

DB2 Alphablox for UNIX and Windows v5.6

Alphablox Cube Server Administrator's Guide



Note: Before using this information and the product it supports, read the information in “Notices” on page 7.

First edition (August 2004)

This edition applies to version 5, release 6, of IBM DB2 Alphablox for UNIX and Windows V5.6 (product number 5724-J16) and to all subsequent releases and modifications until otherwise indicated in new editions.

Copyright © 1996 - 2004 Alphablox Corporation. All rights reserved.

© Copyright International Business Machines Corporation 1996, 2004. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Alphablox Cube Server Administrator's Guide

Notices	7
Trademarks	9
Preface	11
About This Book	12
Related Documents	13
Online Documentation User Interface	14
Document Conventions	15
Icons	15
Typography	15
Contacting IBM	16
Product Information	16
Comments on the Documentation	17
Chapter 1	
Cubing Concepts	19
Overview	20
Cubing Relational Data	21
Applications of the Alphablox Cube Server	21
Relatively Small Cube Data Sets From Potentially Very Large RDBMSs	
22	
Prototyping	22
Cubes With Straightforward Dimensions and Measures	22
Advantages of the Alphablox Cube Server	23
Alphablox Cube Server in an Alphablox Analytics Application Environ-	
ment	23
Alphablox Cube Server Architecture	24
Alphablox Cube Server Components	24
Administration User Interface	25
Cube Manager	25
In-Memory Cache	25
Compiler	25
Executor	25

MDX to SQL Query Translation	26
Schema Requirements	27
Clean Data	27
Dimensional Schema	27

Chapter 2

Dimensional Schema Design 29

Dimensional Schemas	30
Star and Snowflake Schemas	31
Primary Keys	31
Foreign Keys	31
Fact Tables	31
Dimension Tables	31
Star Schemas	32
Snowflake Schemas	33
Hierarchies	33
Many-to-One Relationships	34
Normalized Versus Denormalized	34
Mapping the Relational Schema to a Cube	36
Dimensions and Levels	36
Measures	37

Chapter 3

Creating and Modifying a Cube39

Checklist of Tasks to Create a Cube	40
Create the Relational Data Source	42
Define the Cube	44
Define the Measures	45
Define the Dimensions and Levels	46
Create or Edit the Dimension	46
Create and Edit the Levels	47
Edit the SQL (Advanced Users Only)	49
Create Cube Adapter Data Source	50
Specify and Manage Cube Resources	51
Defining a Refresh Schedule	51
Setting Tuning Parameters	52
Sanity Check the Cube	53

Chapter 4

Maintaining a Cube55

Starting, Stopping, and Rebuilding a Cube	56
Starting an Alphablox Cube	56
Start Cube From the Home Page	56

Start Cube From a Console Window	56
Troubleshooting If the Cube Does Not Start	57
Stopping an Alphablox Cube	58
Stop Cube From the Home Page	58
Stop Cube From a Console Window	58
Rebuilding an Alphablox Cube	59
Deciding on an Administration Strategy	60
Understanding the Database Environment	60
Scheduling Periodic Updates	61
Console Commands	62
Modifying a Cube	64
Tuning a Cube	65
Tuning Controls	65
Connection and Cache Size Limits	65
Maximum Number of Cubes	67
Maximum Rows and Columns	67
Alphablox Cube Memory Considerations	68
Changing the Maximum Memory Heap Size	68
Setting the Maximum Memory Heap Size During Installation	69
Setting the Maximum Memory Heap Size By Modifying the Initializa- tion File	69
Adding More Memory to Your System	71
Chapter 5	
Using MDX to Query Alphablox Cubes	73
Supported MDX Syntax	74
Basic Syntax	74
Specifying Member Sets	75
Qualified Member Names	75
Curly Braces	75
FROM:TO Syntax	75
Functions	76
CHILDREN	76
CROSSJOIN	77
CURRENTMEMBER	78
DESCENDANTS	78
GENERATE	80
MEMBERS	81
Example MDX Queries	83
Example 1	83
Example 2	84
Index	85



Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation, Licensing, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licenses of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation, J46A/G4, 555 Bailey Avenue, San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

Alphablox, InLine Analytics, Alphablox Analysis Server, Blox, SpreadsheetBlox, and the Alphablox logo are trademarks or registered trademarks of Alphablox Corporation.

IBM, DB2, DB2 Universal Database, WebSphere, and DB2 OLAP Server are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Preface

This Preface describes the intended audience, organization, and conventions used in the *Alphablox Cube Server Administrator's Guide*. It also contains information about the DB2 Alphablox documentation set and information about how to contact IBM for technical problems or comments on the documentation.

Contents

- “About This Book” on page 12
- “Related Documents” on page 13
- “Online Documentation User Interface” on page 14
- “Document Conventions” on page 15
- “Contacting IBM” on page 16

About This Book

The *Alphablox Cube Server Administrator's Guide* contains information about using the Alphablox Cube Server to create and manage Alphablox cubes.

The *Alphablox Cube Server Administrator's Guide* is primarily designed for the individuals responsible for defining and managing Alphablox cubes. Administrators who create Alphablox cubes should have knowledge of database administration and star schema design.

The *Alphablox Cube Server Administrator's Guide* is organized into the following chapters:

- Chapter 1, “Cubing Concepts” on page 19
- Chapter 2, “Dimensional Schema Design” on page 29
- Chapter 3, “Creating and Modifying a Cube” on page 39
- Chapter 4, “Maintaining a Cube” on page 55
- Chapter 5, “Using MDX to Query Alphablox Cubes” on page 73

Related Documents

The DB2 Alphablox documentation set includes books and online help. The books are all available in HTML, PDF, and printed format. Context sensitive help is available for all parts of the Alphablox Analytics home page as well as within Alphablox Analytics applications. The DB2 Alphablox documentation set includes the following books:

Title	Description
<i>Administrator's Guide</i>	Contains information about setting up and managing DB2 Alphablox and about DB2 Alphablox in a J2EE environment.
<i>Developer's Guide for the DHTML Client</i>	Provides guidance on designing, developing, and deploying analytical applications using the DHTML client. If you are new to DB2 Alphablox or are developing new applications, it is recommended that you start with this book.
<i>Developer's Reference for the DHTML Client</i>	A complete API reference for developing applications using the DHTML client; contains information on each Blox, including its JSP syntax, properties, methods, and objects.
<i>Relational Reporting Developer's Guide</i>	Contains information about setting up ReportBlox to build a report from relational data.
<i>Alphablox Cube Server Administrator's Guide</i>	Contains information about setting up Alphablox cubes. Alphablox cubes allow you to present a multidimensional view of data stored in a relational data warehouse or data mart database.
<i>Installation Guide</i>	Contains information on system requirements, installing and configuring DB2 Alphablox, installing sample data, and migrating application from previous versions.

Javadoc is available for the server-side API, the ReportBlox API, and FastForward API, and can be from the following directory:

```
<alphablox_dir>/analytics/system/documentation/javadoc
```

where `<alphablox_dir>/analytics` is the directory in which DB2 Alphablox is installed.

Online Documentation User Interface

The DB2 Alphablox documentation is also available online in HTML and PDF formats. To open the Online Documentation, select the **Online Documentation** link on the **Help** menu or from any help page on the Alphablox Analytics home page.

When you select the Online Documentation, it opens in a frameset. The right frame displays documentation pages; the left frame contains the following navigation tabs:




Tab	Description
Contents	<p>The Contents tab presents a tree view of all the online books in the documentation set. Click on a book icon beside a heading to expand or collapse the tree, displaying or hiding the topics within that heading. To view a topic, click on its hyperlinked heading.</p> <p>To access a page containing links to all of the PDF versions of the documentation, click the PDF Documentation book icon and then click the PDF Documentation hyperlink.</p>
Index	<p>The Index tab presents an alphabetical list of all indexed words for every document in the Alphablox Analytics documentation set. To view a topic, click on the indexed item. If multiple pages are available for a topic, click the link with the page title for the first topic, click the link with the number 2 for the second topic, and so on.</p>
Search	<p>The Search tab provides a text search.</p> <p>The search feature provides a simple search on words entered. You can search a single book instead of the entire documentation set by selecting a book from the dropdown list. The search supports the use of asterisks (*) for wildcard searches, but does not use “near” logic or perform partial word search. Entering multiple words implies an and between the words, returning pages that contain all the words entered. The search is not case sensitive.</p> <p>To enter a search, click the Search tab, type one or more words in the search box, and press the Search button. The search presents a list of HTML pages containing the search word(s).</p> <p>To view a page, click on its hyperlinked heading. When the page appears in the right frame, use its hyperlinks or the browser’s Find command to locate the word(s) within the page.</p>

Document Conventions

Icons and typography call attention to or elaborate on areas of interest throughout the DB2 Alphablox documentation set.

Icons

The icons used in the documentation are as follows:

Icons	Description
	Identifies information helpful for the current task.
	Identifies conceptual information on a particular topic or suggestions for usage.
	Identifies important information that the audience should know before proceeding with a task.

Typography

The typography used in the documentation is as follows:

Convention	Description
Bold	Caution statements, labels, headings, and table headers appear in a bold font.
<i>Italics</i>	Italics indicate an emphasized word or phrase as well as book titles.
Monospace type	Code examples, filenames, object names, property names, and method names appear monospace type.
“Quotation marks”	The proper syntax for Blox properties and methods or queries may require single or double quotation marks. In addition, quotation marks surround a cross-reference to another topic.

Contacting IBM

If you have a technical problem, please review and carry out the actions suggested by the product documentation before contacting DB2 Alphablox Customer Support. This guide suggests information that you can gather to help DB2 Alphablox Customer Support to serve you better.

For information or to order any products, contact an IBM representative at a local branch office or contact any authorized IBM software remarketer. If you live in the U.S.A., you can call one of the following numbers:

- 1-800-IBM-SERV for customer support
- 1-888-426-4343 to learn about available service options

Product Information

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

<http://www.ibm.com/software/data/db2/alphablox>

Provides links to information about DB2 Alphablox.

<http://www.ibm.com/software/data/db2/udb>

The DB2 Universal Database Web pages provide current information about news, product descriptions, education schedules, and more.

<http://www.elink.ibm.com/>

Click Publications to open the International Publications ordering Web site that provides information about how to order books.

<http://www.ibm.com/education/certify/>

The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products.

Note: In some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

Comments on the Documentation

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 Alphablox documentation. You can use any of the following methods to provide comments:

- Send your comments using the online readers' comment form at www.ibm.com/software/data/rcf.
- Send your comments by electronic mail (e-mail) to comments@us.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Cubing Concepts

Alphablox Analytics includes the Alphablox Cube Server. The Alphablox Cube Server is designed to provide a multidimensional view of data stored in a relational database. This chapter introduces the Alphablox Cube Server, provides background into the types of applications it is designed for, and describes the requirements for its use.

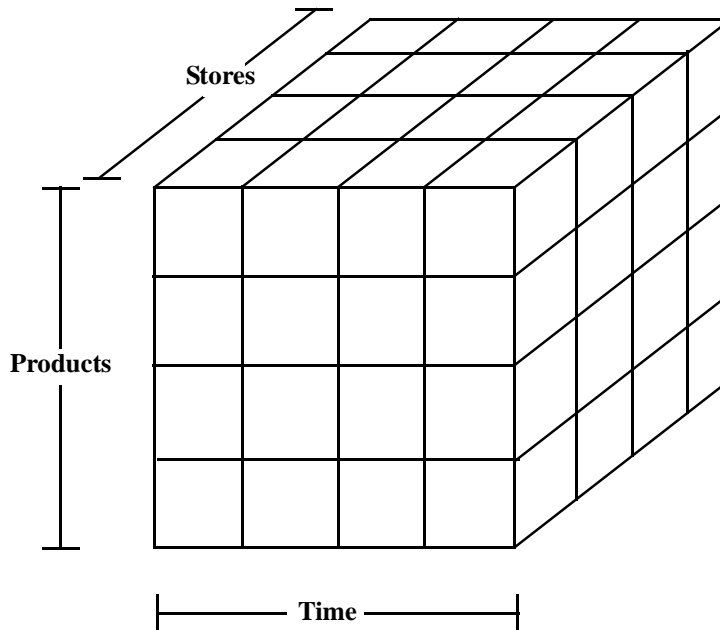
Contents

- “Overview” on page 20
- “Alphablox Cube Server Architecture” on page 24
- “Schema Requirements” on page 27

Overview

Alphablox Cube Server allows administrators to create a multidimensional representation of data that resides in a relational database. A *cube* is a data model often used in online analytical processing (OLAP) to represent business data that is typically analyzed over multiple dimensions. A *dimension* is a conceptual axis over which a business is analyzed. For example, a retail business's performance might be analyzed over time, products, and stores. For this business, *time*, *products*, and *stores* are each dimensions. Each of the dimensions has one or more *levels* which together define the overall hierarchy of the dimension. For example, the *time* dimension might have levels *year*, *quarter*, and *month*.

