IBM DB2 Alphablox

# DB2 Alphablox Cube Server Administrator's Guide

*Version 8.3*

SC18-9433-01

IBM DB2 Alphablox

# DB2 Alphablox Cube Server Administrator's Guide

*Version 8.3*

SC18-9433-01

**Note:**
Before using this information and the product it supports, read the information in "Notices" on page 39.

**Second Edition (November 2005)**

This edition applies to version 8, release 3, of IBM DB2 Alphablox for Linux, UNIX and Windows (product number 5724-L14) and to all subsequent releases and modifications until otherwise indicated in new editions.

# Contents

# Chapter 1. Cubing Concepts

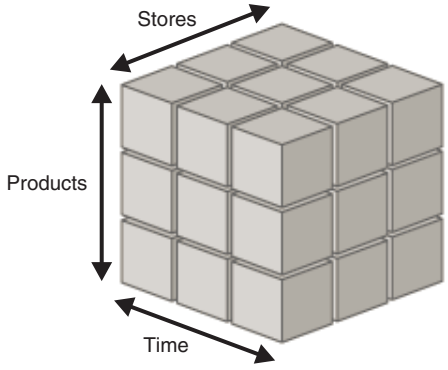
IBM DB2 Alphablox for Linux, UNIX and Windows includes the DB2 Alphablox Cube Server. The DB2 Alphablox Cube Server is designed to provide a multidimensional view of data stored in a relational database. This chapter introduces the DB2 Alphablox Cube Server, provides background into the types of applications it is designed for, and describes the requirements for its use.



## Overview

DB2 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 three-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, 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 Chapter 2, "Dimensional Schema Design," on page 7.

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 a DB2 Alphablox cube on top of relational data; that is, the DB2 Alphablox cube is populated using queries to the underlying RDBMS.

## Applications of the DB2 Alphablox Cube Server

The DB2 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 IBM DB2 OLAP Server, Hyperion Essbase, or Microsoft Analysis Services. A DB2 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 creating multidimensional data sources for which you do not have the time and resources to develop 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 DB2 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, including potentially billions of rows in the fact table (for a definition of a fact table, see "Fact Tables" on page 8). DB2 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 DB2 Alphablox applications.

### Prototyping

A DB2 Alphablox cube can be created very rapidly and can therefore allow applications to access and gain value out of real data very quickly. DB2 Alphablox applications that access DB2 Alphablox cubes can be easily modified to access data that resides in DB2® Cube Views, a DB2 OLAP Server cube, a Hyperion Essbase cube or in a Microsoft Analysis Services cube. Therefore, DB2 Alphablox cubes

provides an excellent platform for prototyping larger scale applications during a development cycle. In some cases, keeping the data in DB2 Alphablox cubes might prove 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

A DB2 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. Many complex business scenarios, however, do not require multiple hierarchies per dimension.

**Note:** 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 a DB2 Alphablox cube are defined with a valid SQL expression to the underlying database. 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 10 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 DB2 Alphablox Cube Server

Because the DB2 Alphablox Cube Server is part of DB2 Alphablox and there is 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:

- DB2 Alphablox Cube Server has no disk space to manage.
- DB2 Alphablox Cube Server uses the DB2 Alphablox security model, requiring no additional work to manage users.
- DB2 Alphablox Cube Server is included with DB2 Alphablox, requiring no additional software to install.

## DB2 Alphablox Cube Server in an DB2 Alphablox Application Environment

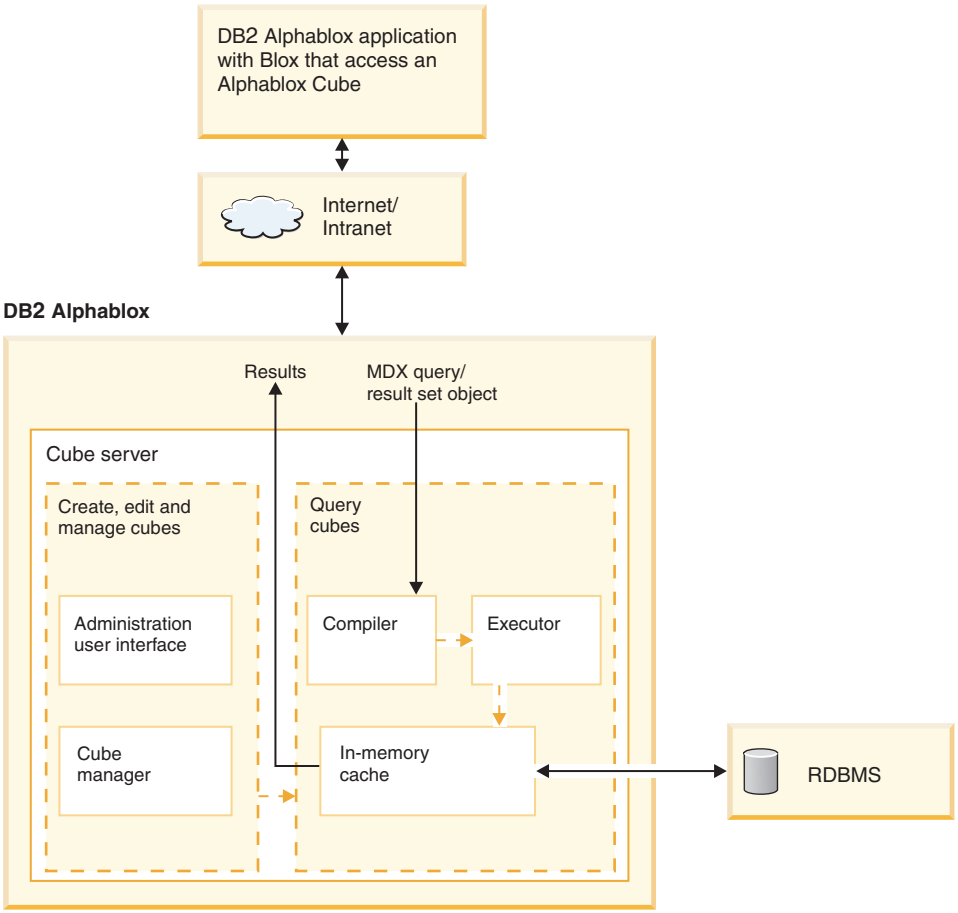
For a DB2 Alphablox application, a DB2 Alphablox cube is just another data source; that is, the Blox functionality works with a DB2 Alphablox cube just like it does with any other data source. DB2 Alphablox cubes use the same Blox as other data sources. For example, you can change an application that accesses a DB2 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 DB2 Alphablox cubes.

