can use the query application MaxL statement, and to set it, use alter application.

Note:

Setting the number of threads higher than the number of processors may improve aggregate storage application performance.

Do not increase the maximum size of the aggregate storage cache beyond what is needed.

A changed aggregate storage cache setting becomes effective when the application is restarted.

Aggregate View Build Optimization

To improve the performance of building aggregates on an Essbase ASO cubes, you can increase the size of the aggregate storage cache, or increase the sample size.

You might encounter the following message while building aggregate views on an aggregate storage cube:

```
For better performance, increase the size of aggregate storage cache
```

This message sometimes occurs when the cube is larger than a few hundred million input cells.

To improve the performance of building aggregates, take the following steps.

1. Increase the size of the aggregate storage cache to at least 512 MB or 20% of the input data size, whichever is smaller. (If the cache setting is already greater than this amount, proceed to the next step.) Use the MaxL statement alter application with **set cache_size** grammar to increase the cache. For example:

   ```
   alter application appname set cache_size xMB
   ```

   This setting takes effect after you restart the application.

2. If you still see the message when building aggregate views after increasing the aggregate storage cache, use the ASOSAMPLESIZEPERCENT configuration setting. Syntax:

   ```
   ASOSAMPLESIZEPERCENT [appname [dbname]] n
   ```

   Gradually increase the *n* value until the message disappears and optimal aggregation performance is reached. For a cube that contains:

   - 20 million input cells, start with 5%
   - 100 million cells, start with 1%
   - More than 1 billion cells, start with 0.1%

   Clear the aggregate views; then reselect and rebuild them. If the message still appears, increase the setting and try again.

   Performance of building aggregate views may not improve until the message no longer occurs. When the message no longer occurs and performance no longer improves, stop increasing the setting.

   > **Note:**
   >
   > If you increase the ASOSAMPLESIZEPERCENT setting too high, performance will start to degrade again. The optimal setting for a cube larger than 1 billion cells will probably be less than 3%.

Aggregate Storage Cube Restructuring

Essbase ASO cube restructures may be forced by some outline changes, including changes to hierarchies. Learn about the types of hierarchies.

A hierarchy is a top member and its descendants.

- A dynamic hierarchy includes only one stored level. In ASOsamp.Basic, the Accounts dimension is a dynamic hierarchy.
- An attribute dimension is one hierarchy. The generation 1 member is the top member of the hierarchy.
- If a standard dimension is not tagged as multiple hierarchies enabled, it is one hierarchy. The generation 1 member is the top member of the hierarchy.
- If a standard dimension is tagged as multiple hierarchies enabled, it contains multiple hierarchies, and the generation 2 members are the top members of the hierarchies. For example, the Products dimension in ASOsamp.Basic contains two hierarchies. The top members are the generation 2 members All Merchandise and High End Merchandise.

What outlines changes affect:

- Whether data must be cleared before restructuring
- The time and storage required to restructure

Levels of Aggregate Storage Cube Restructuring

When you edit the aggregate storage outline, anticipate the restructuring cost. Full Essbase ASO cube restructures are initiated after certain outline changes. Other changes result in light restructures only. Full restructures can require clearing of data or views, and have a higher impact than light restructures.

To minimize the time and storage needed for restructures, if an outline changes frequently, analyze the outline and the types of outline changes.

Adding, deleting, and moving dimensions has the highest restructure cost: you must reload input data, select aggregate views, and run aggregations.

The following table lists levels of restructuring for aggregate storage cubes, from most to least expensive (in regard to time, storage, and data):

Table 40-1 Aggregate Storage Restructuring Levels

User-Outline Changes	Essbase-Restructure Level	Performance Impact
Add, delete, or move a standard dimension	Clears data and aggregate views, and performs full outline restructure	Very high User must reload input (level 0) data, select the aggregate views, and rerun the aggregation.
• Add, delete, or move a hierarchy. • Change the number of stored levels in a hierarchy. See: – Example: No Change in the Number of Stored Levels in a Hierarchy – Example: Change in the Number of Stored Levels in a Hierarchy • Change the top member of a stored hierarchy from label-only to stored or from stored to label-only. • Change a dynamic hierarchy to a stored hierarchy or a stored hierarchy to a dynamic hierarchy. • Change a primary or an alternate hierarchy so that it matches or no longer matches its primary or alternate hierarchy. All level 0 members of a primary hierarchy must be represented directly or indirectly (for example, a parent that is a sum of its children may represent its children) in all alternate hierarchies. The top level of the primary hierarchy must equate to the top level of each alternate hierarchy. See Example: Changes in Alternate Hierarchies.	Clears aggregate views, and performs full outline restructure	Very high Storage requirement is up to three times the size of the database file (`.dat` file). User must select the aggregate views and rerun the aggregation.
Perform a change that is not included in other categories; for example, delete or move a member, or add a member that is not the last of its siblings	Performs full outline restructure	High Storage requirement is up to three times the size of the database file (`.dat` file).
Perform a light restructure change (described below) to an alternate hierarchy or an attribute dimension	Rebuilds all aggregate views that are based on attribute dimensions or alternate hierarchies	Low Storage requirement is up to three times the size of the affected views. Such aggregate views normally exist only if you used query tracking to select views based on usage. See Select Views Based on Usage.