A cube is used to model the business. A 3-dimensional cube is easy to visualize because it can be drawn as a geometric cube, but a cube can have any number of dimensions, from one to n .



At the intersection points of a cube's dimensions, analysts can view the measures. *Measures* are numeric values, usually business metrics (such as sales, profit and cost of goods), at a given set of dimension intersections. For example, to view the sales of a given product at a given store at a given time, examine the cube at the point where those dimensions intersect to find the measures.

Cubing Relational Data

Many organizations have invested in Data Marts and Data Warehouses to store their relational data in a queryable form. This data is typically moved, cleansed, and transformed from the transactional systems where the data originated into another relational database that is optimized for query performance.

These transformed databases contain historical information on one or more subjects, and are sometimes known as data warehouses or data marts. Some common data mart and data warehouse databases are IBM DB2 Universal Database, Oracle 8i, Microsoft SQL Server and Sybase. Whatever the systems are called and in whatever relational database management system (RDBMS) they are stored, the primary purpose of these databases is to allow users to query historical information. For details on the schema designs typical of these data warehouse and data mart databases, see “Dimensional Schema Design” on page 29.

Querying relational databases can be made easier for end users if a dimensional model is used, because dimensional models make it easier to pose business questions related to a particular business process or business area. Depending upon the size and complexity of the dimensional model and the business requirements, your users may need the power of a dedicated OLAP server such as IBM DB2 OLAP Server. In these cases, you extract the data from the relational database and build dedicated high speed cubes that provide advanced analytical functions. In those cases where you do not need the full power of a dedicated OLAP Server, but you do want to provide your users with the power of OLAP, you can use the Cube capability of DB2 Alphablox.

With DB2 Alphablox, an administrator can build an Alphablox Cube on top of relational data; that is, the Alphablox Cube is populated using queries to the underlying RDBMS.

Applications of the Alphablox Cube Server

The Alphablox Cube Server allows you to quickly present relational data in the form of an OLAP cube. It provides an intelligent subset of the functionality of a full-featured OLAP server such as DB2 OLAP Server, Hyperion Essbase or Microsoft OLAP Services. An Alphablox Cube is designed to take advantage of clean data that resides in data warehouse and data marts; it is not intended as a replacement for a full-featured OLAP server. It is useful for prototyping multidimensional applications before going through the time and expense of developing full-featured OLAP databases, and it is very good at presenting relatively small cubes, even if they are built from very large databases.

Relatively Small Cube Data Sets From Potentially Very Large RDBMSs

The Alphablox Cube Server is well-suited to building cubes that return relatively small data sets compared to the underlying databases from which they are populated. The underlying databases can be very large—potentially billions of rows in the fact table (for a definition of a fact table, see “Fact Tables” on page 31). Alphablox cubes store precomputed results in memory, not on disk. Any results not stored in memory remain in the underlying database; the cube retrieves results on an as-needed basis by sending SQL queries to the database. The results from the query are then stored in memory and are instantly accessible to Alphablox Analytics applications.

Prototyping

An Alphablox cube can be created very rapidly and can therefore allow applications to access and gain value out of real data very quickly. Alphablox Analytics applications that access Alphablox cubes can be easily modified to access data that resides in a DB2 OLAP Server cube, a Hyperion Essbase cube or in a Microsoft Analysis Services cube. Therefore, Alphablox cubes provides an excellent platform for prototyping larger scale applications during a development cycle. In some cases, keeping the data in Alphablox cubes might prove completely sufficient for the needs of the application. Other times, the features and scalability of the full-featured products might be beneficial.

Cubes With Straightforward Dimensions and Measures

An Alphablox cube can have a single hierarchy per dimension. To represent complex dimensions with multiple hierarchies, use a full-featured OLAP server such as DB2 OLAP Server, Hyperion Essbase or Microsoft Analysis Services. However, many complex business scenarios do not require multiple hierarchies per dimension.



If your application requires multiple hierarchies in a single dimension, you can create multiple dimensions having the same root level but different hierarchies.

Measures in an Alphablox cube are defined with a valid SQL expression to the underlying database. However, in order to prevent problems with ambiguity, which happens when there are different tables with columns having the same name, there are a few restrictions on the SQL expression specified. See “Measures” on page 37 for more detail.

Different RDBMS vendors support different levels of calculations, but all of the major RDBMS vendors support a fairly rich set of calculations. If an application requires calculations that cannot be expressed in SQL, consider using a full-featured OLAP server.

Advantages of the Alphablox Cube Server

Because the Alphablox Cube Server is part of Alphablox Analytics, and because it has no physical disk storage to manage, many administrative tasks typical of full-featured OLAP servers are simplified or eliminated. The following are some of the advantages:

- Alphablox Cube Server has no disk space to manage.
- Alphablox Cube Server uses the Alphablox Analytics security model, requiring no additional work to manage users.
- Alphablox Cube Server is included with Alphablox Analytics, requiring no additional software to install.

Alphablox Cube Server in an Alphablox Analytics Application Environment

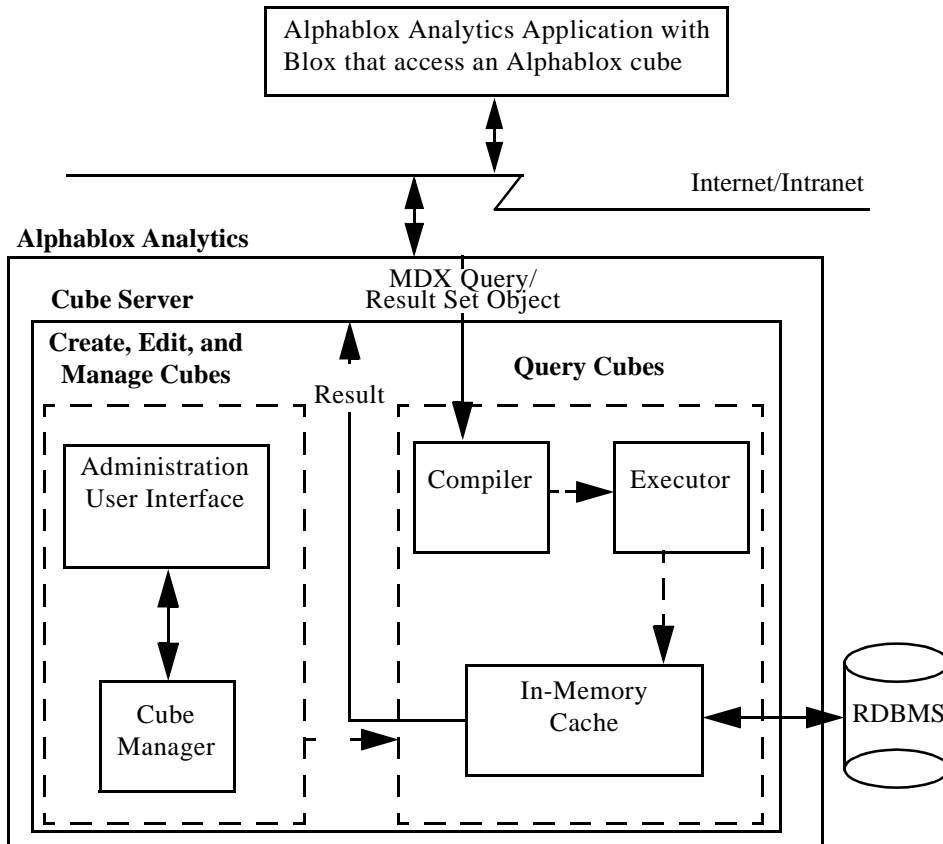
For an Alphablox Analytics application, an Alphablox cube is just another data source; that is, the Blox functionality works with an Alphablox cube just like it does with any other data source. Alphablox cubes use the same Blox as other data sources. For example, you can change an application that accesses an Alphablox cube to access an DB2 OLAP Server cube by simply changing the values of the query and data source DataBlox parameters. The application performs the same way; it is simply accessing different data. The rich Blox functionality available for manipulating other multidimensional and relational data sources are also available for Alphablox cubes.

Alphablox Cube Server Architecture

The Alphablox Cube Server is a high performance, scalable cubing engine designed to support many users querying many different cubes. It is designed to enable quick multidimensional access to relational data stored in a data warehouse or data mart database.

Alphablox Cube Server Components

The Alphablox Cube Server is composed of several components. These complementary components provide the infrastructure to define, manage, and execute queries against Alphablox cubes. The components of the Alphablox Cube Server work within the framework of Alphablox Analytics, as shown in the following figure.



Administration User Interface

A cube administrator performs the tasks for setting up and managing Alphablox cubes through the administration interface of Alphablox Analytics. The **Alphablox Cubes** link under the **Administration** tab of the Alphablox Analytics home page is devoted to cube setup and administration; this is where dimensions, levels, and measures for a cube are defined. An Alphablox Analytics user must be a member of the administrators group in order to create, view, or modify Alphablox cubes. For detailed information on using the Alphablox cubes administration user interface, see “Creating and Modifying a Cube” on page 39 and “Maintaining a Cube” on page 55.

Cube Manager

The Cube Manager is the component that creates objects, performs verification checks, starts and stops, and performs other work on Alphablox cubes. The Alphablox Analytics console also accepts commands that are carried out by the Cube Manager. For a description of the Cube Manager console commands, see “Console Commands” on page 62.

In-Memory Cache

The Cube Server stores calculated results in a cache that resides in memory. These stored results are then shared among all users accessing the Alphablox cube. Internally, each cube is broken down into smaller sections of results. Each of these sections is potentially stored in the cube’s in-memory cache. Depending on how much memory the cube results require and how much memory is available to the cube, some entries might need to be removed from the cache. If memory needs to be freed, entries are purged from the cache. The cache is populated with queries made to the underlying relational database. If a query against an Alphablox cube requests data that is not already stored in the cache, then that data is retrieved from the underlying database and, if necessary, old data is aged out of the cache. The system performs all of these caching functions automatically.

Compiler

Query requests against Alphablox cubes use the MDX query language. The Compiler parses MDX queries, validates the requests, and generates a plan to return the results to the client application. The Compiler takes advantage of metadata stored for each cube to generate an optimized plan for each request.

Executor

The Executor runs the plan generated by the Compiler and retrieves the result set from the cache. After the results are generated, they are returned to the DataBlox, GridBlox, or PresentBlox that requested them.

MDX to SQL Query Translation

Alphablox Analytics applications request results from a cube through MDX queries. Alphablox Cube Server processes the MDX query which results in a plan for retrieving results from the Alphablox cube. The Alphablox cube in turn calculates those results by running SQL queries against the underlying relational database. These SQL queries either were run before the MDX query was issued and are already stored in the cache or are executed during runtime of the MDX query. If the results are already stored in the cube's in-memory cache, then there is no need to run the SQL query again for that result set. When the Alphablox Analytics application issues the MDX query, Alphablox Cube Server automatically issues any needed SQL queries. Often, many SQL queries are needed to fulfill a single MDX request.

Schema Requirements

This section describes the requirements for the underlying database an Alphablox cube references. An Alphablox cube must reference a supported relational database. The *Installation Guide* describes the databases supported by Alphablox Analytics. The databases should have *clean data* that is stored in a *dimensional schema*.

Clean Data

The term *clean data* refers to data that follows the rules of referential integrity (whether or not referential integrity is enforced by the RDBMS). Clean data also implies that any fields in the data that might have had different values with the same meaning have been transformed to have the same values. For example, if the transaction level data has some records in which the second quarter is referred to as *Q2* and some in which it is referred to as *Quarter_2*, the records must be transformed so that there is a unique value to identify the second quarter.