# DB2 Alphablox Cube Server Architecture

The DB2 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.

## DB2 Alphablox Cube Server Components

The DB2 Alphablox Cube Server is composed of several components. These complementary components provide the infrastructure to define, manage, and execute queries against DB2 Alphablox cubes. The components of the DB2 Alphablox Cube Server work within the framework of DB2 Alphablox, as shown in the following figure.




### Administration User Interface

A cube administrator performs the tasks for setting up and managing DB2 Alphablox cubes through the administration interface of DB2 Alphablox. The **Cubes** link under the **Administration** tab of the DB2 Alphablox home page is devoted to cube setup and administration; this is where dimensions, levels, and measures for a cube are defined. An DB2 Alphablox user must be a member of the administrators group in order to create, view, or modify DB2 Alphablox cubes. For detailed information on using the DB2 Alphablox cubes administration user interface, see Chapter 3, “Creating and Modifying a Cube,” on page 13 and Chapter 4, “Maintaining a Cube,” on page 23.

### Cube Manager

The Cube Manager is the component that creates objects, performs verification checks, starts and stops, and performs other work on DB2 Alphablox cubes. The DB2 Alphablox 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 27.

### 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 DB2 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 a DB2 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 DB2 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

DB2 Alphablox applications request results from a cube through MDX queries. DB2 Alphablox Cube Server processes the MDX query which results in a plan for retrieving results from a DB2 Alphablox cube. The DB2 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 DB2 Alphablox application issues the MDX query, DB2 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 a DB2 Alphablox cube references. A DB2 Alphablox cube must reference a supported relational database. The *Installation Guide* describes the databases supported by DB2 Alphablox. 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 a DB2 Alphablox cube must contain only one fact table; multiple fact table schemas are not supported. Each dimension in a DB2 Alphablox cube must have a single hierarchy. For more information about schemas, see "Dimensional Schemas" on page 7.

**Note:** 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 a DB2 Alphablox cube.

# Chapter 2. Dimensional Schema Design

DB2 Alphablox Cube Server requires that the underlying databases have a dimensional schema. To set up a DB2 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.



## 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 SQL language, used by most RDBMS vendors.

A dimensional schema physically separates the measures (also called *facts*) that quantify the business from the descriptive elements (also called *dimensions*) that describe and categorize the business. DB2 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.

**Note:** 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 a DB2 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:



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 10.

## Fact Tables

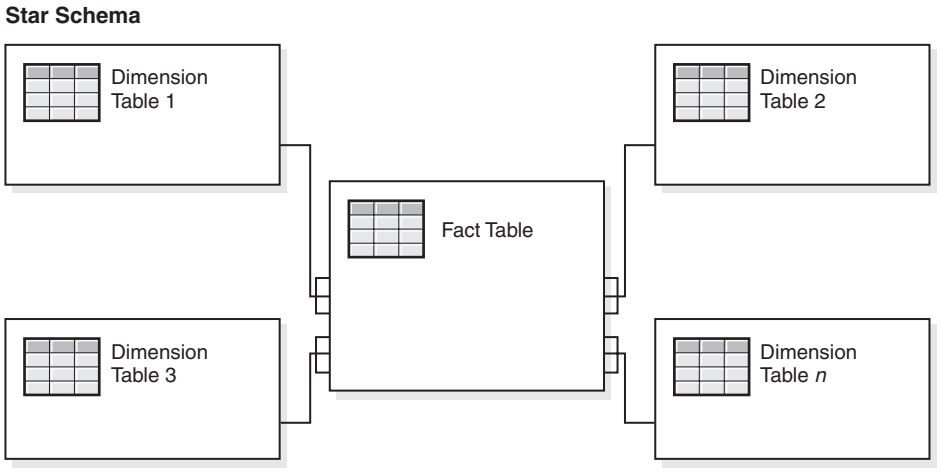
A *fact table* is a table in a star or snowflake schema that stores facts that measure the business, such as sales, cost of goods, or profit. Fact tables also contain foreign keys to the dimension tables. These foreign keys relate each row of data in the fact table to its corresponding dimensions and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

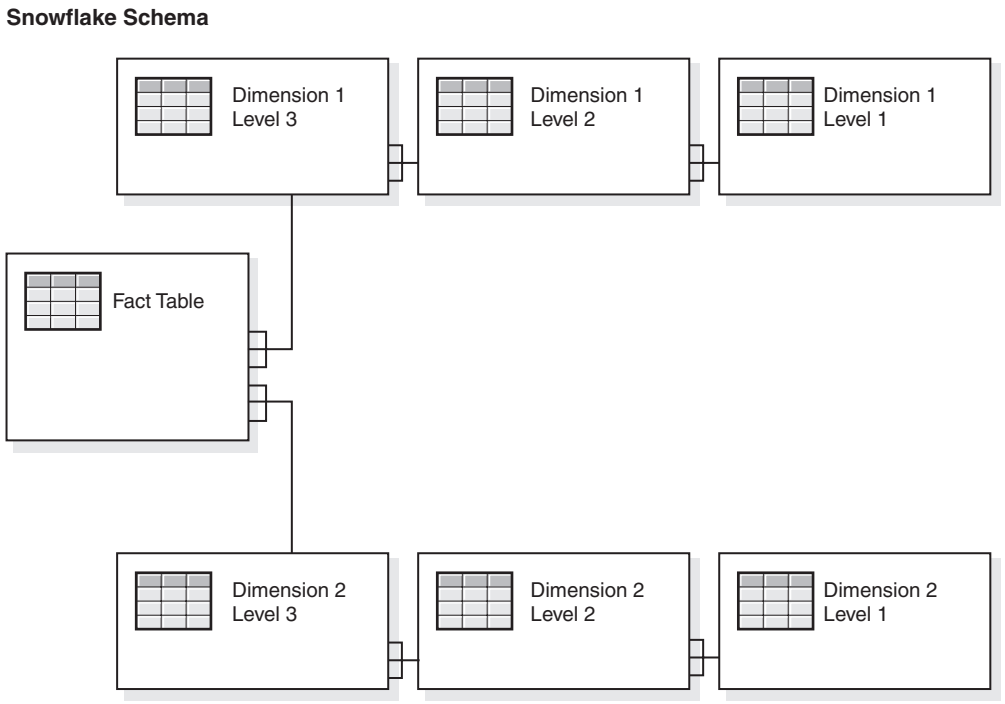
## Star Schemas

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 9.