Table 40-1 (Cont.) Aggregate Storage Restructuring Levels

User-Outline Changes	Essbase-Restructure Level	Performance Impact
On nonattribute dimensions without stored level 0 members (for example, all level 0 members are shared or have formulas), add a child or child branch without changing the number of levels in the hierarchy and without crossing a power of 2 boundary.	Performs light outline restructure **Note:** If the number of levels in the hierarchy changes, Essbase clears all aggregate views and performs a full outline restructure. Performance impact is Very High. If the number of levels in the hierarchy does not change, but adding a child or child branch crosses a power of 2 boundary, Essbase performs a full outline restructure. Performance impact is High.	Very low

Table 40-1 (Cont.) Aggregate Storage Restructuring Levels

User-Outline Changes	Essbase-Restructure Level	Performance Impact
On nonattribute dimensions with stored level 0 members: • Add a child as the last child of a parent without crossing a power of 2 boundary (1, 2, 4, 8, 16, and so on). For example, if a parent member has three children, you may add a fourth child as the last child of the parent. • Add a child branch as the last child branch of an existing parent without crossing a power of 2 boundary and without changing the number of levels in the hierarchy. Examples: • Renames a member • Changes a formula • Changes an alias • Changes a dynamic hierarchy consolidation operator (for example, from + to -)	Performs light outline restructure.	Very low
On nonattribute dimensions with stored level 0 members: • Add a child that crosses a power of 2 boundary as the last child of a parent. For example, if a parent member has three children and you add a fourth and fifth child, the fifth child crosses the power of 2 boundary. See Example: Addition of Child Members. • For scenarios in which adding a child branch as the last child branch of an existing parent that crosses a power of 2 boundary or changing the number of levels in the hierarchy, which triggers a full outline restructure, see Example: Addition of Child Branches.	Clears aggregate views, and performs full outline restructure	Very high

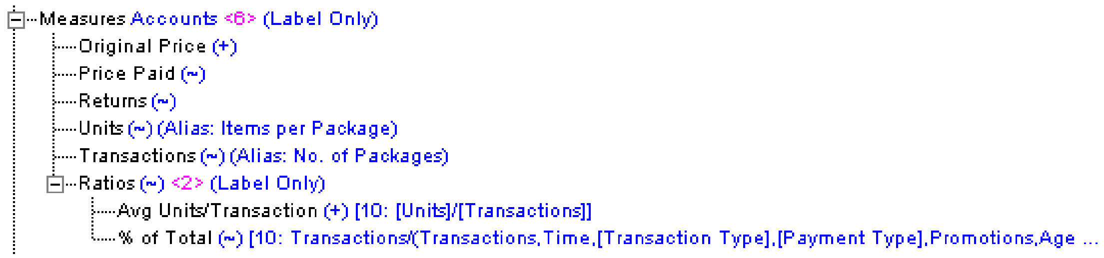
Outline-Change Examples

Examples demonstrate the restructure cost of various kinds of Essbase aggregate storage (ASO) outline changes: member additions that do or do not change the number of stored levels, changes to alternate hierarchies, adding child members, and adding child branches.

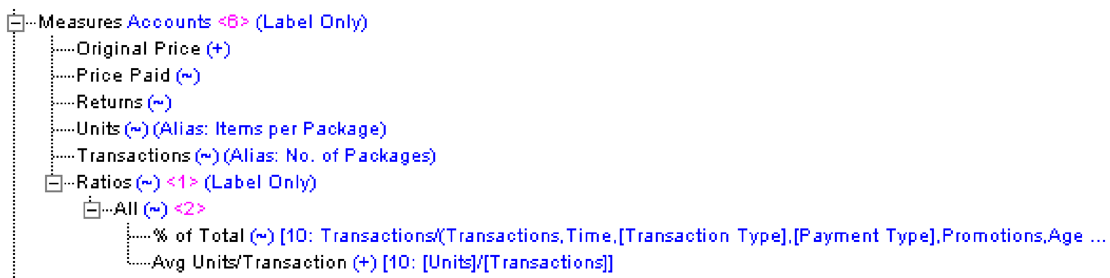
This section contains examples of the outline changes described in Levels of Aggregate Storage Cube Restructuring.

Example: No Change in the Number of Stored Levels in a Hierarchy

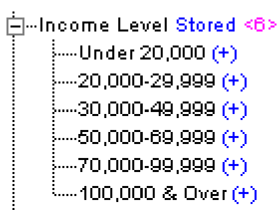
In ASOsamp.Basic, the Measures dimension is tagged as accounts. Therefore, as a dynamic hierarchy, Measures includes only one stored level.



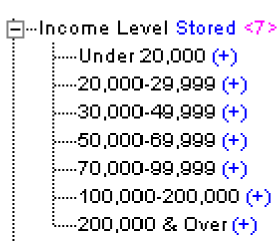
Adding the child member All to Ratios does not change the number of stored levels in the Measures dimension. Saving the outline triggers a light restructure.