Dimensional Schema

A dimensional schema in a relational database has a structure for storing clean data against which it is easy to perform historical queries. Typically, a dimensional schema can take one of the following forms:

- Single table
- Star schema
- Snowflake schema
- Combination of star and snowflake schemas

The underlying database for an Alphablox cube must contain only one fact table; multiple fact table schemas are not supported. Each dimension in an Alphablox cube must have a single hierarchy. For more information about schemas, see “Dimensional Schemas” on page 30.



If the database has multiple fact tables or does not conform to a dimensional schema, you can create views in the database to create a “virtual” single fact table dimensional schema for use with an Alphablox cube.

Dimensional Schema Design

Alphablox Cube Server requires that the underlying databases have a dimensional schema. To set up an Alphablox cube correctly, the administrator should understand the data in the underlying RDBMS. This chapter explains the concepts of dimensional schema design, defines terms such as star schema and snowflake schema, and explains the relationship between the database structure and the cube hierarchies.

Contents

- “Dimensional Schemas” on page 30
- “Mapping the Relational Schema to a Cube” on page 36

Dimensional Schemas

A database is comprised of one or more tables, and the relationships among all the tables in the database is collectively called the database *schema*. Although there are many different schema designs, databases used for querying historical data are usually set up with a *dimensional* schema design, typically a star schema or a snowflake schema. There are many historical and practical reasons for dimensional schemas, but the reason for their growth in popularity for decision support relational databases is driven by two main benefits:

- The ability to form queries that answer business questions. Typically a query calculates some measure of performance over several business dimensions.
- The necessity to form these queries in the language used by most RDBMS vendors, SQL.

A dimensional schema physically separates the measures (sometimes called facts) that quantify the business from the descriptive elements (sometimes called dimensions) that describe and categorize the business. Alphablox cubes require the underlying database to use a dimensional schema; that is, the data for the facts and the dimensions must be physically separate (at least in different columns). Typically, this is in the form of a star schema, a snowflake schema, or some hybrid of the two. While not as common a scenario, the dimensional schema can also take the form of a single table, where the facts and the dimensions are simply in separate columns of the table.



If the database does not conform to a dimensional schema, you can create views in the database to create a “virtual” dimensional schema for use with an Alphablox cube.

This section describes star and snowflake schemas and the way the business hierarchies are represented in these schemas. The following sections are included:

- Star and Snowflake Schemas
- Hierarchies

For a thorough background of dimensional schema design and all of its ramifications, read *The Data Warehouse Toolkit* by Ralph Kimball, published by John Wiley and Sons, Inc.

Star and Snowflake Schemas

Star and snowflake schema designs are mechanisms to separate facts and dimensions into separate tables. Snowflake schemas further separate the different levels of a hierarchy into separate tables. In either schema design, each table is related to another table with a *primary key/foreign key relationship*. Primary key/foreign key relationships are used in relational databases to define many-to-one relationships between tables.

Primary Keys

A *primary key* is a column or a set of columns in a table whose values uniquely identify a row in the table. A relational database is designed to enforce the uniqueness of primary keys by allowing only one row with a given primary key value in a table.

Foreign Keys

A *foreign key* is a column or a set of columns in a table whose values correspond to the values of the primary key in another table. In order to add a row with a given foreign key value, there must exist a row in the related table with the same primary key value.

The primary key/foreign key relationships between tables in a star or snowflake schema, sometimes called many-to-one relationships, represent the paths along which related tables are joined together in the RDBMS. These join paths are the basis for forming queries against historical data. For more information about many-to-one relationships, see “Many-to-One Relationships” on page 34.

Fact Tables

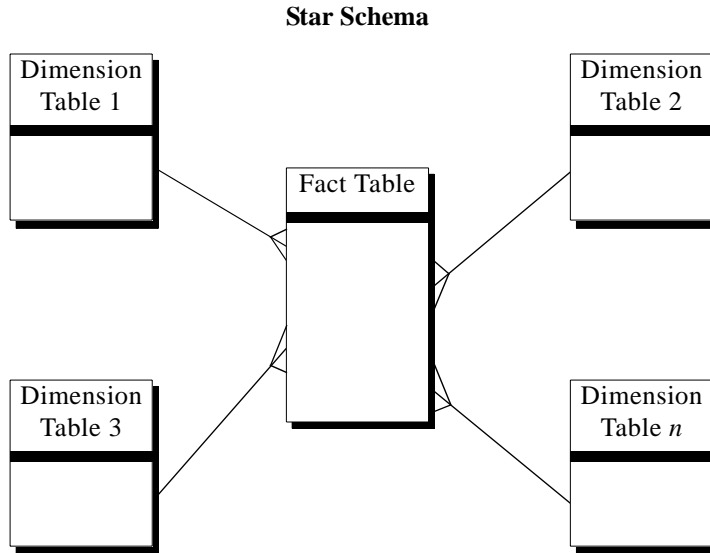
A *fact table* is a table in a star or snowflake schema that stores facts that measure the business, such as sales, cost of goods, or profit. Fact tables also contain foreign keys to the dimension tables. These foreign keys relate each row of data in the fact table to its corresponding dimensions and levels.

Dimension Tables

A *dimension table* is a table in a star or snowflake schema that stores attributes that describe aspects of a dimension. For example, a time table stores the various aspects of time such as year, quarter, month, and day. A foreign key of a fact table references the primary key in a dimension table in a many-to-one relationship.

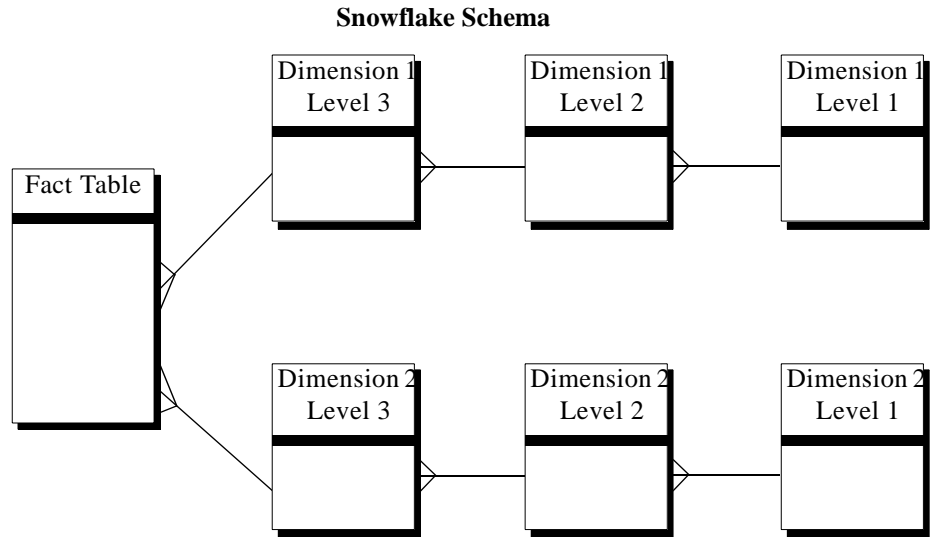
Star Schemas

The following figure shows a star schema with a single fact table and four dimension tables. A star schema can have any number of dimension tables. The crow's feet at the end of the links connecting the tables indicate a many-to-one relationship between the fact table and each dimension table.



Snowflake Schemas

The following figure shows a snowflake schema with two dimensions, each having three levels. A snowflake schema can have any number of dimensions and each dimension can have any number of levels.



For details about how the different levels of a dimension form a hierarchy, see “Hierarchies” on page 33.

Hierarchies

A hierarchy is a set of levels having many-to-one relationships between each other, and the set of levels collectively makes up a dimension. In a relational database, the different levels of a hierarchy can be stored in a single table (as in a star schema) or in separate tables (as in a snowflake schema).

Many-to-One Relationships

A many-to-one relationship is where one entity (typically a column or set of columns) contains values that refer to another entity (a column or set of columns) that has unique values. In relational databases, these many-to-one relationships are often enforced by foreign key/primary key relationships, and the relationships typically are between fact and dimension tables and between levels in a hierarchy. The relationship is often used to describe classifications or groupings. For example, in a geography schema having tables *Region*, *State* and *City*, there are many states that are in a given region, but no states are in two regions. Similarly for cities, a city is in only one state (cities that have the same name but are in more than one state must be handled slightly differently). The key point is that each city exists in exactly one state, but a state may have many cities, hence the term “many-to-one.”

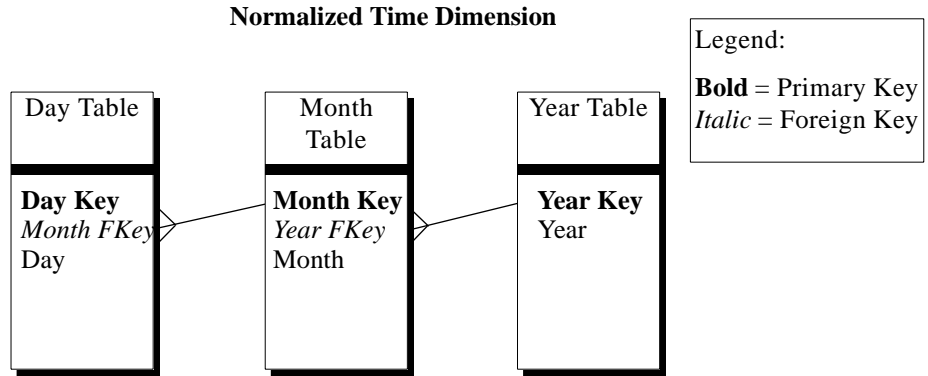
The different elements, or levels, of a hierarchy must have many-to-one relationships between children and parent levels, regardless of whether the hierarchy is physically represented in a star or snowflake schema; that is, the data must abide by these relationships. The clean data required to enforce the many-to-one relationships is an important characteristic of a dimensional schema. Furthermore, these relationships make it possible to create Alphablox cubes out of the relational data.

When you define an Alphablox cube, the many-to-one relationships that define the hierarchy become levels in a dimension. You enter this information through the administration user interface. For details about setting up the metadata to define an Alphablox cube, see “Creating and Modifying a Cube” on page 39.

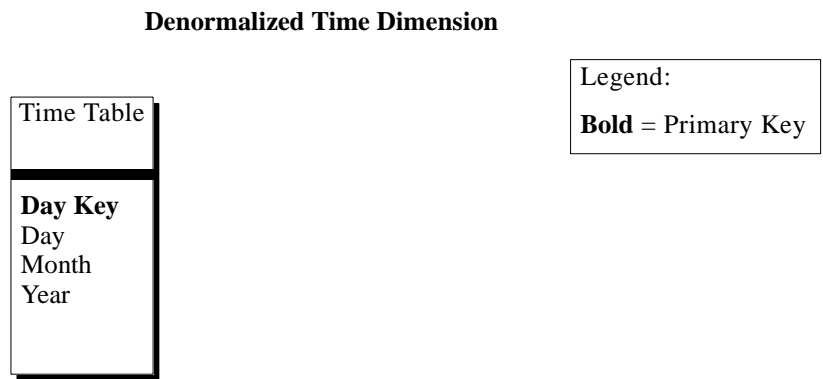
Normalized Versus Denormalized

In the relational database underlying each Alphablox cube, each dimension is either *normalized* or *denormalized*. A normalized dimension has the data for different levels stored in different tables (as in a snowflake schema). A denormalized dimension has the data for different levels stored in the same table (as in a star schema). It is important to note that in either case, the *data* still must conform to the many-to-one relationships between child and parent levels. In a normalized schema, there are primary key/foreign key relationships between tables containing parent levels and tables containing child levels, so the RDBMS ensures these relationships are true. In a denormalized schema, it is up to the database designers and administrators to make sure the data conforms to these rules; that is, the data must be clean.

The following figure shows a normalized time dimension where *Day*, *Month*, and *Year* are separate tables. The table containing the most detailed level (*Day* in this example) is referenced by the fact table; there can be many rows in the fact table that reference a given day, but there is only one row in the *Day* table for a given day.



The following figure shows a denormalized time dimension where *Day*, *Month*, and *Year* are all columns in the same table. The time table is referenced by the fact table.



When defining each level for a dimension in the administration user interface, use the check box to indicate if the level is normalized. The Cube Server needs this information to generate the correct SQL queries for populating the cube.