## 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 DB2 Alphablox cubes out of the relational data.

When you define a DB2 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 a DB2 Alphablox cube, see Chapter 3, "Creating and Modifying a Cube," on page 13.

## Mapping the Relational Schema to a Cube

It is important for the administrator who designs and builds a DB2 Alphablox cube to understand, at least at a high level, the mapping between the relational database and the DB2 Alphablox cube. Understanding this mapping helps to ensure there are no errors in the design or creation of the DB2 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, Levels, and Attributes

You can define any number of dimensions in a DB2 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 DB2 Alphablox Cube Server relies on the relationships among these different tables to create dimensions in the cube. When you define a DB2 Alphablox cube, you must provide details about the schema as part of the DB2 Alphablox cube definition.

**Note:** 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 DB2 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.

# Chapter 3. Creating and Modifying a Cube

An administrator uses the **Cubes** section of the **Administration** tab to define DB2 Alphablox cubes. This chapter describes the steps necessary to create a DB2 Alphablox cube.

## Checklist of Tasks to Create a Cube

This section provides a checklist of tasks needed to define a DB2 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 a DB2 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 Chapter 2, "Dimensional Schema Design," on page 7. |
| 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 a DB2 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 14. | Create an DB2 Alphablox data source definition for the underlying relational database from which the DB2 Alphablox cube is created. |
| 4 | "Define the Cube" on page 15. | Use the **Cubes** administration user interface to define the properties of a DB2 Alphablox cube. |
| 5 | "Define the Measures" on page 16. | Specify what facts are measured in the DB2 Alphablox cube and the mapping for each measure from the relational fact table to the DB2 Alphablox cube. |
| 6 | "Define the Dimensions" on page 17. | Specify each dimension in the DB2 Alphablox cube and each level for each dimension. Define the mapping between the relational tables and the DB2 Alphablox cube dimensions and levels. |
| 7 | "Create Alphablox Cube Server Adapter Data Source" on page 19. | To query a DB2 Alphablox cube, define a data source created with the **Alphablox Cube Adapter** multidimensional driver. |
| 8 | "Specify and Manage Cube Resources" on page 20. | Enter the DB2 Alphablox cube connection limits, update frequency, and other administrative parameters. |
| 9 | "Review the Cube" on page 21. | Make sure there are no errors in the information you entered to define the DB2 Alphablox cube. |

## Create the Relational Data Source

A DB2 Alphablox cube requires that its underlying relational data source be defined in an DB2 Alphablox data source. Each DB2 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 a DB2 Alphablox cube, see "Schema Requirements" on page 5. For a discussion of dimensional schemas, see Chapter 2, "Dimensional Schema Design," on page 7.

If you have already defined a data source for the relational database, skip to the next topic, "Define the Cube" on page 15. For more information about data sources, see the *Administrator's Guide*.

To specify a relational database as an DB2 Alphablox data source, perform the following steps:

1. Log into the DB2 Alphablox 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-down list, select one of the Relational Driver options (for example, `IBM DB2 JDBC Type 4 Driver`).
6. Enter a name for your new data source in the **Data Source Name** text box.
7. Enter the appropriate information for **Client Host Name**, **Port Number**, **SID**, and **Database** fields (available fields depend the driver selected). 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 Username** and **Default Password**. The username and password must be valid on the relational database. The default username and password are always used when a DB2 Alphablox cube accesses a relational database. The specified database user requires read access to the database.

   **Note:** The value of the **Use DB2 Alphablox Username and Password** drop-down list is ignored when the data source is being used to populate a DB2 Alphablox cube. Use access control lists (ACLs) to control user access to DB2 Alphablox cubes. For information about ACLs, see the *Administrator's Guide*.
9. The **Maximum Rows** and **Maximum Columns** text boxes are ignored when the data source is being used to populate a DB2 Alphablox cube. You can still enter values and they will be used when other applications use the data source, but a DB2 Alphablox cube ignores these text boxes.
10. Set the **JDBC Tracing Enabled** drop-down list to **No** unless you want to write JDBC logging information to the DB2 Alphablox 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 a DB2 Alphablox cube as follows:

1. Log into the DB2 Alphablox 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 **Cubes** link.
4. Click the **Create** button. A **Cube Administration** dialog will appear in a new web page window.
5. In the **DB2 Alphablox Cube Name** text box, enter a unique name for the DB2 Alphablox cube. Allowable characters for DB2 Alphablox cube names are A-Z, a-z, 0-9, underscore (_), and space.
6. Check the **Enabled** checkbox to the right of the **DB2 Alphablox Cube Name** text box if you want the cube to start automatically whenever the server restarts. If you are working on the cube definition and do not expect it to run

properly or you do not want to give others access to it yet, you can leave the checkbox unchecked and enable the cube later.