In ASOsamp.Basic, Income Level is a stored hierarchy dimension.

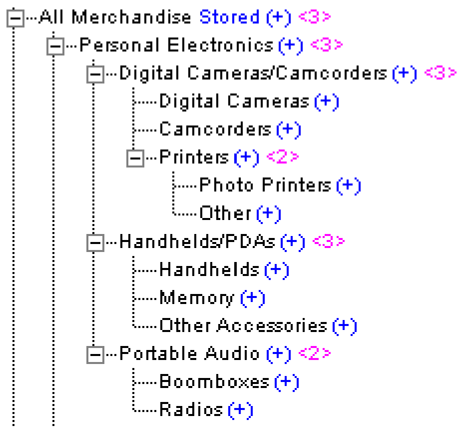


Adding a child member does not change the number of levels (two) in the hierarchy. Adding a seventh or eighth child member at the end is allowed; however, adding a ninth child member crosses the power of 2 boundary (see Example: Addition of Child Members), requiring a full restructure.



Example: Change in the Number of Stored Levels in a Hierarchy

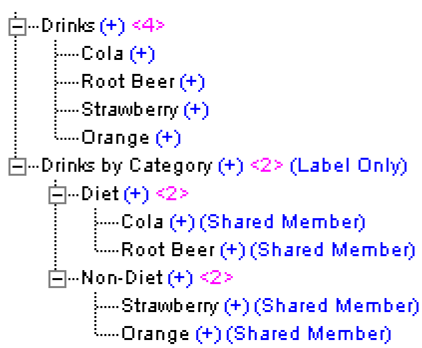
In the Product dimension in ASOsamp.Basic, renaming Photo Printers to Printers and adding child members increases the number of levels in the All Merchandise hierarchy from four to five. When the outline is saved, Essbase clears all aggregate views and performs a full restructure.



Example: Changes in Alternate Hierarchies

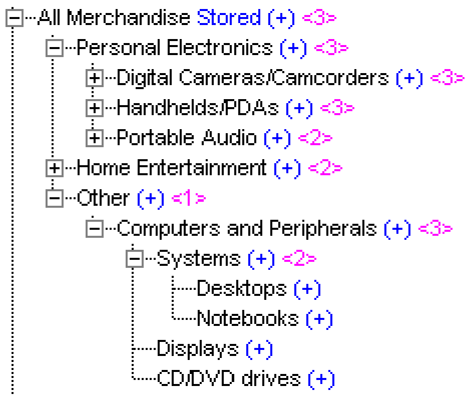
If you delete the shared member Orange under Drinks by Category and do not delete its prototype member under Drinks, the alternate hierarchy Drinks by Category is no longer a replica of the Drinks hierarchy. When the outline is saved, Essbase clears all aggregate views and performs a full restructure.

If you delete the shared and prototype Orange members, the alternate hierarchy Drinks by Category remains a replica of the Drinks hierarchy. When the outline is saved, Essbase performs a full restructure but does not clear aggregate views.

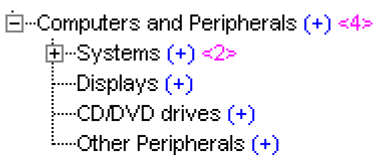


Example: Addition of Child Members

In ASOsamp.Basic, adding a child member under Systems in the All Merchandise hierarchy increases the number of children under Systems to three, crossing the power of 2 boundary. When the outline is saved, Essbase performs a full restructure.

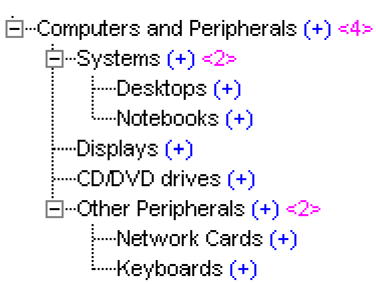


However, adding a child member under Computers and Peripherals increases the number of children under Computers and Peripherals from three to four. Adding a fourth child, which must be added after the existing members, does not cross the boundary of 2 or 4. The child must be added after existing members. When the outline is saved, Essbase performs a light restructure.



Example: Addition of Child Branches

In ASOsamp.Basic, adding a child branch under Computers and Peripherals in the All Merchandise hierarchy increases the number of children to four. Adding this child, which must be added after the existing members, does not cross the power of 2 boundary. The new member, called Other Peripherals, has two children. Systems (which is a sibling of Other Peripherals) has two children. Adding the child branch stays within the power of 2 boundary for children of sibling members at the same level. When the outline is saved, Essbase performs a light restructure.



Adding a child branch with three child members crosses the power of 2 boundary and may require that Essbase perform a full restructure. However, if Systems already had three members, the power of 2 boundary would be four and, at most, four children can be added to Other Peripherals without triggering a full restructure.

Export Aggregate Storage Cubes

If you have at least Read access to an Essbase aggregate storage (ASO) cube, you can export input-level data.

You can export level 0 data from the cube to a specified text file. The export file contains only uncompressed data. During data export, users can connect to Essbase Server and perform read-only operations on the cube.