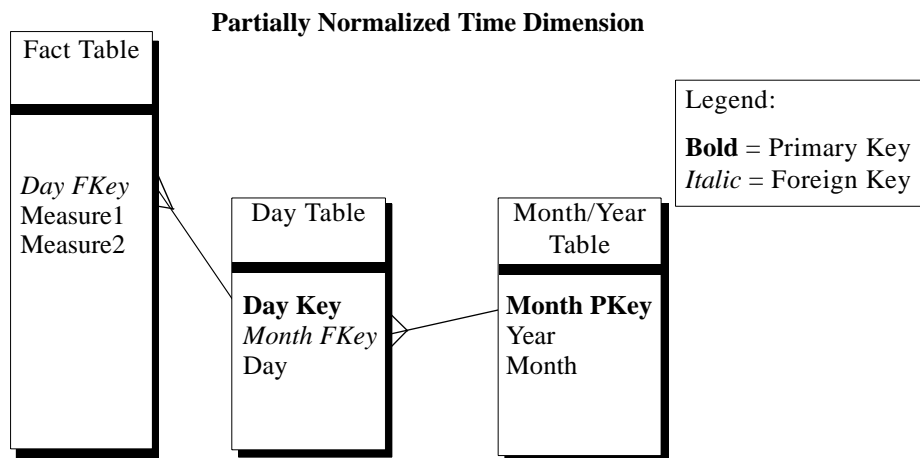
Mapping the Relational Schema to a Cube

It is important for the administrator who designs and builds an Alphablox cube to understand, at least at a high level, the mapping between the relational database and the Alphablox cube. Understanding this mapping helps to ensure there are no errors in the design or creation of the Alphablox cube. Because the cube is populated by queries to the underlying relational database, it is possible to perform quality assurance testing on the cube by comparing query results on the cube to query results on the relational database.

Dimensions and Levels


You can define any number of dimensions in an Alphablox cube and for each dimension, you can define any number of levels. In a typical snowflake schema, each level is normalized into a separate table, and the most detailed level is referenced by a foreign key from the fact table. The Alphablox Cube Server relies on the relationships among these different tables to create dimensions in the cube. When you define an Alphablox cube, you must provide details about the schema as part of the Alphablox cube definition.

The **Dimension Level Page** dialog box has a check box labeled **Normalized** to define levels that are in a separate table. When the **Normalized** box is checked, you must also enter the primary key column from the dimension table and the foreign key column from the table that references it (either another level table or the fact table). If a level is stored in a table that also contains the next level of detail, then no other table references that level, and the **Normalized** box is not checked. For example, consider the following schema:



The *Month* and *Year* levels are both stored in the *Month/Year* table. If you define the levels for an Alphablox cube that references this schema, configure the levels *Year*, *Month*, and *Day* as follows:

Level	Normalized?	Primary Key Column	Foreign Key Column
Year	No		
Month	Yes	Month PKey	Month FKey
Day	Yes	Day Key	Day FKey

 By using views in the database, it is possible for all logical tables to be stored in a single physical table.

Measures

The measures for the Alphablox cube are calculated from the fact table in the relational database. When a query requests a measure, the Cube Server calculates the values for the immediate siblings of every member specified in the query. For example, the Cube Server calculates the sales measures for a year as the sum of the sales measures for the twelve months in the year.

Note that in the SQL expression that defines the measures, all column names are qualified with the table they are from in order to prevent problems with ambiguity, which happens when there are different tables with columns having the same name. As a result, there are a few restrictions with the SQL expression for measures:

- 1 The first token in the expression must be a column from the measures table. The following expression is invalid because it starts with an open parenthesis:


```
(store_sales - unit_sales) / store_cost
```
- 2 All columns in the rest of the expression must exist in exactly one table.
- 3 The columns in the expression must not be any of the foreign key columns in the measures table.

3

Creating and Modifying a Cube

An administrator uses the **Alphablox Cubes** section of the **Administration** tab to define Alphablox cubes. This chapter describes the steps necessary to create an Alphablox cube.

Contents

- “Checklist of Tasks to Create a Cube” on page 40
- “Create the Relational Data Source” on page 42
- “Define the Cube” on page 44
- “Define the Measures” on page 45
- “Define the Dimensions and Levels” on page 46
- “Create Cube Adapter Data Source” on page 50
- “Specify and Manage Cube Resources” on page 51
- “Sanity Check the Cube” on page 53

Checklist of Tasks to Create a Cube

This section provides a checklist of tasks needed to define an Alphablox cube, with a brief description of each task. Detailed task instructions appear later in this chapter.

Task	Description
1 Understand the schema of the underlying database.	To define an Alphablox cube, you must know the schema in the relational database from which the cube is built. Use database tools to browse the database and make sure you have access to the names of the tables, columns, primary keys, and foreign keys in the database. For information about schemas, see “Dimensional Schema Design” on page 29.
2 Decide what measures, dimensions, and levels you need for the cube.	In addition to understanding the schema, you must know the data in the relational database underlying an Alphablox cube. You must understand what measures to define in the cube, where those measures are stored in the database, and the relationships between the different levels of the hierarchy for each dimension.
3 “Create the Relational Data Source” on page 42.	Create an Alphablox Analytics data source definition for the underlying relational database from which the Alphablox cube is created.
4 “Define the Cube” on page 44.	Use the Alphablox Cubes administration user interface to define the properties of an Alphablox cube.
5 “Define the Measures” on page 45.	Specify what facts are measured in the Alphablox cube and the mapping for each measure from the relational fact table to the Alphablox cube.
6 “Define the Dimensions and Levels” on page 46.	Specify each dimension in the Alphablox cube and each level for each dimension. Define the mapping between the relational tables and the Alphablox cube dimensions and levels.

Task		Description
7	“Create Cube Adapter Data Source” on page 50.	To query an Alphablox cube, define a data source created with the Alphablox Cube Server Adapter multidimensional driver.
8	“Specify and Manage Cube Resources” on page 51.	Enter the Alphablox cube connection limits, update frequency, and other administrative parameters.
9	“Sanity Check the Cube” on page 53.	Make sure there are no errors in the information you entered to define the Alphablox cube.

Create the Relational Data Source

An Alphablox cube requires that its underlying relational data source be defined in an Alphablox Analytics data source. Each Alphablox cube must reference a relational data source. The data source must reference a relational database with a dimensional schema design. For a description of the relational schema requirements for an Alphablox cube, see “Schema Requirements” on page 27. For a discussion of dimensional schemas, see “Dimensional Schema Design” on page 29.

If you have already defined a data source for the relational database, skip to the next topic, “Define the Cube” on page 44. For more information about data sources, see the *Analysis Server Administrator’s Guide*.

To specify a relational database as an Alphablox Analytics data source, perform the following steps:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Data Sources** link.
- 4 Click the **Create** button.
- 5 From the **Adapter** drop list, select one of the Relational Adapters.
- 6 Enter a name in the **Data Source Name** text box.
- 7 Enter the appropriate information for **Host Name**, **Port Number**, **SID**, and **Database** fields (some of these fields exist only for some of the drivers). If you do not know the correct connect information, contact the database administrator for the relational database to which you are trying to connect.
- 8 Enter a **Default User Name** and **Default Password**. The username and password must be valid on the relational database. The default username and password are always used when an Alphablox cube accesses a relational database. The database user requires read access to the database.



The value of the **Use Alphablox Analytics User Name and Password** drop list is ignored when the data source is being used to populate an Alphablox cube. Use access control lists (ACLs) to control user access to Alphablox cubes. For information about ACLs, see the *Analysis Server Administrator’s Guide*.

- 9 The **Maximum Rows** and **Maximum Columns** text boxes are ignored when the data source is being used to populate an Alphablox cube. You can still enter values and they will be used when other applications use the data source, but an Alphablox cube ignores these text boxes.
- 10 Set the **JDBC Tracing Enabled** drop list to **No** unless you want to write JDBC logging information to the Alphablox Analytics log file. Enable JDBC tracing only if you are experiencing problems and you need to debug their causes.
- 11 Click the **Save** button to save the data source.

Define the Cube

Define the general properties of an Alphablox cube as follows:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Click the **Create** button. A page showing the Create Alphablox Cube General tab appears.
- 5 In the **Alphablox Cube Name** text box, enter a unique name for the Alphablox cube. Allowable characters for Alphablox cube names are A-Z, a-z, 0-9, underscore (_), and space.
- 6 Select **Enabled** from the drop list next to the **Alphablox Cube Name** text box if you want to be able to start the cube. If you do not want to enable the cube so it can be started, leave this set to **Disabled** and enable it after completing the cube definition.
- 7 From the **Relational Data Source** drop list, select the data source for the relational database created in “Create the Relational Data Source” on page 42. Only relational data sources appear in the drop list, and if there are no Alphablox Analytics relational data sources defined, the list is blank.
- 8 Optionally, if you have defined an ACL with which you want to limit user access to the Alphablox cube, enter the ACL name in the **Access Control List** text box and select **Enabled** from the drop list.
- 9 Optionally, enter an MDX query in the **Cache Seeding MDX Query** text box and select **Enabled** from the drop list.

The query entered in this text box is run when the cube is started or rebuilt. The purpose for the query is to populate the in-memory cache with an initial set of results. These results *seed* the cache with data retrieved from the underlying database. Any queries that only require data already in the cache are answered directly from the cache, thus improving response time by avoiding queries to the underlying database. The name of the cube referenced in the FROM clause of the MDX query must be the name defined in the **Alphablox Cube Name** text box.

- 10 Click the **Save** button to save the Alphablox cube.

Define the Measures

All Alphablox cubes must have one or more measures defined. For a description of measures, see “Measures” on page 37. To define measures in an Alphablox cube, perform the following steps:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox cube from the list of cubes and click the **Edit** button. A page showing the Edit Alphablox Cube General tab appears.
- 5 Click the **Measures** tab.
- 6 In the **Fact Table Name** text box, enter the name of the fact table exactly as it is defined in the underlying relational database.
- 7 In the **Name** text box, enter a name for the measure. Allowable characters for measure names are A-Z, a-z, 0-9, underscore (_), and space.

The name will appear in result sets sent to Alphablox Analytics applications, so enter a name that is easy to read and descriptive of its content. For example, if the measure calculates the sum of sales at a store, you can name the measure *Store Sales*.

- 8 From the **Aggregate Function** drop list, select a function (AVG, COUNT, MAX, MIN, or SUM). The function is used in the SQL sent to the underlying database to calculate the measure. SUM is the default.
- 9 In the **Column Name** text box, enter the column on which the aggregation is performed. For example, to create a measure equivalent to the SQL expression *SUM(store_sales)*, select SUM from the **Aggregate Function** drop list and enter *store_sales* in the **Column Name** text box.
- 10 Click the **Add** button to add the measure to the list.
- 11 Repeat the procedure to define any other measures by entering the appropriate information in the **Name**, **Aggregate Function**, and **Measure Expression** text boxes and then clicking the **Add** button.
- 12 Click the **Save** button to update the Alphablox cube definition.



Any changes made to a started Alphablox cube do not take effect until the cube is either restarted or rebuilt. For information on restarting and rebuilding a cube, see “Starting, Stopping, and Rebuilding a Cube” on page 56.

Define the Dimensions and Levels

You must enter information to define the dimensions and levels for the Alphablox cube. The procedure for defining dimensions and levels has three parts:

- Create or Edit the Dimension
- Create and Edit the Levels
- Edit the SQL (Advanced Users Only)

For a description of dimensions and levels, see “Dimensions and Levels” on page 36.

Create or Edit the Dimension

To create or edit a dimension, perform the following steps:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox cube from the list of cubes and click the **Edit** button. A page showing the Edit Alphablox Cube General tab appears.
- 5 Click the **Dimensions** tab.
- 6 Click the **Create** button to define a new dimension or select a dimension from the **Dimensions** list and click the **Edit** button to edit an existing dimension.
- 7 In the **Name** text box, enter a name for the dimension. Allowable characters for dimension names are A-Z, a-z, 0-9, underscore (_), and space.
- 8 Optionally, in the **Description** text box enter a description of the dimension. The description is a comment field only; it has no effect on the dimension definition.
- 9 Click the **Save** button to save the dimension.
- 10 Repeat the procedure to define more dimensions, if appropriate.