7. Using the **Relational Data Source** drop-down list, select a relational data source previously defined in "Create the Relational Data Source" on page 14. If DB2 Alphablox relational data sources have not been defined, the list is blank.
8. Using the Security Role drop-down list, you can pick a role (predefined in the application server or in DB2 Alphablox) that can restrict users of this cube to the selected role. To enable the use of the selected security role, the Enabled checkbox must be checked.
9. Optionally, if you are using IBM DB2 UDB as a data source and have DB2 Cube Views cubes available on this data source, the **Enable DB2 Cube Views Settings** option will become available. By selecting this option, you can use the available cube definitions in DB2 Cube Views to specify your DB2 Alphablox cube. To use this option, perform the following sub-steps:
   a. Using the **Cube Model** drop-down list, select a cube model.
   b. Using the **Cube** drop-down list, select a cube.
   c. Selecting either the **Use Business Names** or **Use Object Names** radio button to specify the names to be used to define objects in your DB2 Alphablox cube.
   d. Click the **Import Cube Definition** button to import a cube definition and pre-populate measures and dimensions in your DB2 Alphablox cube. Depending on the cube definition imported, DB2 Alphablox Cube Server attempts to specify a DB2 Alphablox cube that closely matches the one in DB2 Cube Views. Click on the **Show Import Log** button to see a log specifying information and debugging messages related to the import operation.
   e. At this point, you can edit the imported cube measures and dimensions (as described below) to customize your cube, or you can optionally check the **Import cube definition (on start, rebuild, and edit)** option. Selecting this option will result in your DB2 Alphablox cube loading the latest DB2 Cube Views cube definition each time your DB2 Alphablox cube is started, rebuilt, or opened for edit.
10. Click the **Save** button to save the DB2 Alphablox cube.

## Define the Measures

All DB2 Alphablox cubes must have one or more measures defined. For a description of measures, see "Measures" on page 10. To define measures in a DB2 Alphablox cube, perform the following steps:

1. Log into the DB2 Alphablox 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 **Cubes** link.
4. Select the DB2 Alphablox cube from the list of cubes and click the **Edit** button. The DB2 Alphablox **Cube Administration** dialog for the selected cube will appear in a new web page window.
5. In the cube navigation tree, click on the **Measures** node. A measures panel appears.
6. In the **Measures Fact Table** text box, you must enter the fully-qualified name of the fact table as it is defined in the underlying relational database (for

example, CVSAMPLE.SALESFACT). Alternatively, select the correct schema, catalog, and table combination in the drop-down lists to automatically insert the fact table name.

7. After the fact table has been specified, you can create a new measure by clicking the **Create New Measure** button. A new set of options appears.
8. In the **Name** text box, replace "New Measure" with a name for your new 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 DB2 Alphablox 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*.
9. In the **Expression** text box, enter a valid expression. The Expression Builder tool can be used to help in entering the correct syntax for columns and functions. There are shortcut buttons available for frequently used functions (AVG, COUNT, MAX, MIN, and SUM), but you can manually enter any valid function you need. These functions are used in generating the SQL sent to the underlying database to calculate the new measure. The following expression example defines the COGS measure:

   ```
   SUM(@col(CVSAMPLE.SALESFACT.COGS))
   ```
10. Click the **Apply** button to add the measure to the list.
11. Repeat these steps as needed to define any other measures you need. To delete a measure, click on the measure label in the navigation tree, then click the **Delete Selected** button below the tree.
12. Click the **OK** button if you are finished modifying your DB2 Alphablox cube definition, or proceed on to define the dimensions and levels.

    **Note:** Any changes made to a started DB2 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 23.

## Define the Dimensions

You must enter information to define the dimensions, levels, joins, attributes, and other information for the DB2 Alphablox cube.

For a description of dimensions and levels, see "Dimensions, Levels, and Attributes" on page 10.

### Create or Edit Dimensions

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

1. Log into the DB2 Alphablox 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 **Cubes** link.
4. Select a DB2 Alphablox cube from the list of cubes and click the **Edit** button. The DB2 Alphablox **Cube Administration** dialog for the selected cube will appear in a new web page window.

5. In the DB2 Alphablox cube tree on the left, click on the **Dimensions** label. In the right panel the Create Dimension button will appear. To edit an existing dimension, click on the dimension name and the existing dimension definition will appear.
6. Click the **Create New Dimension** button to create a new dimension or select a dimension from the **Dimensions** list 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. After you have named your new dimension, you can define any required fact table joins and dimension joins.
10. Click the **OK** button to save the dimension.

## Create or Edit a Fact Table Join

For each dimension that you create, you need to define a fact table join. To create or edit a fact table join in a dimension, perform the following steps:

1. After creating a new dimension using the Cube Administration dialog, click on the **Fact Table Join** node under the new dimension.
2. To create a fact table join (if one does not exist yet), click on the **Create New Join** button that appears. A join specification panel appears. If a fact table join already exists, expand the Fact Table Join folder and click on the join.
3. In the **Expression** text box, enter an expression that specifies the fact table join. You can also use the Expression Builder to assist you in entering an expression defining the join. Example:

   ```
   @col(MDSAMPLE.MARKET.STATEID) = @col(MDSAMPLE.SALESFACT.STATEID)
   ```

4. Click the **Apply** button to apply and save these settings without closing the dialog. Click the **OK** button to save the level definition.

## Create or Edit a Dimension Join

For each dimension that you create, you can also create dimension joins between relevant tables. To create or edit a dimension join in the selected dimension, perform the following steps:

1. After creating a new dimension using the Cube Administration dialog, click on the **Dimension Joins** node under Joins folder of the new dimension.
2. To create a new dimension join, click on the **Create New Join** button that appears. A dimension join dialog appears. To edit an existing dimension join, expand the Dimension Joins folder and select the join you want to edit.
3. In the **Expression** text box, enter an expression that specifies the dimension join. You can also use the Expression Builder to assist you in entering an expression defining the join. Example:

   ```
   @col(MDSAMPLE.PRODUCT.FAMILYID) = @col(MDSAMPLE.FAMILY.FAMILYID)
   ```

4. Click the **Apply** button to apply and save these settings without closing the Cube Administration dialog. Click the **OK** button to save your changes and close the Cube Administration dialog.