Exported data can be reloaded without a rules file if there are no outline changes. Consider exporting data for the following reasons:

- To transfer data across platforms
- To create an exported file in text, rather than binary, format
- To create backups

The export file is created in the cube directory.

Aggregate storage exports have limits:

- You can export only level 0 data (input data).
- You cannot perform columnar exports. In a columnar export, the output file displays a member name from each dimension in every row (and names can be repeated from row to row).

To avoid creating export files larger than 2 GB, Essbase may create multiple export files that include a number suffix in the name, as follows: _1, _2, and so on. For example, if the first file name is `/home/exportfile.txt`, the next file is `/home/exportfile_1.txt`.

To improve performance, you can export data in parallel.

To export data, you can use the Essbase web interface to run the Export Data job, or you can use the export data MaxL statement.

Stored Levels in an Aggregate Storage Outline

It is not always possible to derive disk space requirements of an Essbase aggregate storage (ASO) cube based on the number of cells. Not all dimension levels are stored. Knowing the number of stored levels can help you manage the storage requirements.

To calculate the number of stored dimension levels in an outline:

1. Determine the stored-level factor for each dimension.
2. Multiply the dimension factors together; for example, *a* * *b* * *c* * *d*, and so on.

To determine the stored-level factor for each dimension, use the following guidelines:

- On dynamic dimensions, the factor is 1.
- On stored dimensions, for an initial dimension factor, count the non-label-only levels.
  - If a stored dimension has attribute dimensions, count the levels in each attribute dimension, and add that number to the dimension factor. Then, if all level 0 members of the dimension have an attribute from a specific attribute dimension, subtract 1 for each attribute dimension in which this occurs.

- – If a stored dimension has additional stored hierarchies, count the number of non-label-only levels in each stored hierarchy, and add the count to the dimension factor. Do not count a level if it comprises only shared members.
- – Dynamic hierarchies in a stored dimension do not affect the dimension factor.

For example, multiplying the factors for the 11 dimensions in ASOsamp.Basic, 1 * 1 * 4 * 2 * 2 * 2 * 3 * 2 * 6 * 7 * 7 equals 56,448.

To view dimension level factors, use the query database MaxL statement with the keywords **list aggregate_storage runtime_info**. Example:

```
query database ASOSamp.Basic list aggregate_storage runtime_info;
```

A
Limits

Essbase limitations include character and length rules for names, paths, data types, and artifacts. Some limitations apply to data load and dimension build, and others to aggregate storage and block storage cube artifacts, dimensions and other metadata, cells, parallel operations, drill through, and various size and quantity limits.

Essbase 19c and Essbase 21c supports Unicode applications only. For Essbase 11g On-Premise instances, the term non-Unicode applications refers to non-Unicode-mode applications, and applications on Essbase Servers that are not Unicode-enabled.

- Name and Related Artifact Limits
- Data Load and Dimension Build Limits
- Aggregate Storage Database Limits
- Block Storage Database Limits
- Drill-through to Oracle Applications Limits
- Other Size or Quantity Limits

Name and Related Artifact Limits

Names, paths, and other characteristics of Essbase data types and artifacts are subject to character and length rules specified here.

The following table lists limits for names and related artifacts.

Release Essbase 19c and later supports Unicode applications only.

Table A-1 Names and Related Artifacts

Artifact	Limits
Alias name	Limit: 1024 characters See Naming Conventions for Dimensions, Members, and Aliases for more limitations.
Alias table name	Limit: 30 characters

Application names	Limit: 30 characters
	The following special characters are not permitted:
	`: ; , = + * ? [ ] |< > \ " ' /`
	`[Space] [Tab] [Ascii values <32]`
	`[Unicode characters]`

Application description	Limit: 80 characters

Table A-1 (Cont.) Names and Related Artifacts

<table>
<tr><th>Artifact</th><th>Limits</th></tr>
<tr><td>• Custom-defined function name
• Custom-defined macro name
• Custom-defined function specification
• Custom-defined macro specification</td><td>Limit: 128 characters.
MaxL and the API truncate characters after 128 characters.
No truncation on server. No error is displayed if truncation occurs.</td></tr>
<tr><td>Custom-defined function and macro comment</td><td>Limit: 256 characters.
MaxL and the API truncate characters after 256 characters.
No truncation on server. No error is displayed if truncation occurs.</td></tr>
<tr><td>Database (cube) names</td><td>Limit: 30 characters
The following special characters are not permitted:
<code>: ; , = + * ? [ ] |< > \ " ' /</code>
<code>[Space] [Tab] [Ascii values <32]</code>
<code>[Unicode characters]</code></td></tr>
<tr><td>Database description</td><td>Limit: 80 characters</td></tr>
<tr><td>Dimension name</td><td>Limit: 1024 characters
See Naming Conventions for Dimensions, Members, and Aliases for more limitations.</td></tr>
<tr><td>Directory path
For example:
<code>/scratch/user/essbase/oracle_home/</code>
<code>oracle_common/bin</code></td><td>Limit: 1024 characters</td></tr>
<tr><td>Environment variable name</td><td>Limit: 320 characters</td></tr>
<tr><td>Environment variable value</td><td>Limit: 256 characters</td></tr>
<tr><td>Essbase Server name</td><td>Limit: 1024 characters</td></tr>
<tr><td>File names</td><td>Limit: 255 on Linux and 175 on Windows, including the file extension.
The following special characters are not permitted:
<code>: ; , = + * ? [ ] |< > \ " ' /</code>
<code>[Space] [Tab] [Ascii values <32]</code>
<code>[Unicode characters]</code></td></tr>
<tr><td>Filter name</td><td>Limit: 256 characters
The following special characters are not permitted:
<code>: ; , = + * ? [ ] |< > \ " ' /</code>
<code>[Space] [Tab] [Ascii values <32]</code>
<code>[Unicode characters]</code></td></tr>
</table>