Create and Edit the Levels

You should define the levels for a dimension in the order from the most summarized (least granular) to the most detailed (most granular) so you do not need to change the order later. For example, if you are defining the levels *Year*, *Month*, and *Day* for the *Time* dimension, first create the level *Year*, then *Month*, then *Day*. To create or edit levels, perform the following steps:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox Cube from the list of cubes and click the **Edit** button. A page showing the Edit Alphablox Cube General tab appears.
- 5 Click the **Dimensions** tab.
- 6 Select a dimension from the **Dimensions** list and click the **Edit Levels** link. The Level Page for the dimension appears.
- 7 Click the **Create** button.
- 8 In the **Level Name** text box, enter a name for the level. Allowable characters for levels are A-Z, a-z, 0-9, underscore (_), and space.
- 9 In the **Table Name** text box, enter the name of the table from which the level is generated exactly as it is defined in the underlying relational database.
- 10 In the **Column Name** text box, enter the name of the column that contains data with the unique members for the levels. The column must be from the table identified in the **Table Name** text box.
- 11 Check the **Normalized** box if the table from which the level is populated is referenced from another table. The only conditions when the **Normalized** box is *not* checked are:
 - The entire database schema consists of a single table.
 - The level is defined in a table that defines more than one level *and* the other level(s) in the table are at a lower (i.e., less summarized) level (for example, in a star schema).

For more information about the **Normalized** checkbox, see “Dimensions and Levels” on page 36.

- 12 If you checked the **Normalized** box, for each column in the primary key, perform the following:
 - a Enter the name of the primary key column in the **Primary Key Column** text box. The **Primary Key Column** text box refers to the primary key of the table specified in the **Table Name** text box.
 - b In the **Foreign Key Column** text box, enter the name of the foreign key column corresponding to the primary key column entered above. The **Foreign Key Column** text box refers to the foreign key of the table from which the table specified in the **Table Name** text box is referenced.
 - c Click the **Add** button.
 - d Repeat these steps for each primary key/foreign key column pair.
- 13 Click the **Save** button to save the level definition. The definition appears in the list of levels.
- 14 Repeat the procedure to define more levels, if appropriate.
- 15 Move the levels up or down so they are in the order from the most summarized (least granular) to the most detailed (most granular). When the levels are in the correct order, the most summarized has the number 1 next to it in the list of levels and each subsequent level is numbered sequentially. To move the levels up or down, select a level from the list and click the up or down buttons to change the order respectively.

Dimension 'Time' Level Page close

Levels

- (1) Year
- (2) Quarter
- (3) The Month
- (4) Day of Month

Edit Level "Day of Month"

Level Name:

Table Name:

Column Name:

Normalized

Primary Key Column:

Foreign Key Column:

- 16 If you need to delete a level, select a level from the list and click the **Delete** button.

- 17 Click the **Close** button when you are done editing the levels.

Edit the SQL (Advanced Users Only)

The Edit SQL link allows you to edit the SQL statement used to generate the dimension. You can edit the SQL statement to customize the query that is sent to the underlying database. The main reason to modify the SQL is to change the ORDER BY clause to order by a different column. The members will be displayed in the order specified in the SQL ORDER BY clause.



Use this feature with caution and only if you are very familiar with the SQL supported by the underlying database. The Alphablox Cube Server sends the SQL statement to the database exactly as it is edited. Editing the SQL incorrectly can result in syntax errors, poor performance, or even wrong answers from the underlying database.

To modify the SQL used to populate the dimensions, perform the following steps:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox cube from the list of cubes and click the **Edit** button. A page showing the Edit Alphablox Cube General tab appears.
- 5 Click the **Dimensions** tab.
- 6 Select a dimension from the **Dimensions** list and click the **Edit SQL** link. The Edit SQL Page for the dimension appears.
- 7 Modify the SQL statement as necessary.
- 8 Click the **Save** button to save the changes to the dimension definition. If the system prompts you for confirmation of the change, read the dialog box and then confirm the change.
- 9 If you want to revert the SQL to the system-generated statement, click the **Default** button, then click the **Save** button.
- 10 Click the **Close** button to close the window.



If you modify the dimension definition and have edited the SQL for the dimension, you must then re-edit the dimension to reflect any changes to the definition. If the modifications you are making require adding columns to the SELECT list, you must add the columns at the end of the SELECT list.

Create Cube Adapter Data Source

To query an Alphablox cube, an Alphablox Analytics data source that uses the **Alphablox Cube Server Adapter** must be defined. A single data source can be used to access multiple Alphablox cubes from multiple applications. The cube that is accessed is determined by the FROM clause of the MDX query used by an Alphablox application. To create an Alphablox Cube Server Adapter data source, perform the following steps.

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Data Sources** link.
- 4 Click the **Create** button.
- 5 From the **Adapter** drop list, select the adapter named **Alphablox Cube Server Adapter**.
- 6 Enter a name in the **Data Source Name** text box.
- 7 Optionally, enter a description in the **Description** text box.
- 8 Specify a number in the **Maximum Rows** and the **Maximum Columns** text boxes. The values limit the number of rows or columns returned for queries entered through this data source. The default values are 1000.
- 9 Click the **Save** button to save the data source.

Specify and Manage Cube Resources

For each Alphablox cube, you can define a schedule for refreshing its data from the underlying database. You can also set several tuning parameters for each cube.

Defining a Refresh Schedule

If the data in the relational database that underlies an Alphablox cubes changes, any data cached in an Alphablox cube might be stale. When the data becomes stale, you should rebuild the cube to guarantee that answers derived from the Alphablox cube are correct with respect to the underlying database. You can manually rebuild the cube, which rebuilds the dimensions and empties the in-memory cache, by either stopping and restarting the Alphablox cube or using the `REBUILD CUBE <cube_name>` console command. Alternately, if the dimensions have not changed but new or changed data has been added to the database, you can manually empty only the in-memory cache by using the `EMPTYCACHE <cube_name>` console command.

If the underlying database is updated at regular and predictable intervals, it might make sense to schedule regular updates to the Alphablox cube that references that database. For example, if the database is updated every night at 9:00 PM, you might want to rebuild the Alphablox cube every morning at 3:00 AM.

To configure an Alphablox cube to rebuild itself at regular intervals, perform the following steps:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox Cube from the list of cubes and click the **Edit** button. A page showing the Edit Alphablox Cube General tab appears.
- 5 Click the **Schedule** tab.
- 6 Check the **Refresh every** box to enable scheduled Alphablox cube rebuilding.
- 7 Set the refresh interval by clicking the desired buttons and modifying the corresponding time periods. For example, to set the Alphablox cube to rebuild every day at 3:00 AM, select the second button and enter 3:00 AM for the time.
- 8 Click the **Save** button to update the Alphablox cube definition.

Setting Tuning Parameters

For each Alphablox cube, you can set several tuning parameters for resource management. To do so, perform the following steps:

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox cube from the list of cubes and click the **Edit** button. A page showing the Edit Alphablox Cube General tab appears.
- 5 Click the **Tuning** tab.
- 6 Check the box to the left of each parameter you want to enable and specify a numerical value for the limit. The following table shows each available parameter and its description. For more detailed information of these and other tuning parameters, see “Tuning a Cube” on page 65.

Tuning Parameter	Description
Maximum Connections	The maximum number of concurrent connections to this Alphablox cube. The limit is reached only when the connections are all executing queries simultaneously. When the limit is reached, a new connection must wait for a free connection.
Maximum Data Source Connections	The maximum number of connections made to the underlying relational database. When the limit is reached, a new connection must wait for a free database connection. When using this limit, once each connection is opened it remains open (up to the specified limit) for use by other SQL queries. When not using this limit, each query uses and then closes a separate connection.
Maximum Rows Cached	The maximum number of rows returned from the database to be stored in the Cube Server’s in-memory cache. This limit is controlled separately for each cube. When this limit is reached, the results from the least recently used cached queries are aged out of the cache to make room for the new rows.

- 7 Click the **Save** button to update the Alphablox cube definition.

Sanity Check the Cube

It is usually worthwhile to take a few minutes after you have created an Alphablox cube to ensure that the measures, dimensions, and levels are defined correctly. If you find any errors, you can easily correct them. To do a “sanity” check on an Alphablox cube, perform the following steps.

- 1 Log into the Alphablox Analytics home page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox cube from the list of cubes and click the **Edit** button. A page showing the Edit Alphablox Cube General tab appears.
- 5 Verify that the data source specified in the **Relational Data Source** text box references the desired relational database. You might need to check the settings for the data source on the **Data Sources** administration page.
- 6 Before attempting to start the Alphablox cube, verify that **Enabled** is selected from the drop list next to the **Alphablox Cube Name** text box.
- 7 Click the **Measures** tab.
- 8 From the **Measures** page, check the following:
 - a Make sure that the table listed in the **Fact Table Name** text box is the correct table in the relational schema, the name is spelled correctly, and the name is not a qualified name.
 - b For each defined measure, check that the desired aggregation is selected from the **Aggregation Function** drop list and that the column listed in the **Column Name** text box is the correct column from the relational fact table, the name is spelled correctly, and the name is not a qualified name.
- 9 Correct and save any discrepancies you find on the **Measures** page.
- 10 Click the **Dimensions** tab.
- 11 Verify that all the desired dimensions have been defined, and that the names are correct. To change any of the dimension names, select the dimension from the **Dimensions** list, click the **Edit** button, and make any desired changes. Click the Save button to **Save** the changes.
- 12 For each dimension, check that all the levels are defined correctly:
 - a Select the dimension from the **Dimensions** list and click the **Edit Levels** link. The Level Page for the dimension appears.
 - b Verify that the levels are in the correct order. The first level (1) should be the most summarized level, and each successive level should be the next

level down in the hierarchy. For example, if the hierarchy for the *Time* dimension is *Year, Month, Day*, then *Year* should be the first level (1), followed by *Month* (2), followed by *Day* (3). If the levels are in the wrong order, correct the order by selecting a level from the **Levels** list and clicking the up or down buttons. Continue this process until the levels are ordered correctly.

- c For each level, perform the following steps:
 - a Select a level from the **Levels** list and click the **Edit** button.
 - b Make sure that the table listed in the **Table Name** text box is the correct table in the relational schema, the name is spelled correctly, and the name is not a qualified name.
 - c Make sure that the column name listed in the **Column Name** text box is the correct column from the relational table, the name is spelled correctly, and the name is not a qualified name.
 - d Verify that the **Normalized** box is checked appropriately. If the relational schema is a star schema where all of the levels are stored in a single dimension table, only the last level in the list should have the **Normalized** box checked. If each level is stored in separate table which is referenced by a foreign key from the next level down, then the **Normalized** box should be checked for all the levels. For more information about the **Normalized** checkbox, see “Dimensions and Levels” on page 36.
 - e Make any necessary changes and click the **Save** button to save your changes.
 - f After checking all the levels in the dimension and making any necessary changes, click the **Close** button to close the Level Page.
- 13 If you made any changes to any of the dimensions *and* you had previously used the **Edit SQL** page to change the dimension ordering, be sure to update your changes on the **Edit SQL** page.
- 14 Click the **Schedule** tab and verify that all the settings are the way you want them.
- 15 Click the **Tuning** tab and verify that all the settings are the way you want them.

After completing the sanity check for the Alphablox cube, you can start the cube. For details on starting an Alphablox cube, see “Starting, Stopping, and Rebuilding a Cube” on page 56.

4

Maintaining a Cube

Alphablox Cube Server provides functionality to perform administrative tasks on Alphablox cubes. These tasks are performed either through the Alphablox Analytics administration user interface or through the server Console. This chapter describes these tasks.

Contents

- “Starting, Stopping, and Rebuilding a Cube” on page 56
- “Deciding on an Administration Strategy” on page 60
- “Console Commands” on page 62
- “Modifying a Cube” on page 64
- “Tuning a Cube” on page 65

Starting, Stopping, and Rebuilding a Cube

The most common administrative tasks you need to perform on an Alphablox cube are to start, stop, and rebuild the cube.

Starting an Alphablox Cube

You must start an Alphablox cube to make it available for querying. You can start a cube either from the Alphablox Analytics Home Page from the command line of a Console window. When you start a cube, the Cube Server runs queries to the underlying relational database. The results of these queries are used to load the dimension members into the cube's in-memory cache. An Alphablox cube can have a cache seeding MDX query specified as part of its definition, which is used to precompute some results to store in the cube's cache. If one is specified, at startup time the Cube Server runs the MDX query against the Alphablox cube to populate the cache with the measure values returned from the MDX query.