## Create or Edit Levels

For each dimension, you can specify levels in a dimensional hierarchy. To create or edit levels, perform the following steps:

1. To create a new level, click on the **Levels** node under that dimension and click the **Create New Level** button. To edit an existing level, open the Levels folder and select the level you want to edit.
2. For a new level, specify a name in the **Name** text box. Allowable characters for DB2 Alphablox cube names are A-Z, a-z, 0-9, underscore (_), and space.
3. In the **Expression** text box, enter an expression that specifies the level. You can also use the Expression Builder to assist you in entering an expression defining the level. Example:

   ```
   @col(MDSAMPLE.TIME.QUARTER)
   ```

4. Click the **Apply** button to apply and save these settings without closing the Cube Administration dialog. Click the **OK** button to save your changes and close the Cube Administration dialog.

### Create or Edit Attributes

Attributes represent additional database table columns that belong to a level. Attributes are specified by a SQL expression that can either be a simple mapping to a single table column or a more complex expression combining multiple columns or attributes and using SQL functions. To create or edit attributes, perform the following steps:

1. To create a new attribute, click on the **Attributes** node that appears under your dimension, then click the **Create New Attribute** button. An attribute definition dialog appears. To edit an existing attribute, click on the attribute node that you want to edit.
2. In the **Expression** text box, enter an expression that specifies your attribute. You can also use the Expression Builder to assist you in entering an expression defining the attribute. Example:

   ```
   @col(FAMILY.FAMILYID)
   ```

3. Click the **Apply** button to apply and save these settings without closing the Cube Administration dialog. Click the **OK** button to save your changes and close the Cube Administration dialog.

## Create Alphablox Cube Server Adapter Data Source

To query a DB2 Alphablox cube, a DB2 Alphablox data source that uses the **Alphablox Cube Server Adapter** must be defined. A single data source can be used to access multiple DB2 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 a DB2 Alphablox Cube Server Adapter data source, perform the following steps.

1. Log into the DB2 Alphablox 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-down 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 DB2 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 a DB2 Alphablox cube changes, any data cached in a DB2 Alphablox cube might be stale. When the data becomes stale, you should rebuild the cube to guarantee that answers derived from the DB2 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 DB2 Alphablox cube or using the `REBUILD CUBE <cube_name>` console command. Alternatively, 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 DB2 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 DB2 Alphablox cube every morning at 3:00 AM.

To configure a DB2 Alphablox cube to rebuild itself at regular intervals, perform the following steps:

1. Log into the DB2 Alphablox 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 **Cubes** link.
4. Select a DB2 Alphablox cube from the list of cubes and click the **Edit** button. The DB2 Alphablox **Cube Administration** dialog for the selected cube will appear in a new web page window.
5. In the cube navigation tree on the left, click on the **Schedule** label. A scheduling panel appears.
6. Check the **Refresh every** box to enable scheduled DB2 Alphablox cube rebuilding.
7. Set the refresh interval by clicking the desired buttons and modifying the corresponding time periods. For example, to set the DB2 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 DB2 Alphablox cube definition.

## Setting Tuning Parameters

For each DB2 Alphablox cube, you can set several tuning parameters for resource management. To do so, perform the following steps:

1. Log into the DB2 Alphablox 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 **Cubes** link.

4. Select a DB2 Alphablox cube from the list of cubes and click the **Edit** button. The DB2 Alphablox **Cube Administration** dialog for the selected cube will appear in a new web page window.
5. In the cube navigation tree on the left, click on the **Tuning** node to open the tuning panel.
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 28.

| Tuning Parameter | Description |
|---|---|
| Maximum Connections | The maximum number of concurrent connections to this DB2 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 is 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. Optionally, you can enter an MDX query in the **MDX Query to preload performance cache** text box and check the **Enabled** checkbox.

   The query entered in this text box executes when the cube is started or rebuilt. This query will populate the DB2 Alphablox Cube Server in-memory cache with an initial set of results. These results *seed* the cache with data retrieved from the underlying database. Any subsequent DB2 Alphablox cube queries requiring only data that is already in the DB2 Alphablox Cube Server cache are answered directly from the cache, thus improving response time by avoiding additional SQL queries to the underlying database. The name of the cube referenced in the FROM clause of the MDX query must be the name previously defined in the **DB2 Alphablox Cube Name** text box.
8. Click the **Save** button to update the DB2 Alphablox cube definition.

## Review the Cube

It is usually worthwhile to take a few minutes after you have created a DB2 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 review on a DB2 Alphablox cube, perform the following steps:

1. Log into the DB2 Alphablox 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 **Cubes** link.
4. Select the DB2 Alphablox cube from the list of cubes and click the **Edit** button. A page showing the Edit DB2 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 DB2 Alphablox cube, verify that **Enabled** is selected next to the **Alphablox Cube Name** text box. If it is not enabled, you will see an error message when attempting to start the cube.
7. Verify that your measures are properly created:
   a. Click on the **Measures** node to verify that the table specified in the **Measures Fact Table** text box is the correct table in the relational schema, the name is spelled correctly, and the name is a fully-qualified name.
   b. For each defined measure, check that the desired aggregation is specified in the **Expression** text box.
8. Verify that all the desired dimensions have been correctly defined, and that the names are correct. For each dimension, check the following:
   a. Verify that you had added any required fact table join and dimension joins, and that the expressions are correct.
   b. Verify that the levels are correctly specified and appear in the correct order. The first level 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, followed by *Month* , followed by *Day* .
   c. Verify that any attributes that you have defined are correct, including the expected names and expressions.
9. Click the **Schedule** tab and verify that all the settings are the way you want them.
10. Click the **Tuning** tab and verify that all the settings are the way you want them.

After completing this review of your DB2 Alphablox cube, you can start the cube. For details on starting a DB2 Alphablox cube, see "Starting, Stopping, and Rebuilding a Cube" on page 23.

# Chapter 4. Maintaining a Cube

DB2 Alphablox Cube Server provides functionality to perform administrative tasks on DB2 Alphablox cubes. These tasks are performed either through the DB2 Alphablox administration user interface or through the Console. This chapter describes these tasks.



## Starting, Stopping, and Rebuilding a Cube

The most common administrative tasks you need to perform on a DB2 Alphablox cube are to start, stop, and rebuild the cube.