Table A-1 (Cont.) Names and Related Artifacts

Artifact	Limits

Group name	Limit: 256 characters
	The following special characters are not permitted:
	`: ; , = + * ? [ ] |< > \ " ' /`
	`[Space] [Tab] [Ascii values <32]`
	`[Unicode characters]`

LRO cell note	Limit: 600 characters
LRO URL	512 characters (always single-byte characters)
LRO description for URLs and files	Limit: 80 characters
Member comment field	Limit: 256 characters
Member name	Limit: 1024 characters See Naming Conventions for Dimensions, Members, and Aliases for more limitations.
Qualified member name limit: the number of levels that can be specified in a qualified member name	Limit: 20 levels
Password	Limit: 100 characters. Use of `$` (dollar sign) character within the Essbase password is not supported for logins in a Linux environment.
Runtime substitution variable name	Limit: 320 bytes
Runtime substitution variable value	Limit: 256 bytes
Substitution variable name	Limit: 320 bytes See also Rules for Setting Substitution Variable Names and Values
Substitution variable value	Limit: 256 bytes See also Rules for Setting Substitution Variable Names and Values
Maximum number of text values in a text list object	32,766
Maximum length of the name of a text value	80
Trigger name	Limit: 30 characters
UDA	Limit: 1024 characters

User name	Limit: 50 characters
	The following special characters are not permitted:
	`; , = + * ? [ ] |< > \ " ' / #`
	`[Space] [Tab]`

Data Load and Dimension Build Limits

The following table lists limits related to data loading and dimension building.

Table A-2 Data Load and Dimension Build Limits

Artifact	Limits
Selection and rejection criteria	Number of characters that describe selection and rejection criteria: combination of all criteria limited to 32 KB
Maximum size per unique member name in a duplicate-member enabled outline	16500 bytes
Maximum number of source files for incremental dimension build	64
Maximum number of levels in an outline hierarchy	127

Aggregate Storage Database Limits

The following limits are related to aggregate storage databases.

Table A-3 Aggregate Storage Limits

Artifact	Limits
Number of hierarchies in a single dimension of an aggregate storage database	65535 (including all stored hierarchies, dynamic hierarchies, and any attribute dimensions based on the dimension)
Number of members in an aggregate storage outline	10,000,000 to 20,000,000 members, depending on available memory and other memory requirements
Maximum file location size in an aggregate storage database	4,294,967,295 MB
Number of cells that can be queried in an aggregate storage database that has a very sparse data set	2^{64} The limit is based on the number of cells that Essbase processes in the query, which is the product of the member count across all dimensions, not on the number of cells contained in the output report. You might encounter this limit when using a client interface that suppresses missing data in order to generate a relatively small report compared to the size of the query.
Number of alias tables	56
Number of parallel export threads	8 For information about parallel export, see the export data MaxL statement.

Table A-3 (Cont.) Aggregate Storage Limits

Artifact	Limits
Number of dimensions in an aggregate storage outline	255 When working with an aggregation storage database outline, dimensions that you delete count toward the total number of dimensions because Essbase does not reclaim the ID of deleted dimensions. Therefore, if you create and then delete dimensions, the maximum number of dimensions is reduced by the number of deleted dimensions. For example, if you create 10 dimensions in your aggregate storage outline, and you then delete two of those dimensions, the maximum number of dimensions of 255 becomes 253 (255-2) for this outline.

Block Storage Database Limits

Members, cells, blocks, dimensions, alias tables, parallel threads, linked reporting objects, and other artifacts of Essbase block storage databases are subject to limitations specified here.

The following limits are related to block storage databases.