Start Cube From the Home Page

To start an Alphablox cube from the Alphablox Analytics Home Page, perform the following steps:

- 1 Log into the Alphablox Analytics Home Page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab. The **Server** page appears.
- 3 Under the **Runtime Management** section, click the **Alphablox Cubes** link.
- 4 From the **Alphablox Cubes** list, select the Alphablox cube to start.
- 5 To view the current status of the Alphablox cube, click the **Details** button.
- 6 Click the **Start** button. When the Alphablox cube has completed the startup operation, the status field displays **Running**.

Start Cube From a Console Window

To start an Alphablox cube from a console window, perform the following.

- 1 If Alphablox Analytics is not already running, start it. See *Analysis Server Administrator's Guide* for details about starting Alphablox Analytics.
- 2 In a Console window, enter the following command:

```
start cube cube_name
```

where *cube_name* is the name of the Alphablox cube to start. You can use the window that opens when you start Alphablox Analytics open a Console window from the **Launch Server Console** link on the **Server** administration page.

Troubleshooting If the Cube Does Not Start

If the Alphablox cube fails to start, an error message appears that can help determine why the problem. When troubleshooting a problem, the following logging tools can provide more information:

- Check the Alphablox Analytics log file.
- Raise the message level on the Console to DEBUG by entering the following in a Console window:

```
report debug
```

- Enable JDBC tracing in the Alphablox Analytics relational data source.

For information about enabling any of these logging options, see the *Analysis Server Administrator's Guide*.

The following table shows some common scenarios that might cause the startup operation to fail and lists suggestions for correcting the problem. After you determine the problem, correct it and try to start the Alphablox cube again.

Error	Description
<p>“Make sure the cube is enabled.”</p>	<p>Check to see if the cube is enabled. On the command line, enter the following to see if the cube is enabled:</p> <pre>show cube cube_name</pre> <p>To enable an Alphablox cube, from the General tab in the Alphablox Cubes user interface, select Enabled from the drop list next to the Alphablox Cube Name text box.</p>
<p>An error connecting to the underlying database.</p>	<p>Connection errors can be caused by a variety of problems. The following are some common things to check:</p> <ul style="list-style-type: none"> • Check that the relational data source has the correct connect information. • Check that the relational data source has a valid, non-null username and password. • Make sure the database is available for connections.

Error	Description
A syntax error from the underlying database.	Syntax errors from the relational database generally indicate an error in the cube definition. For example, if the syntax error indicates that a column is not found, check the dimension definitions to ensure that the column and table names are named exactly as they are in the database.

Stopping an Alphablox Cube

Stopping an Alphablox cube makes it unavailable for querying and removes all entries in the cube's in-memory cache and all the dimension members from the cube's outline.

Stop Cube From the Home Page

To stop an Alphablox cube through the Alphablox Analytics Home Page, perform the following steps:

- 1 Log into the Alphablox Analytics Home Page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab. The **Server** page appears.
- 3 Under the **Runtime Management** section, click the **Alphablox Cubes** link.
- 4 Select the Alphablox cube you want to stop from the **Alphablox Cubes** list.
- 5 To view the current status of the Alphablox cube, click the **Details** button.
- 6 Click the **Stop** button. When the Alphablox cube has completed the shutdown operation, the status field displays **Stopped**.

Stop Cube From a Console Window

To stop an Alphablox cube from a Console window, enter the following command:

```
stop cube cube_name
```

where *cube_name* is the name of the Alphablox cube to stop. You can use the window that opens when you start Alphablox Analytics open a Console window from the **Launch Server Console** link on the **Server** administration page.



The Alphablox cube will not stop until any executing queries have completed.

Rebuilding an Alphablox Cube

You should either rebuild or restart an Alphablox cube when the data, including the dimension data, changes in the underlying database. You must rebuild or restart (or wait for the next refresh interval, if one is configured) the cube when you change the cube definition in order for the changes to take effect in queries.

During a rebuild operation, the cube is unavailable for querying; new queries wait and are executed after the rebuild operation completes. The rebuild operation waits until any running queries complete before starting the operation. The size of the dimensions and the performance of the queries that populate the dimension from the underlying database determine how long the operation takes.

To rebuild an Alphablox cube enter the following command from a Console window:

```
rebuild cube cube_name
```

where *cube_name* is the name of the Alphablox cube to rebuild. You can use the window that opens when you start Alphablox Analytics open a Console window from the **Launch Server Console** link on the **Server** administration page.

If the dimension data has not changed but the fact data has (for example, the sales numbers for the last quarter were added to the database), then you can empty the contents of the in-memory cache only. To empty all the entries in the cache but leave the dimension members as is, enter the following command from a Console window:

```
emptycache cube cube_name
```

Deciding on an Administration Strategy

After you have defined and started an Alphablox cube, maintenance tasks are needed only if one of the following occurs:

- The data changes in the underlying database.
- The cube definition changes.

Because the Alphablox cube resides in memory, there is no disk space to manage. There are memory considerations, but those are not usually day-to-day administrative tasks. For information about memory issues, see “Alphablox Cube Memory Considerations” on page 68.

You do have to be aware of the environment in which the underlying relational database operates. The way the underlying database is managed can have important implications on an Alphablox cube.

Understanding the Database Environment

Every time data changes in the database underlying an Alphablox cube, parts of the cube potentially become out of date. An Alphablox cube gets its data from queries to the underlying database. When a query requests data from an Alphablox cube, Alphablox Cube Server checks to see if the results are in its in-memory cache. If the results are there, they are immediately available to the application, resulting in very fast response times. Although the results were originally retrieved from the underlying database, those results were retrieved at some point in the past. If the data has not changed, there is no problem. If the data in the underlying database changed between the time when the cache entry occurred and the time a query asks for the results, then the results are out of date.

Furthermore, if some of the members in the Alphablox cube were inserted, updated, or deleted from the database, the results from the Alphablox cube would not reflect the true state of the dimensions. The results from a new query to the Alphablox cube might still match the results in the underlying database, but they might not. It depends on exactly what values changed in the database, what is stored in the in-memory cache for the Alphablox cube, and what data the query requests.

Because there is no sure way to know if the Alphablox cube is still valid and up-to-date when the data in the underlying database changes, the safest action is to rebuild the cube. Therefore, it is critical to know when and how the underlying database changes.

For example, if you know the database never changes, you never need to rebuild the Alphablox cube. If the database only adds new data to parts of the database you do not have defined in the cube, you might not need to rebuild.

If the database is updated nightly with potential changes to all parts, you probably need to rebuild the Alphablox cube nightly, after the database update is completed. The more you know about the environment in which the database operates, the better you can predict when the data in your Alphablox cube becomes stale.

Scheduling Periodic Updates

It is very common for data warehouse and data mart databases to be updated on a known schedule. Based on that schedule, you can schedule periodic updates to Alphablox cubes. You can perform the updates ad-hoc with the `REBUILD CUBE` or the `EMPTYCACHE CUBE` commands. Alternatively, you can set up an automated rebuild schedule for each Alphablox cube. For details on setting up an automatic schedule, see “Defining a Refresh Schedule” on page 51.

There is no one best way to schedule updates to an Alphablox cube. It is very important to know what is going on in the relational database. It is equally important to know the habits and requirements of your user community. Rebuilding an Alphablox cube might take some time, depending on the size of the cube and its underlying database. Typically, the best time to schedule rebuilds is late at night when there are few or no users on the system. Also, particularly if the rebuild operations take a long time, make sure the users know that the cube will not be available during those times.

Console Commands

You can perform most cube management tasks from the Alphablox Analytics console window. To access the console, click the **Administration** tab, **Server** page, **Launch Server Console** link, or use the Alphablox Analytics Console window that opens when you start Alphablox Analytics. The following table lists the cube commands and a description of what each does.

Command Syntax	Description
<code>delete cube <cube_name></code>	Deletes a cube and its entire definition.
<code>disable cube <cube_name></code>	Sets a cube in the disabled state. A disabled cube cannot be started until it is enabled and therefore does not automatically start when Alphablox Analytics starts. A cube should be stopped before it is disabled.
<code>emptycache cube <cube_name></code>	Removes all entries from the cube's in-memory cache. After emptying the cache, a query against the Alphablox cube must retrieve the results from the underlying database. Use this command when the underlying database has changed to ensure the results retrieved from the Alphablox cube are equivalent to the data stored in the database. Note that the <code>EMPTYCACHE</code> command does not rebuild the dimension outlines of the cube. To rebuild the dimension outlines, use the <code>REBUILD</code> command or stop and start the cube.
<code>enable cube <cube_name></code>	Sets a cube to the enabled state. A cube must be enabled before it can be started. Enabled cubes start automatically when Alphablox Analytics starts.

Command Syntax	Description
<code>rebuild cube <cube_name></code>	First removes the member names for all the dimensions and all the measures from the in-memory cache; then queries the underlying database to repopulate the dimension member names for all the dimensions. If an initial MDX cache seeding query is specified in the cube definition, that query is executed to populate the cache.
<code>show cube <cube_name></code>	Shows the current status of the cube. The cube status can be: <ul style="list-style-type: none"> • disabled • stopped • starting • running To show the status of all defined Alphablox cubes, enter the following command: <pre style="text-align: center;">show cube</pre>
<code>start cube <cube_name></code>	Starts a cube and makes it available for querying. When a cube starts, it queries the underlying database to populate the dimension members and runs the MDX cache seeding query (if one is specified in the cube definition).
<code>stop cube <cube_name></code>	Stops a cube that is running. When a cube stops, it becomes unavailable for querying and the dimension members and the measures are removed from the in-memory cache.

Modifying a Cube

You can change any part of the Alphablox cube definition at any time. Changes to a stopped cube apply immediately. Changes to a running cube are saved to the cube definition immediately but are not applied to the running cube until it is either rebuilt or restarted, either through the Console or scheduled refreshes.

You use the **Alphablox Cubes** administration page to modify an Alphablox cube the same way as you create one. You can update any part of the cube definition and save it. For details on how to enter your definitions in each part of the user interface, see “Creating and Modifying a Cube” on page 39.

Tuning a Cube

There are a number of administrative controls for tuning and configuring Alphablox cubes. Because Alphablox cubes run in memory and can potentially grow to use large amounts of memory, you should be aware of some memory considerations.

Tuning Controls

Use the controls described in this section to control the resources Alphablox cubes.

Connection and Cache Size Limits

You can specify connection and cache size limits for each defined Alphablox cube on the **Alphablox Cubes** page, **Tuning** tab.

Maximum Connections

When there are many users querying the Alphablox cube simultaneously, machine resources on the computer running Alphablox Analytics can be consumed faster than if there are only a few users. Keep in mind, however, that the queries have to be executing at *exactly the same time* for there to be contention for resources. This might not happen very often, even if there are many users connected at the same time. If this becomes a problem on your system, you can limit the number of connections allowed for each Alphablox cube.

The amount of resources used is completely dependent on the types of queries that are being issued. Many queries use very little machine resources, but some long-running queries might consume significant resources.

Maximum Data Source Connections

The **Maximum Data Source Connections** limits the number of connections to the underlying database. If the box is checked, it enables “connection pooling” to the database. In this mode, each time a query is sent to the database, a connection is opened and the query issued. The connection remains open, even after the query completes, for subsequent queries. When another query is sent, if there is an open and idle connection, it is used. If all the connections are busy, then it opens a new connection, up to the limit specified.

The database “connection pooling” is useful because it limits the number of backend database connections. The result is that after a period of time, you might have the specified number of connections open to the database, but never any more than the specified number.

If the box is not checked, each query the Cube Server sends to the database opens a new connection and then closes it when the results are returned. The new connections are opened regardless of the status of any of the other connections. The connections are never shared and are never left idle.

Each connection to the database has a cost associated with it, however small. In many cases, the difference in response time is not noticeable, but in some cases it might be. It is also possible that the underlying database might restrict the number of connections it accepts, so the DBA might not want you using up too many connections. If you do not want to keep connections open, do not check the **Maximum Data Source Connections** box.

Maximum Rows Cached

The **Maximum Rows Cached** limits the number of rows returned from the database that the cache can store. If the box is checked, when the limit is reached, the least recently used rows are removed to make room for rows returned by a new query. If the box is not checked, there is no limit to the size of the cache, allowing it to grow indefinitely (up to the amount of data in the underlying database). In some situations, not checking the box can cause the system to run out of memory. For more information about memory, see “Alphablox Cube Memory Considerations” on page 68.