### Starting a DB2 Alphablox Cube

You must start a DB2 Alphablox cube to make it available for querying. You can start a cube either from the DB2 Alphablox 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. A DB2 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 DB2 Alphablox cube to populate the cache with the measure values returned from the MDX query.

#### Start Cube From the Home Page

To start a DB2 Alphablox cube from the DB2 Alphablox Home Page, perform the following steps:

1. Log into the DB2 Alphablox Home Page as the *admin* user or as a user who is a member of the administrators group.
2. Click the **Administration** tab. The **General** page appears.
3. Under the **Runtime Management** section, click the **Cubes** link.
4. From the **DB2 Alphablox Cubes** list, select the DB2 Alphablox cube to start.
5. To view the current status of the DB2 Alphablox cube, click the **Details** button.
6. Click the **Start** button. When the DB2 Alphablox cube has completed the startup operation, the status field displays **Running**.

#### Start Cube From a Console Window

To start a DB2 Alphablox cube from a console window, perform the following.

1. If DB2 Alphablox is not already running, start it. See *Administrator's Guide* for details about starting DB2 Alphablox.
2. In a Console window, enter the following command:

   ```
   start cube cube_name
   ```

where *cube_name* is the name of the DB2 Alphablox cube to start. When using the DB2 Alphablox Admin Pages in a web browser, you can also open a console window by clicking on Administration -> General -> Start Console Session.

### Troubleshooting If the Cube Does Not Start

If the DB2 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 DB2 Alphablox 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 DB2 Alphablox relational data source.

For information about enabling any of these logging options, see the *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 DB2 Alphablox cube again.

| Error | Description |
|---|---|
| "Make sure the cube is enabled." | Check to see if your cube is enabled. On the command line, enter the following to see if the cube is enabled:<br>`show cube cube_name`<br><br>To enable a DB2 Alphablox cube, from the **General** tab in the **Cubes** user interface, select **Enabled** next to the **DB2 Alphablox Cube Name** text box. |
| An error connecting to the underlying database. | Connection errors can be caused by a variety of problems. The following are some common things to check:<br>• Check that the relational data source has the correct connect information.<br>• Check that the relational data source has a valid, non-null username and password.<br>• Make sure the database is available for connections. |
| 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 a DB2 Alphablox Cube

Stopping a DB2 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 a DB2 Alphablox cube through the DB2 Alphablox Home Page, perform the following steps:

1. Log into the DB2 Alphablox Home Page as the *admin* user or as a user who is a member of the administrators group.
2. Click the **Administration** tab. The **General** page appears.

3. Under the **Runtime Management** section, click the **DB2 Alphablox Cubes** link.
4. Select the Alphablox cube you want to stop from the **DB2 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 a DB2 Alphablox cube from a Console window, enter the following command:

```
stop cube cube_name
```

where *cube_name* is the name of the DB2 Alphablox cube to stop. When using the DB2 Alphablox Admin Pages in a web browser, you can also open a console window by clicking on Administration -> General -> Start Console Session.

**Note:** The DB2 Alphablox cube will not stop until any executing queries have completed.

## Rebuilding a DB2 Alphablox Cube

You should either rebuild or restart a DB2 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 a DB2 Alphablox cube enter the following command from a Console window:

```
rebuild cube cube_name
```

where *cube_name* is the name of the DB2 Alphablox cube to rebuild. When using the DB2 Alphablox Admin Pages in a web browser, you can also open a console window by clicking on Administration -> General -> Start Console Session.

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 a DB2 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 DB2 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 "DB2 Alphablox Cube Memory Considerations" on page 30.

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 a DB2 Alphablox cube.

## Understanding the Database Environment

Every time data changes in the database underlying a DB2 Alphablox cube, parts of the cube potentially become out of date. A DB2 Alphablox cube gets its data from queries to the underlying database. When a query requests data from a DB2 Alphablox cube, DB2 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 DB2 Alphablox cube were inserted, updated, or deleted from the database, the results from the DB2 Alphablox cube would not reflect the true state of the dimensions. The results from a new query to the DB2 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 DB2 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 DB2 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 DB2 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 DB2 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 DB2 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 DB2 Alphablox cube. For details on setting up an automatic schedule, see "Defining a Refresh Schedule" on page 20.

There is no one best way to schedule updates to a DB2 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 a DB2 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 DB2 Alphablox console window. To access the console, click the **Administration** tab, **General** page, **Start Console Session**link, or use the DB2 Alphablox Console window that opens when you start DB2 Alphablox. The following table lists the cube commands and a description of what each does.

| Command Syntax | Description |
|---|---|
| **delete cube** <*cube_name*> | Deletes a cube and its entire definition. |
| **disable cube** <*cube_name*> | Sets a cube in the disabled state. A disabled cube cannot be started until it is enabled and therefore does not automatically start when DB2 Alphablox starts. A cube should be stopped before it is disabled. |
| **emptycache cube** <*cube_name*> | Removes all entries from the cube's in-memory cache. After emptying the cache, a query against the DB2 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 DB2 Alphablox cube are equivalent to the data stored in the database. Note that the `EMPTYCACHE` command does not rebuild the dimension outlines of the cube. To rebuild the dimension outlines, use the REBUILD command or stop and start the cube. |
| **enable cube** <*cube_name*> | Sets a cube to the enabled state. A cube must be enabled before it can be started. Enabled cubes start automatically when DB2 Alphablox starts. |
| **rebuild cube** <*cube_name*> | 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. |
| **show cube** <*cube_name*> | Shows the current status of the cube. The cube status can be:<br>• disabled<br>• stopped<br>• starting<br>• running<br><br>To show the status of all defined DB2 Alphablox cubes, enter the following command:<br>`show cube` |

| | |
|---|---|
| **start cube** <*cube_name*> | 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). |
| **stop cube** <*cube_name*> | 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 DB2 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 **Cubes** administration page to modify a DB2 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 Chapter 3, "Creating and Modifying a Cube," on page 13.

## Tuning a cube

There are a number of administrative controls for tuning and configuring DB2 Alphablox cubes. Because DB2 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 DB2 Alphablox cubes.