Table A-4 Block Storage Database Limits

Artifact	Limits
Number of members in a block storage outline	Approximately 1,000,000 explicitly defined members in an Essbase outline. Longer member names, which can occur if using multibyte characters, decrease the number of members that are allowed.
Number of possible cells per block in a block storage database	64-bit: 2^{28} -1 (268,435,455)

Table A-4 (Cont.) Block Storage Database Limits

Artifact	Limits
Number of possible sparse blocks in a block storage database	The product of the stored, dynamic, and never share member counts of sparse dimensions cannot exceed the following value: 64-bit: 2^{104} Blocks for members of the type store and never share are stored. Blocks for members of the type dynamic calc are dynamically calculated, and they contribute to the number of possible sparse blocks. Shared members (including implied share) and label only members do not contribute. The product of member counts is tracked using two 64-bit numbers. The product of member counts of sparse dimensions is taken in the order in which they are defined in the outline. Whenever the product exceeds 2^{52} on the first 64-bit number, there is a rollover to the second 64-bit number, where the product tracking starts again at 0 and goes up to 2^{52}. If the second number hits the 2^{52} limit, you will encounter an Essbase block limit error. To avoid this, try rearranging the sparse dimensions in a different order in the outline so as to accommodate the products of the member counts without reaching the 2^{52} limit.
Number of Linked Reporting Objects in a block storage database	64-bit: 2^{104} **Note:** When viewing or editing Linked Reporting Objects attached to cells, you may encounter a limitation prior to 2^{104}, if the outline has non-stored members. If this occurs, the following error is displayed: "Error 1140013 Invalid Cell Address Entered" You can work around the limit by reordering the dimensions in the outline.
Number of alias tables	56
Number of parallel export threads	1024 For information about parallel export, see the export data MaxL statement.

Table A-4 (Cont.) Block Storage Database Limits

Artifact	Limits
Number of dimensions in a block storage outline	254

Drill-through to Oracle Applications Limits

The following limits are related to drill-through URLs.

Table A-5 Drill-through to Oracle Applications Limits

Artifact	Limits
Number of drill-through URLs per database	255
Number of drillable regions in a drill-through URL	256
Number of characters per drillable region	65536

Other Size or Quantity Limits

Refer to this section for limits on size and quantity of various Essbase strings, filters, records, queries, arrays, and other artifacts.

Table A-6 Other Size or Quantity Limits

Artifact	Limits
Formula size	65534 bytes
Number of security filters	• Per Essbase Server: 65535 • Per Essbase cube: 32290
Number of users and groups (combined)	30,000 Exceeding this limit results in an error.
Rules file maximum record size	64 KB
Rules file SQL query maximum size	64 KB
MDX maximum query limits and set sizes	See MDX Query Limits.
Maximum size for variable arrays in calculation scripts and formulas. Arrays temporarily store arguments to calculation functions, including MEMBERS, NUMBERS, and STRINGS, or values associated with them.	Maximum integer value of 2,147,483,647
MaxL maximum statement length	81,920 characters or bytes
Maximum length of a format string	256 characters
Maximum number of data audit records you can view in the Essbase web interface and in Smart View.	5000. More records can be fetched using the REST API. Refer to Get Audit Data.

B

Naming Conventions for Essbase

Review these naming conventions for Essbase applications, databases, dimensions, members, and aliases. Check for reserved words that require special handling if used as names.

- Naming Conventions for Applications and Databases
- Naming Conventions for Dimensions, Members, and Aliases
- Naming Conventions for Dynamic Time Series Members
- Naming Conventions for Attribute Calculations Dimension Member Names
- Naming Conventions in Calculation Scripts, Report Scripts, Formulas, Filters, and Substitution and Environment Variable Values
- List of Essbase System-Defined Dimension and Member Names
- MaxL Reserved Words List

Naming Conventions for Applications and Databases

When naming applications and databases, follow these rules:

- Use no more than 8 bytes when naming non-Unicode-mode applications and databases.
- Use no more than 30 characters when naming Unicode-mode applications and databases.
- Do not use spaces in the name.
- Do not use the characters listed in the name:

Table B-1 List of Restricted Characters in Application and Database Names

Character	Description
*	asterisk
[]	brackets
:	colon
;	semicolon
,	comma
=	equal sign
>	greater-than sign
<	less-than sign
.	period
+	plus sign
?	question mark
"	double quotation mark
‘	single quotation mark
`	back apostrophe

Table B-1 (Cont.) List of Restricted Characters in Application and Database Names

Character	Description
/	forward slash
\	backslash
|	vertical bars
[TAB]	tabs

- For database names, do not use the:
  - String drxxxxxx (not case-sensitive)
  - Reserved word Replay
- For aggregate storage databases, do not use the following words as application or database names:

```
DEFAULT
LOG
METADATA
REPLAY
TEMP
```

Application and database names are not case-sensitive. However, on case-sensitive file systems, the application or database name is created exactly as you enter it. Therefore, when creating, renaming, or copying applications and databases on case-sensitive file systems, Essbase ensures that the same application or database name but with different case usage cannot be used. For example, if you create an application name with all uppercase letters (NEWAPP), you cannot then create an application with the same name but with mixed-case letters (Newapp). Also, when manually copying application and database files from one computer to another and then creating an application or database, you must use the same case for the application and database directory names on both computers.