The more data that is stored in the cache, the less often queries to the Alphablox cube will need to retrieve results from the underlying database, thus providing faster query response time. However, if the cache grows too big, it will use up memory on the machine, potentially slowing the performance for all users. To find the optimal size for your system, you will need to experiment and consider memory resources, user load, and query load. Depending on the user and query loads, balance the trade offs to decide the best cache size.

To specify a limit for the number of connections, number of data source connections, and the maximum cache size for each Alphablox cube, perform the following steps:

- 1 Log into the Alphablox Analytics Home Page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab.
- 3 Click the **Alphablox Cubes** link.
- 4 Select the Alphablox cube from the list of cubes and click the **Edit** button. A page showing the **Edit Alphablox Cube General** tab appears.
- 5 Click the **Tuning** tab.

- 6 Check any of the boxes for the limits you want to set and enter a corresponding number.
- 7 Click the **Save** button to save the limits to the Alphablox cube definition.

Maximum Number of Cubes

If you have defined many Alphablox cubes, and if each cube starts using large amounts of memory and machine resources, the performance of your entire system will be affected. To help control this, you can limit the number of Alphablox cubes allowed to run in Alphablox Analytics. The limit controls the number of Alphablox cubes that can run simultaneously; it does not limit the number that can be defined.

To set a limit on the number of concurrently running Alphablox cubes, perform the following steps:

- 1 Log into the Alphablox Analytics Home Page as the *admin* user or as a user who is a member of the administrators group.
- 2 Click the **Administration** tab. The **Server** page appears.
- 3 Under the **Server Properties** section, click the **Alphablox Cube Manager** link.
- 4 Check the box labeled **Maximum Cubes** and enter a number for the limit you want to set.
- 5 Click the **Save** button to save your changes.

Maximum Rows and Columns

By restricting the maximum number of rows and columns in an Alphablox Cube Data Source, you can restrict applications from issuing queries that return large amounts of data. You set these limits in the Alphablox cube data source on the **Data Sources** administration page. The data source is the one used to issue MDX queries against an Alphablox cube.

Alphablox Cube Memory Considerations

The Alphablox Cube Server runs as part of the Java process in which Alphablox Analytics runs. Therefore, as the Cube Server uses more memory, the Java process uses more memory. The memory limits for the Alphablox Analytics Java process are set at installation. If you find that the Alphablox Analytics is running out of memory due to Alphablox cubes using large amounts of memory, there are several possible actions you might take:

- Limit the size of the in-memory cache for each cube. For details, see “Connection and Cache Size Limits” on page 65.
- Limit the number of Alphablox cubes in the system. For details, see “Maximum Number of Cubes” on page 67.
- Change the maximum size of the memory heap for the Java process in which Alphablox Analytics runs. For details, see Changing the Maximum Memory Heap Size below.
- Increase the memory capacity of the computer in which Alphablox Analytics runs. For details, see “Adding More Memory to Your System” on page 71.

Changing the Maximum Memory Heap Size

The maximum memory heap size is set in the initialization file for Alphablox Analytics. The limit determines the maximum memory size to which the Alphablox Analytics Java process can grow. The Cube Server runs as part of that Java process. If you are experiencing out-of-memory errors in Alphablox Analytics, you might need to raise the maximum memory heap size of the Java process.

The memory limit is configured during installation, and can also be changed by modifying the initialization file used to start Alphablox Analytics. After you change the maximum memory size, you must restart Alphablox Analytics for the change to take effect.

Set the maximum memory heap size to a value high enough to accommodate your memory requirements but low enough so that it does not cause the operating system to excessively swap to disk when the process size approaches the maximum. Also, leave some room for unexpected memory use on the machine. For example, if your machine has 1024 megabytes of memory and other resources on the machine use about 300 megabytes of memory, consider setting the maximum memory heap size to a value as large as 600 megabytes.

It might require some experimentation to find the ideal maximum for your system. If you are not having any problems, performance is good, and there are no out-of-memory errors in your Alphablox cubes, then the limits are set well for your environment.

Setting the Maximum Memory Heap Size During Installation

On Windows NT systems, you can set the maximum memory heap size of the Java process in which Alphablox Analytics runs during the installation process. For more detailed installation instructions, see the *Installation Guide*. To set the maximum memory heap size during installation, perform the following steps:

- 1 From the installation media, run the *setup.exe* file to start the installation.
- 2 If you are upgrading an existing release of Alphablox Analytics, click the **No** button on the **Analysis Server Auto-Upgrade Installation Options** screen. The **General Installation Information** screen appears.
- 3 Enter any appropriate information on the **General Installation Information** screen and click the **Next** button.
- 4 Click the **Advanced** button on the **Analysis Server Configuration Information** screen.
- 5 On the **Advanced Analysis Server Settings** screen, enter the appropriate value in the **Maximum size** field of the **JRE Heap Sizes in Megabytes** section.
- 6 Enter any other appropriate information and click the **Continue** button.
- 7 Enter any appropriate information in the **Analysis Server Configuration Information** screen and click the **Next** button.
- 8 Check the **Review Installation Options** screen to ensure that all of the options you have set were entered correctly. If everything is correct, click the **Install** button.

Setting the Maximum Memory Heap Size By Modifying the Initialization File

You can modify the initialization file to change the maximum memory heap size of the Java process in which Alphablox Analytics runs. To modify the maximum memory heap size, perform the following steps. The changes you make do not take effect until you restart Alphablox Analytics.

- 1 Before modifying the initialization file, make a copy of it and save as a different name. For example, copy the *AnalysisServer.bat* file to a file named *AnalysisServer.bat.old*.
- 2 On Windows NT systems, open the following file in a text editor such as *Notepad*:

```
alphablox_directory\analytics\AnalysisServer\AnalysisServer.bat
```

where *alphablox_directory* is the directory in which Alphablox Analytics is installed.

- 3 On Sun Solaris systems, open the following file in a text editor such as *vi*:

```
alphablox_directory/analytcs/AnalysisServer/AnalysisServer.sh
```

where *alphablox_directory* is the directory in which Alphablox Analytics is installed.

- 4 Find the line in the initialization file that starts with the word *java*.
- 5 On the line that starts with the word *java*, find the *-mx* flag. It looks similar to the following:

```
-mx150M
```

- 6 Change the numeric value of the *-mx* flag to the maximum value in which you want to allow the memory heap size to grow. For example, to set the maximum memory heap size to 600 megabytes, set the *-mx* flag as follows:

```
-mx600M
```

- 7 Make sure you do not change any other values on the initialization file.
- 8 Save your changes and close the file.
- 9 On Windows NT systems, if Alphablox Analytics is configured to run as a service, edit the Windows NT registry as follows:

- a From the Windows **Start** menu, **Run** command, start the **Registry Editor** by entering the following command:

```
regedit
```

- b Navigate to the following folder:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AAS_<instance>
```

where *instance* is the name of your Alphablox Analytics instance (by default, *AlphabloxAnalytics*).

- c Select the **Arguments** string, right click and select **Modify**. The Edit String dialog appears.
- d Change the numeric value of the *-mx* flag to the maximum value in which you want to allow the memory heap size to grow. For example, to set the maximum memory heap size to 600 megabytes, set the *-mx* flag as follows:

```
-mx600M
```

- e Make sure you do not change any other values in the string.
- f Click the **OK** button.
- g Exit the **Registry Editor**.

10 Restart Alphablox Analytics for the changes to take effect.

Adding More Memory to Your System

One often overlooked solution to memory issues is to add more memory to the system in which Alphablox Analytics runs. Check with your hardware vendor to determine how much memory you can install on your computer. As the memory use on a system grows toward the limits of the installed physical memory, the system will swap memory to disk to make room for new memory requests, resulting in much more inefficient memory management.

A memory upgrade is often a relatively inexpensive way to increase server capacity. Also, it often helps or eliminates memory usage issues. If there is room on the system for adding more memory, consider doing so.

5

Using MDX to Query Alphablox Cubes

Applications use the multidimensional expressions (MDX) language to query an Alphablox cube. Alphablox cubes support a subset of the MDX syntax. This chapter describes the supported MDX syntax for querying Alphablox cubes and provides example queries.

Contents

- “Supported MDX Syntax” on page 74
- “Example MDX Queries” on page 83

Supported MDX Syntax

MDX is a multidimensional query language used by several multidimensional databases including Microsoft OLAP Services. Alphablox Cube Server uses a subset of the MDX syntax as the query language for Alphablox cubes. For an Alphablox Analytics application that accesses an Alphablox cube, the MDX query is used as the value for the DataBlox query parameter (or associated methods).

Basic Syntax

The basic syntax for an MDX query against an Alphablox cube is as follows:

```
SELECT {axis_specification} ON COLUMNS,
       {axis_specification} ON ROWS
FROM cube_name
WHERE ( slicer_items)
```

where:

<i>axis_specification</i>	is a set of one or more tuples. Tuples can be entered as a list or “generated” with the CROSSJOIN function.
<i>cube_name</i>	is the name of a defined Alphablox cube.
<i>slicer_items</i>	is a tuple (often a comma-separated list of members) on which the query result set is filtered. If there is more than one slicer member, each must be from a different dimension, and the dimension cannot be referenced in any of the axes specified in the query.

Usage Notes

A dimension can only appear on a single axis in a query. Queries that place a dimension on more than one axis fail with an error.

A query can specify zero or more axes, although it is typical to specify two axes. The COLUMNS axes can also be specified as AXIS(0), the ROWS axis as AXIS(1). Subsequent axes are referred to as AXIS(*n*), where *n* is the next consecutive integer. Note that Alphablox Analytics applications that display the data from a query (GridBlox, ChartBlox, or PresentBlox) can only accept queries with at most two specified axes. A query rendered as an XML data set can accept any number of axes.

The keywords in MDX are not case sensitive, but member names in an MDX query are case sensitive when surrounded by square brackets []. When member names are not surrounded by square brackets [], they must be entered with uppercase letters.

Specifying Member Sets

A *member set* is comprised of one or more members from the same dimension. It is good practice to always enclose member names in square brackets [], although it is not required. When a member name contains spaces, the square brackets are required. Member names are case sensitive; therefore, the following member specifications are not equivalent:

```
[Time].[Fiscal Year]
[Time].[fiscal year]
```

Qualified Member Names

You can qualify a member name by using the dimension name and its parents in the hierarchy, similar to object syntax, as follows:

```
[Dimension].[Level].[Member]
```

You can also qualify a member name by using the dimension name and one or more ancestors of the member, as follows:

```
[Dimension].[Member].[Member]
```



Always qualify a member name at least enough to make it unique.

Curly Braces

Curly braces denote sets, and a set placed on an axis in an MDX query must be enclosed in curly braces { }. For example, the syntax to specify a set containing the products Golden Oats and Sugar Grains is as follows:

```
{[Product].[Golden Oats], [Product].[Sugar Grains]}
```

FROM:TO Syntax

You can specify a member set that extends from one point in the level to another (inclusive) by using a colon (:) to separate the members. For example, if you have a dimension called *Alphabet* with members A-Z, the following evaluates to the set {D, E, F, G, H}:

```
{[Alphabet].[D]:[Alphabet].[H]}
```

The order of the members in the dimension is based on the order specified in the level definition on the administration page accessed by clicking the **Administration** tab, **Alphablox Cubes** link, **Dimensions** tab, **Edit Levels** link from the Alphablox Analytics Home Page.

Functions

Functions simplify and broaden the possible scope of MDX queries. This section provides the basic syntax and describes the functions supported for MDX queries against Alphablox cubes. In general, Alphablox cubes support member functions, not calculated functions. Function names are not case sensitive.

CHILDREN

The CHILDREN function returns the member set for the level directly below the specified member or dimension. The CHILDREN function has the following syntax:

```
member.CHILDREN
```

or

```
dimension.CHILDREN
```

where:

member is an expression that evaluates to a single member.

dimension is an expression that evaluates to a dimension defined in the Alphablox cube.

The CHILDREN function does not operate on a level, only on a dimension or on a specific member.

It is safest to fully qualify the member names and to put square brackets [] around each element of the member name. Fully qualifying the names ensures that the expression is not ambiguous (although the Cube Server will report ambiguous member names).

Examples

For the following examples, assume there is a dimension named *Time* with levels *Year* and *Quarter*. *Year* has members named 1999, 2000, and 2001. *Quarter* has members named Q1, Q2, Q3, and Q4.

Example	Evaluates To	Explanation
[Time].CHILDREN	1999, 2000, 2001	The children of the <i>Time</i> dimension are all of the members of the first level, <i>Year</i> , in the dimension.

Example	Evaluates To	Explanation
[Time].[Year].[2000].CHILDREN	Q1, Q2, Q3, Q4	The expression [Time].[Year].[2000] evaluates to the member name 2000 from the <i>Year</i> level. The children of that member name are the names of the four quarters.

CROSSJOIN

The **CROSSJOIN** function performs a cross product (calculates all possible combinations) of two sets, generating a new set of tuples. Because it generates a set, you can nest the **CROSSJOIN** function if you want to take the cross product of more than two sets. The **CROSSJOIN** function is useful when you need to have multiple dimensions on a single axis. The **CROSSJOIN** function has the following syntax:

```
CROSSJOIN ({set1}, {set2})
```

The syntax for nesting the **CROSSJOIN** function is as follows:

```
CROSSJOIN (CROSSJOIN ({set1}, {set2}), {set3})
```

For example, the following statement:

```
CROSSJOIN ({[1999], [2000]}, {[Diamonds], [Emeralds]})
```

evaluates to the following set of tuples:

```
{([1999],[Diamonds]), [1999],[Emeralds]}, ([2000],[Diamonds]),  
([2000],[Emeralds])}
```

This set might look like the following on an axis in a grid result set:

1999	Diamonds
	Emeralds
2000	Diamonds
	Emeralds

CURRENTMEMBER

The CURRENTMEMBER function returns the member being currently manipulated in a GENERATE operation. The CURRENTMEMBER function is useful during an operation that searches recursively through a dimension. The CURRENTMEMBER function has the following syntax:

```
dimension.CURRENTMEMBER
```

For an example, see “GENERATE” on page 80.

DESCENDANTS

The DESCENDANTS function returns a member set derived from a specified boundary in the hierarchy. It is similar to the CHILDREN and MEMBERS functions but provides more choices for how to enumerate the member set. The DESCENDANTS function has the following syntax:

```
DESCENDANTS (member, level [, boundary])
```

where:

member is an expression that evaluates to a single member.

level is an expression that evaluates to a level defined in the Alphablox cube.

boundary is an optional keyword defining the members to include. Possible values are:

SELF	The default if no argument is specified. Returns members from the specified level only. If the specified level contains the specified member, returns only the specified member.
BEFORE	Returns all members between the specified member and the specified level, not including the specified level but including the specified member.
AFTER	Returns all members from all the levels below the specified level.
BEFORE_AND_AFT ER	Returns all members between the specified member and the specified level, not including the specified level but including the specified member, and all members from all the levels below the specified level.
SELF_AND_AFTER	The same as after, but also includes all the members of the specified level.
SELF_BEFORE_AFT ER	Returns all members between the specified member and the bottom of the hierarchy, including the specified level and including the bottom level.

Examples

For the following examples, assume there is a dimension named *Time* with levels *Year*, *Quarter*, and *Month*. *Year* has members named 1998, 1999, 2000, and 2001. *Quarter* has members named Q1, Q2, Q3, and Q4 for each year. *Month* has members named 1-12 for each year.

```
DESCENDANTS ([Time].[Year].[1998], [Month])
returns: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

where the month numbers represent the months of 1998.

```
DESCENDANTS ([Time].[Year].[1998], [Month], BEFORE)
returns: 1998, Q1, Q2, Q3, Q4
```

where the quarters represent the quarters of 1998.

```
DESCENDANTS ([Time].[Year].[1998], [Quarter], SELF_AND_AFTER)
returns: Q1, 1, 2, 3, Q2, 4, 5, 6, Q3, 7, 8, 9, Q4, 10, 11, 12
```

where each quarter and month number represents the quarters and months of 1998.

```
DESCENDANTS ([Time].[Year].[1998], [Quarter], BEFORE_AND_AFTER)
returns: 1998, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

where the month numbers represent the months of 1998.

GENERATE

The **GENERATE** function returns a set that is generated by recursively applying an expression over a specified set. The **GENERATE** function has the following syntax:

```
GENERATE (set, expression[, ALL])
```

where:

set is a set of tuples.

expression is an expression that evaluates to a set of members.

The **GENERATE** function removes duplicates from the returned set by default. If the optional argument (,**[ALL]**) is used, duplicates are retained.

For example, given a *Time* dimension with levels *Year* and *Quarter*, the following expression:

```
GENERATE({[1999], [2000]}, [Time].CURRENTMEMBER.CHILDREN)
```

returns a list of the children of 1999 and of 2000: Q1, Q2, Q3, Q4, Q1, Q2, Q3, Q4.

MEMBERS

The MEMBERS function returns either all of the members within a specified dimension or the members of a specified level. The MEMBERS function has the following syntax:

dimension.MEMBERS

or

level.MEMBERS

where:

dimension is an expression that evaluates to a dimension defined in the Alphablox cube.

level is an expression that evaluates to a level defined in the Alphablox cube. To avoid ambiguity, the level should be fully qualified.

When the MEMBERS function operates on a dimension, it returns every member of the dimension at every level defined in the dimension.

When the MEMBERS function operates on a level, it returns every member of the specified level only. It does not return members of levels below the specified level.

Examples

For the following examples, assume there is a dimension named *Time* with levels *Year* and *Quarter*. *Year* has members named 1999, 2000, and 2001. *Quarter* has members named Q1, Q2, Q3, and Q4.

Example	Evaluates To	Explanation
[Time].MEMBERS	1999, Q1, Q2, Q3, Q4, 2000, Q1, Q2, Q3, Q4, 2001, Q1, Q2, Q3, Q4,	The members of the <i>Time</i> dimension include all of the members of all of its levels and the children of each member.

Example	Evaluates To	Explanation
[Time].[Year].MEMBERS	1999, 2000, 2001	The expression [Time].[Year] evaluates to the level named <i>Year</i> ; the MEMBERS function, when operating on a level, returns only the members of the level.

Example MDX Queries

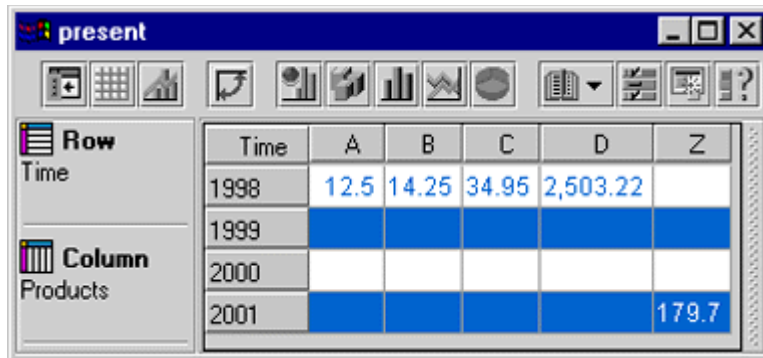
This section shows some examples of MDX queries against an Alphablox cube named *AlphabloxCube*. Assume the Alphablox cube in the examples has the following dimensions, levels, and measures.

Time	Products	Measures
Year { 1998, 1999, 2000, 2001 }	Imported { Yes, No }	{ Sales, Cost, Profit }
Quarter { Q1, Q2, Q3, Q4 }	Product Name { A-Z }	
Month { 1-12 }		

Example 1

The following query selects several members (*A*, *B*, *C*, *D*, and *Z*) from the *Product Name* level on the columns axis, uses the *CHILDREN* function on the *Time* dimension for the rows axis to generate a set of years, and slices the query by the *Sales* measure in the *WHERE* clause.

```
SELECT {[Products].[Product Name].[A]:[D],
       [Products].[Product Name].[Z]} ON COLUMNS,
       {[Time].CHILDREN} ON ROWS
FROM [AlphabloxCube]
WHERE ([Sales])
```



Example 2

The following query uses the CROSSJOIN function to show both the product members E and F and the 4 quarters from 1999 on the columns axis. The rows axis shows the three measures in the Alphablox cube.

```
SELECT CROSSJOIN ({[Products].[Product Name].[E],
                  [Products].[Product Name].[F]}, [Time].[1999].CHILDREN)
ON COLUMNS,
   {[Sales], [Cost], [Profit]} ON ROWS
FROM [AlphabloxCube]
```

		E				F			
		Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Measures	Sales	17,700	16,800	44	18,100	1,413.87	1,413.87	5,510	1,413.87
	Cost	12,300	12,300	50	13,200	599.97	599.97	4,400	599.97
	Profit	5,400	4,500	-6	4,900	813.9	813.9	1,110	813.9

Index

A

- AAS Cube, *see* Alphablox Cube
- access control lists
 - Alphablox Cubes, using with 42
- Alphablox Cube
 - administration strategy 60
 - applications of 21
 - cache 25, 65
 - Console commands 62
 - creating, checklist for 40
 - data source, relational, creating 42
 - defining 44
 - dimensions and levels, define 46
 - MDX, supported syntax 74
 - measures, defining 45
 - memory considerations 68
 - modifying 64
 - overview 20
 - rebuilding 59
 - refreshing 51
 - relational schema, mapping to cube 36
 - resources, specify and manage 51
 - sanity check 53
 - SQL editing 49
 - starting 56
 - stopping 58
 - troubleshooting 57
 - tuning controls 65
- Alphablox cube
 - requirements 27
- Alphablox Cube Server 27
 - architecture 24
 - requirements 27
- architecture
 - Alphablox Cube Server 24

C

- cache seeding MDX query 44
- cache, cube
 - in architecture 25
 - maximum rows 66
- CHILDREN function 76
- clean data
 - defined 27
- columns and rows, maximum, setting for a cube 67
- Console
 - command list, cube 62
 - creating a cube, checklist 40
- CROSSJOIN function 77
- Cube Manager 25
- cube, Alphablox, *see* Alphablox cube
- CURRENTMEMBER function 78

D

- data sources
 - cube adapter, creating 50
 - maximum connections for a cube 65
 - relational, creating for a cube 42
- define the cube 44
- DELETE CUBE command 62
- denormalized, defined 34
- DESCENDANTS function 78
- dimension tables 31
- dimensional schemas
 - described 30
 - hierarchies 33
 - requirements for Alphablox Cube Server 27
 - snowflake 31
 - star 31
- dimensions, cube, define 46
- DISABLE CUBE command 62

E

edit SQL, cube 49
 EMPTYCACHE CUBE command 62
 ENABLE CUBE command 62

F

fact tables 31
 foreign key
 defined 31
 defining in a level 48

G

GENERATE function 80

H

heap size, memory, changing 68
 hierarchies
 relational database schema 33

K

keys, foreign, *see* foreign keys
 keys, primary, *see* primary keys

L

levels, cube, define 46

M

many-to-one relationships 34
 maximum connections, cube 65
 maximum data source connections, cube 65
 maximum number of cubes 67
 maximum rows and columns, cube 67

MDX

basic syntax 74
 CHILDREN function 76
 CROSSJOIN function 77
 CURRENTMEMBER function 78
 DESCENDANTS function 78
 example queries 83
 FROM TO syntax 75
 GENERATE function 80
 member sets 75
 MEMBERS function 81
 SQL queries, relationship to 26
 measures, cube, define 45
 measures, cube, restrictions 37
 member sets, specifying 75
 MEMBERS function 81
 memory considerations, cube 68
 memory heap size, changing size 68

N

Normalized button, level dialog box 36
 normalized, defined 34

P

primary key
 defined 31
 defining in a level 48

R

REBUILD command 59
 REBUILD CUBE command 63
 rebuilding a cube 59
 refreshing a cube 51
 relational data
 cubing 21
 database schemas 27, 30
 dimensional schemas 30
 mapping schema to a cube 36
 measures expression restriction 37
 schema requirements 27
 requirements
 cube 27
 rows and columns, maximum, setting for a cube 67

S

SHOW CUBE command 63

snowflake schema 31

SQL, edit for cube 49

star schema 31

START CUBE command 56, 63

starting a cube 56

- from Alphablox Analytics Home Page 56

- from console 56

- troubleshooting 57

STOP CUBE command 58, 63

T

tables

- dimension 31

- fact 31