#### Connection and cache size limits

You can specify connection and cache size limits for each defined DB2 Alphablox cube on the **Cubes** page, opening the Cube Administration dialog, and clicking on the **Tuning** label in the cube navigation tree.

**Maximum Connections:** When there are many users querying the DB2 Alphablox cube simultaneously, machine resources on the computer running DB2 Alphablox 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 DB2 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 "DB2 Alphablox Cube Memory Considerations" on page 30.

The more data that is stored in the cache, the less often queries to the DB2 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 DB2 Alphablox cube, perform the following steps:

1. Log into the DB2 Alphablox 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 **Cubes** link.
4. Select the DB2 Alphablox cube from the list of cubes and click the **Edit** button. The DB2 Alphablox **Cube Administration** dialog for the selected cube will appear in a new web page window.
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 DB2 Alphablox cube definition.

### Maximum Number of Cubes

If you have defined many DB2 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 DB2 Alphablox cubes allowed to run in DB2 Alphablox. The limit controls the number of DB2 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 DB2 Alphablox cubes, perform the following steps:

1. Log into the DB2 Alphablox Home Page as the *admin* user or as a user who is a member of the administrators group.
2. Click the **Administration** tab. The **General** page appears.
3. Under the **General Properties** section, click the **DB2 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 a DB2 Alphablox Cube Data Source, you can restrict applications from issuing queries that return large amounts of data. You set these limits in the DB2 Alphablox cube data source on the **Data Sources** administration page. The data source is the one used to issue MDX queries against a DB2 Alphablox cube.

## DB2 Alphablox Cube Memory Considerations

The DB2 Alphablox Cube Server runs as part of the Java™ process in which DB2 Alphablox runs. Therefore, as the Cube Server uses more memory, the Java process uses more memory. The memory limits for the DB2 Alphablox Java process are set at installation. If you find that the DB2 Alphablox is running out of memory due to DB2 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 28.
- Limit the number of Alphablox cubes in the system. For details, see "Maximum Number of Cubes" on page 30.
- Change the maximum size of the memory heap for the Java process in which DB2 Alphablox runs. For details, see "Changing the Maximum Memory Heap Size" below.
- Increase the memory capacity of the computer in which DB2 Alphablox runs. For details, see "Adding More Memory to Your System" on page 31.

### Changing the Maximum Memory Heap Size

The Cube Server runs as part of that Java process. If you are experiencing out-of-memory errors in DB2 Alphablox, you might need to raise the maximum memory heap size of the Java process. 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 DB2 Alphablox cubes, then the limits are set well for your environment.

## Adding More Memory to Your System

One often overlooked solution to memory issues is to add more memory to the system in which DB2 Alphablox 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.

# Chapter 5. Using MDX to Query DB2 Alphablox Cubes

DB2 Alphablox applications use the Multidimensional Expressions (MDX) language to query a DB2 Alphablox cube. MDX is the query language component of the OLE DB for OLAP specification, created and maintained by Microsoft. DB2 Alphablox cubes support a subset of the MDX syntax and functions. This section describes the supported MDX syntax for querying DB2 Alphablox cubes and provides example queries.

## Supported MDX Syntax

MDX is a multidimensional query language used by several multidimensional databases including Microsoft Analysis Services. DB2 Alphablox Cube Server uses a subset of the MDX syntax as the query language for DB2 Alphablox cubes. For a DB2 Alphablox application that accesses a DB2 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 a DB2 Alphablox cube is as follows:

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

where:

*axis_specification* — is a set of one or more tuples. Tuples can be entered as a list or "generated" with the CrossJoin function.

*cube_name* — is the name of a defined Alphablox cube.

*slicer_items* — 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 axis are referred to as AXIS(*n*), where *n* is the next consecutive integer. Note that DB2 Alphablox 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 om DB2 Alphablox 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 are

converted to uppercase before being sent to the server. Unless all of your member names are uppercased in the database, you should use the square bracket syntax

## 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]
```

**Note:** 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]}
```

## Functions

MDX functions are used to simplify and broaden the possible scope of MDX queries. The following table lists the subset of MDX functions supported in queries against DB2 Alphablox cubes.

For information on the syntax and usage of the MDX functions listed below, see the following information resources:

- Microsoft MDX Function Reference (http://msdn.microsoft.com/library/en-us/olapdmad/agmdxfunctintro_6n5f.asp)
- Spofford, George. 2001. *MDX Solutions*. New York: John Wiley & Sons.

| MDX Function | Syntax |
|---|---|
| Ancestor | `Ancestor(<Member>,<Level>)`<br><br>`Ancestor(<Member>,<Numeric Expression>)` |
| Ancestors | `Ancestors(<Member>,<Level>)`<br><br>`Ancestors(<Member>,<Numeric Expression>)` |

| MDX Function | Syntax |
|---|---|
| Ascendants | `Ascendants(<Member>)` |
| Avg | `Avg(<Set>[,<Numeric Expression>])` |
| BottomCount | `BottomCount(<Set>,<Count>[,<Numeric Expression>])` |
| BottomPercent | `BottomPercent(<Set>,<Percentage>[,<Numeric Expression>])`<br><br>**Note:** `<Numeric Expression>` is optional here, but is required in MSAS. |
| BottomSum | `BottomSum(<Set>,<Value>[,<Numeric Expression>])`<br><br>**Note:** `<Numeric Expression>` is optional here, but is required in MSAS. |
| Children | `<Member>.Children` |
| ClosingPeriod | `ClosingPeriod(<Level>,<Member>)`<br><br>**Note:** `<Level>` and `<Member>` are required here, but are optional in MSAS. |
| Count | `Count(<Set>[, ExcludeEmpty \| IncludeEmpty])`<br><br>**Note:** Only`Count(<Set>[, ExcludeEmpty \| IncludeEmpty])` is supported here. The `.Count` syntax is not supported here. |
| Cousin | `Cousin(<Member1>,<Member2>)` |
| CrossJoin | `Crossjoin(<Level>,<Member>)` |
| CurrentMember | `<Dimension>.CurrentMember` |
| DefaultMember | `<Dimension>.DefaultMember` |
| Descendants | `Descendants(<Member>,[<Level>[,<Desc flags>]])`<br><br>**Note:** Only`Descendants(<Member>,[<Level>[,<Desc flags>]])` is supported here. `Descendants()` with the `<set>` option is not supported here. |
| Distinct | `Distinct(<Set>)` |
| DrilldownLevel | `DrilldownLevel(<Set>[,{<Level>\|,<Index>}])` |
| DrilldownMember | `DrilldownMember(<Set1>,<Set2>[,RECURSIVE])` |
| DrillupMember | `DrillupMember(<Set1>,<Set2>)` |
| Except | `Except(<Set1>,<Set2>[,ALL])` |
| FirstChild | `<Member>.FirstChild` |
| FirstSibling | `<Member>.FirstSibling` |
| Generate | `Generate(<Set1>,<Set2>[,ALL])`<br><br>**Note:** `Generate(<Set1>,<Set2>[,ALL])` is supported. `Generate(<Set>,<String Expression>[,<Delimiter>])` is not supported here. |
| Head | `Head(<Set>[,<Numeric Expression])` |
| Hierarchize | `Hierarchize(<Set>[,POST])` |
| Hierarchy | `<Member>.Hierarchy`<br><br>`<Level>.Hierarchy` |
| Intersect | `Intersect(<Set1>,<Set2>[,ALL])` |