Naming Conventions for Dimensions, Members, and Aliases

When naming dimensions, members, and aliases in the database outline, follow these rules:

- Use no more than 1024 bytes when naming non-Unicode-mode dimensions, members, or aliases.
- Use no more than 1024 characters when naming Unicode-mode dimensions, members, or aliases.
- Names are not case-sensitive unless case-sensitivity is enabled.
- Even when case-sensitivity is enabled, in an aggregate storage outline for which duplicate member names is enabled, do not use matching names with only case differences for a dimension name. For example, do not name two dimensions Product and product.
- Duplicate member names or aliases are not allowed as siblings in a dimension.
- Do not use quotation marks (" "), brackets ([ ]), newlines, or tabs in dimension, member, or alias names.
- Do not use the following characters to begin dimension or member names:

Character	Description
@	at sign
\	backslash
/	forward slash
{ }	brace
,	comma
-	dash, hyphen, or minus
=	equal sign
<	less than sign
( )	parentheses
.	period
+	plus sign
'	single quotation mark
_	underscore
|	vertical bar

- Do not place spaces at the beginning or end of names, as they are ignored.

Do not use these words as dimension, member, or alias names:

- Calculation script commands, operators, and keywords
- Function names and function arguments
- Names of other dimensions and members (unless the member is shared), and generation names, level names, and aliases in the database
- @_NULL, $$$UNIVERSE$$$, and #MISSING
- Words reserved for Sandbox dimensions, including "Base" or any name beginning with "sb"

The following list of words are not strictly prohibited; however, Oracle recommends that you avoid using these words unless you are certain that doing so will not cause unanticipated problems:

```
ALL
AND
ASSIGN
AVERAGE
CALC
CALCMBR
COPYFORWARD
CROSSDIM
CURMBRNAME
DIM
DIMNAME
DIV
DYNAMIC
EMPTYPARM
EQ
EQOP
EXCEPT
EXP
EXPERROR
FLOAT
```

```
FUNCTION
GE
GEN
GENRANGE
GROUP
GT
ID
IDERROR
INTEGER
LE
LEVELRANGE
LOOPBLOCK
LOOPPARMS
LT
MBR
MBRNAME
MBRONLY
MINUS
MISSING
MUL
MULOP
NE
NON
NONINPUT
NOT
OR
PAREN
PARENPARM
PERCENT
PLUS
RELOP
SET
SKIPBOTH
SKIPMISSING
SKIPNONE
SKIPZERO
TO
TOLOCALRATE
TRAILMISSING
TRAILSUM
UMINUS
UPPER
VARORXMBR
XMBRONLY
#MI
```

Naming Conventions for Dynamic Time Series Members

If you enable Dynamic Time Series members, do not use the following associated generation names:

- History
- Year
- Season

- Period
- Quarter
- Month
- Week
- Day

Naming Conventions for Attribute Calculations Dimension Member Names

In unique member outlines that contain an attribute dimension (and, therefore, an Attribute Calculations dimension), do not use the following names unless you change the default names in the Attribute Calculations dimension:

- Sum
- Count
- Min
- Max
- Avg

If the outline is tagged as a duplicate member outline, you can use the default names to name other base or attribute members.

See Changing the Member Names of the Attribute Calculations Dimension.

Naming Conventions in Calculation Scripts, Report Scripts, Formulas, Filters, and Substitution and Environment Variable Values

When Essbase member names contain special characters, and you cite these member names in variable values, scripts, filters, partitions, or formulas, you need to use the appropriate enclosures to quote the member names. Which enclosure you need depends on whether the database is block storage or aggregate storage.

In substitution variable values, environment variable values, calculation scripts, report scripts, filter definitions, partition definitions, or formulas, you must enclose member names in brackets ([ ]) when used within MDX statements (or aggregate storage outline formulas), and otherwise in quotation marks (" "), in these situations:

- Name starts with one or more numerals (for example, 100).
- Name contains spaces or any characters listed in the table below:

Table B-2 Characters that Require Member Name Enclosures

Character	Description
&	ampersand
*	asterisk
@	at sign

Table B-2 (Cont.) Characters that Require Member Name Enclosures

Character	Description
\	backslash
{ }	braces
:	colon
,	comma
-	dash, hyphen, or minus
!	exclamation point
=	equal sign
>	greater than sign
<	less than sign
()	parentheses
%	percent sign
.	period
+	plus sign
;	semicolon
/	slash
~	tilde

In calculation scripts and formulas, you must enclose these member names, which are also Essbase keywords, in quotation marks (" ") for block storage databases, and in brackets ([ ]) for aggregate storage databases:

```
BEGIN
DOUBLE
ELSE
END
FUNCTION
GLOBAL
IF
MACRO
MEMBER
RANGE
RETURN
STRING
THEN
```

List of Essbase System-Defined Dimension and Member Names

When using attribute dimensions, Essbase creates the following dimension and members names:

Member names in attribute dimensions:

- True
- False

Dimension name: Attribute Calculations

Member names in the Attribute Calculations dimension:

- Sum
- Count
- Min
- Max
- Avg

See Naming Conventions for Attribute Calculations Dimension Member Names.

MaxL Reserved Words List

The following keywords are part of the MaxL grammar, and are reserved. If you intend to use any of these words as names or passwords, you must enclose the word in single quotation marks.

```
abort
absolute_value
account_type
active
add
administrator
advanced
after
aggregate
aggregates
aggregate_assume_equal
aggregate_missing
aggregate_storage
aggregate_sum
aggregate_view
aggregate_use_last
algorithm
alias
alias_names
alias_table
all
all_users_groups
allocation
alloc_rule
allow
allow_merge
alter
alternate_rollups
amount
amountcontext
amounttimespan
any
append
application
application_access_type
apply
archive
```

```
archive_file
area
as
aso_level_info
at
attribute
attribute_calc
attribute_info
attribute_spec
attribute_to_base_member_association
auto_password
autostartup
b
backup_file
based
basis
basistimespan
basistimespanoptions
before
begin
bitmap
blocks
buffer_id
buffered
build
by
cache_pinning
cache_size
calc_formula
calc_script
calc_string
calculation
cascade
cell_status
change_file
clear
client
cnt_sempaphore
column_width
columns
combinebasis
commands
comment
commitblock
committed_mode
compact
compression
compression_info
config_values
connect
connects
consolidation
copy
copy_subvar
copy_useraccess
create
```

```
create_application
create_blocks
create_user
creation
creation_user
creditmember
cube_size_info
currency
currency_category
currency_conversion
currency_database
currency_member
currency_rate
custom
data
data_block
data_cache_size
data_file
data_file_cache_size
data_storage
data_string
database
database_synch
database_asynch
days
dbstats
debitmember
debug
default
definition_only
definitions
delete
designer
destroy
dimension
dimensions
direct
direction
directory
disable
disabled
disallow
discard_errors
disk
display
divideamount
division
drillthrough
dml_output
drop
dump
dynamic_calc
eas_loc
enable
enabled
encrypted
```

```
end
end_transaction
enforce
eqd
error
error_file
errors_to_highest
errors_to_location
errors_to_lowest
estimated
event
exact
excel
exceeds
excludedrange
execute
existing_views
export
export_directory
external
failed_sss_migration
fragmentation_percent
freespace
from
file
file_location
file_size
file_type
filter
filter_access
fixed_decimal
for
force
force_dump
formatted_value
function
gb
get
get_missing_cells
get_meaningless_cells
global
grant
group
group_id
ha_trace
held
high
hostname
identified
identify
ignore_missing_values
ignore_zero_values
immediate
implicit_commit
import
in
```

```
inactive
inactive_user_days
including
incremental
index
index_cache_size
index_data
index_page_size
information
initialize
input
instead
invalid_block_headers
invalid_login_limit
io_access_mode
kb
kernel_io
kernel_cache
kill
level
level0
license_info
linked
list
load
load_buffer
load_buffers
load_buffer_block
local
location
lock
lock_timeout
locked
log_level
logfile
login
logout
long
lotus_2
lotus_3
lotus_4
low
lro
macro
manager
mapped
max_disk_size
max_file_size
max_lro_file_size
mb
medium
member
member_alias_namespace
member_calculation
member_comment
member_data
```

member_fixed_length_data
member_formula
member_info
member_name_namespace
member_property
member_uda
member_uda_namespace
member_variable_length_data
merge
meta_read
metadata_only
migr_modified_access
miner
minimum
mining
minutes
missing_value
mode
model
move
multiple
multiplication
mutex
name
negativebasisoptions
never
no_access
none
non_unique_members
nonunicode_mode
note
nothing
numerical_display
object
objects
of
off
offset
on
only
opg_cache
opg_state
optional
optional_group
options
or
outline
outline_id
outline_paging_file
output
override
overview
partition
partition_file
partition_size
passive

password
password_reset_days
performance
permission
persistence
perspective
physical
pmml_file
ports
pov
pre_image_access
precision
preserve
preserve_groups
private
privilege
process
project
property
protocol
purge
query
query_data
query_tracking
range
read
recover
reference_cube
reference_cube_reg
refresh
region
registration
reregister
remote
remove
remove_zero_cells
rename
repair
repeatamount
replace
replay
replicated
replication_assume_identical
report_file
request
request_history
request_id
reset
resource_usage
restore
restructure
result
resync
retrieve_buffer_size
retrieve_sort_buffer_size
reverse

```
revoke
rle
round
row
rows
rules_file
runtime
runtime_info
save
scientific_notation
scope
score
script_file
seconds
security
security_backup
select
selecting
selection
self_session_info
semaphore
sequence_id_range
server
server_port
session
session_idle_limit
session_idle_poll
set
shared_services_native
short
shutdown
single
singlecell
size
size_limit
skip_to_next_amount
skip_missing
skip_negative
skip_zero
slice
sourceregion
spec
spinlock
splitbasis
spread
SSL
sss
sss_mode
sss_name
starting
startup
statistics
status
stop
stopping
storage
```

storage_info
structure_file
subtract
supervisor
suppress
sync
system
table
tablespace
target
targettimespan
targettimespanoptions
task
tb
template
text
thread
to
total_size
transactions
transformation
transparent
trigger
trigger_func
trigger_spool
two_pass_calc
type
uda
unicode
unicode_mode
unlimited
unload
unlock
update
updated
updates
use
user
username_as_password
using
validate
values
variable
vector
verification
version
view_file
views
volume
wait_for_resources
warn
when
with
wizard
worksheet
write

```
xml_file
zero_value
zeroamountoptions
zerobasisoptions
```