| MDX Function | Syntax |
|---|---|
| Item | <Set>.Item(Index)<br><br>**Note:** <Set>.Item(<StringExpression>[,<String Expression>]) and <Tuple>.Item(<Index>) are not supported here. |
| Lag | <Member>.Lag(<Numeric Expression>) |
| LastChild | <Member>.LastChild |
| LastPeriods | LastPeriods(<Index>,<Member>)<br><br>**Note:** <Member> is required here; optional in MSAS. |
| LastSibling | <Member>.LastSibling |
| Lead | <Member>.Lead(<Numeric Expression>) |
| Level | <Member>.Level |
| Max | Max(<Set>[,<Numeric Expression>]) |
| Median | Median(<Set>[,<Numeric Expression>]) |
| Members | <Dimension>.Members<br><br><Hierarchy>.Members<br><br><Level>.Members<br><br>**Note:** Members(<String Expression>) is not supported. |
| Min | Min(<Set>[,<Numeric Expression>]) |
| Name | <Dimension>.Name<br><br><Level>.Name<br><br><Member>.Name<br><br><Hierarchy>.Name |
| NextMember | <Member>.NextMember |
| OpeningPeriod | OpeningPeriod(<Level>,<Member>)<br><br>**Note:** <Level> and <Member> required here; optional in MSAS. |
| Order | Order(<Set>,<Numeric Expression>[,ASC\|DESC\|BASC\|BDESC]) |
| ParallelPeriod | ParallelPeriod(<Level>,<Numeric Expression>,<Member>)<br><br>**Note:** <Level>, <Numeric Expression>, and <Member> are required here, but are optional in MSAS. |
| Parent | <Member>.Parent |
| PeriodsToDate | PeriodsToDate(<Level>,<Member>)<br><br>**Note:** <Level> and <Member> are required here; optional in MSAS. |
| PrevMember | <Member>.PreviousMember |
| Properties | <Member>.Properties(<String Expression>)<br><br>**Note:** Only user-defined member properties are supported here by the Properties() function. |
| Subset | Subset(<Set>,<Start>[,<Count>]) |
| Sum | Sum(<Set>,<Numeric Expression>) |
| Tail | Tail(<Set>[,<Count>]) |
| TopCount | TopCount(<Set>,<Count>[,<Numeric Expression>]) |

| MDX Function | Syntax |
|---|---|
| TopPercent | `TopPercent(<Set>,<Percentage>[,<Numeric Expression>])`<br><br>**Note:** `<Numeric Expression>` is optional here; required in MSAS. |
| TopSum | `TopSum(<Set>,<Value>[,<Numeric Expression>])`<br><br>**Note:** `<Numeric Expression>` is optional here; required in MSAS. |
| Union | `Union(<Set1>,<Set2>[,ALL])`<br><br>`Union({<Set1>,<Set2>})` |
| UniqueName | `<Dimension>.UniqueName`<br><br>`<Level>.UniqueName`<br><br>`<Member>.UniqueName`<br><br>`<Hierarchy>.UniqueName` |

**Note:** As implemented in DB2 Alphablox, functions that assume a Time dimension (for example, ParallelPeriod or PeriodsToDate) only implement the variants that do not require knowing about that dimension, thus leaving level arguments out is not supported currently in the DB2 Alphablox Cube Server.

## MDX Query Examples

This section shows some examples of MDX queries against an DB2 Alphablox cube named *DB2AlphabloxCube*. Assume the DB2 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 [DB2AlphabloxCube]
WHERE ([Sales])
```

| Time | A | B | C | D | Z |
|---|---|---|---|---|---|
| 2001 | 12.5 | 14.25 | 34.95 | 2,503.22 | |
| 2002 | | | | | |
| 2003 | | | | | |
| 2004 | | | | | 179.7 |

## 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 DB2 Alphablox cube.

```
SELECT CrossJoin({[Products].[Product Name].[E],
     [Products].[Product Name].[F]}, [Time].[1999].Children)
     ON COLUMNS,
  {[Sales], [Cost], [Profit]} ON ROWS
FROM [DB2AlphabloxCube]
```

| | E | | | | F | | | |
|---|---|---|---|---|---|---|---|---|
| Measures | Q1 | Q2 | Q3 | Q4 | Q1 | Q2 | Q3 | Q4 |
| Sales | 17,700 | 16,80044 | 44 | 18,100 | 1,413.87 | 1413.87 | 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 |

# 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.

Licensees 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

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

| | | |
|---|---|---|
| IBM | DB2 | DB2 OLAP Server |
| DB2 Universal Database | WebSphere | |

Alphablox and Blox are trademarks or registered trademarks of Alphablox 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

# Index

# N



# P



# R



# S



# T

Program Number: 5724-L14

Printed in USA

SC18-9433-01