IBM DB2 Cube Views

Guide and Reference

Version 8.2

SC18-7298-01

IBM DB2 Cube Views

Guide and Reference

Version 8.2

SC18-7298-01

Note

Note: Before using this information and the product it supports, read the information in "Notices" on page 281.

Second Edition (September 2004)

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1800–IBM-4YOU (425–4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this book

This book provides information about the following DB2 Cube Views topics:

- How to get started with DB2 Cube Views
- The OLAP Center (graphical user interface), which you can use to import and export metadata, create cube models and cubes
- Optimization, which helps you improve the performance of OLAP queries
- Metadata objects that can be stored in the DB2 Universal Database™ (DB2®) catalogs
- Application programming interface (API), with which you can create applications that use SQL to access data
- Examples of how to build dimensions and complex measures from metadata objects; these dimensions and measures can be used to model typical business scenarios

Who should read this book

With DB2 Cube Views, you can capture multidimensional metadata from OLAP and database tools and store that metadata in the DB2 catalogs. You can then use that metadata to create OLAP (online analytical processing) cube models and cubes. (Cubes are subsets of cube models.)

DB2 Cube Views also provides an Optimization Advisor that provides providing SQL scripts to build summary tables to help you to improve the performance of queries that are issued to the cube models.

Read this book if you are a database administrator who works with OLAP metadata and DB2 Universal Database (DB2 UDB). You should be familiar with:

- The DB2 catalogs and summary tables
- OLAP concepts, such as cubes, dimensions, hierarchies, and measures
- API concepts and CLI, ODBC, JDBC, XML, and DB2 stored procedures

Syntax conventions used in this book

Command syntax is shown in running text format with the following conventions:

- Commands are shown in lowercase, bold font.
- Variables are shown in italics and are explained immediately after the command in a list.
- If you can enter one of two or more alternative parameters, the available parameters are shown separated by vertical bars, and the default parameter is underlined.
- Optional variables and parameters are enclosed in square brackets.

For example, the **copy** command syntax can be shown in the following format:

copy *filename [filetype]*

filename is the required name of the file.

[filetype] is the optional file type.

Online information

This section provides Web addresses related to this product.

www.ibm.com/redbooks
IBM® Redbooks™ Web site

Search for, view, download, or order hardcopy/CD versions of the following Redbooks from the Redbooks Web site:

- *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546-00
- *Up and Running with DB2 UDB ESE Partitioning for Performance in an e-Business Intelligence World*, SG24-6917-00
- *Database Performance Tuning on AIX®*, SG24-5511-01
- *DB2 UDB V7.1 Performance Tuning Guide*, SG24-6012-00

www.ibm.com/software/data/db2/db2md
IBM DB2 Cube Views Web site

www.ibm.com/software/data/
IBM Data Management Web site

www.ibm.com/software/data/db2/udb/winos2unix/support/
DB2 Universal Database and DB2 Connect™ Online Support Web site

www.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report
DB2 Maintenance - Fix Packs for DB2 UDB Web site

www.ibm.com/software/data/developer
The DB2 Developer Domain Web site

www.ibm.com/software/data/db2/library
DB2 Product and Service Technical Library Web site

www.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v8pubs.d2w/en_main
DB2 Publications Web site

Chapter 1. Installing, migrating, and configuring DB2 Cube Views

This section describes the following topics:

DB2 Cube Views enhancements
This release incorporates several changes to the DB2 Cube Views metadata since DB2 Cube Views V8.1.

Introduction to DB2 Cube Views
DB2 Cube Views is an add-on feature of DB2 Universal Database that improves the ability of DB2 UDB to perform OLAP processing. You can use DB2 Cube Views to streamline the deployment and management of OLAP solutions, and improve the performance of OLAP tools and applications.

Installation requirements
Before installing DB2 Cube Views, ensure that you meet all of the requirements.

Installing DB2 Cube Views
Install DB2 Cube Views on the Windows, AIX, Linux, and Solaris Operating System platforms using the Installation Launchpad.

Migrating DB2 Cube Views
You can migrate your DB2 Cube Views-enabled database from Version 8.1 to Version 8.2 by manually running a script or by using the OLAP Center.

Setting up a database for DB2 Cube Views
You can set up a new database for DB2 Cube Views to use.

Setting up the CVSAMPLE sample database
DB2 Cube Views provides sample data that you can use to create a sample database called CVSAMPLE.

Troubleshooting a database connection
If you cannot connect to a database using the OLAP Center, check that the version of DB2 Cube Views matches the version of the metadata tables in the DB2 catalog.

DB2 Cube Views enhancements

This release incorporates several changes to the DB2 Cube Views metadata since DB2 Cube Views V8.1.

Constraint and functional dependency information to improve summary tables

DB2 Cube Views utilizes constraint and functional dependency information to produce summary tables that are smaller and provide better query coverage.

Constraint information

In DB2 Cube Views, cube model constraints are defined between the fact and dimension tables (or between tables of a snowflake dimension). If a summary table contains the primary key of a dimension table, DB2 UDB can reroute the query to the summary table even if the query references other columns from the dimension.

Functional dependency information

DB2 UDB has intelligent routing capability based on functional dependency information to resolve SQL queries.

Functional dependencies allow you to specify that one or more columns are functionally dependent upon one or more other columns, provided all of the columns exist within the same table.

If an SQL query includes a column from within a summary table and a column from a table from which the summary table was constructed *and* there is a functional dependency between the two columns, the DB2 Optimizer can resolve the query by joining the two tables together to get the final result set.

DB2 Cube Views can recommend summary tables that include just the key columns from selected dimensions and levels so that the resulting summary table is more narrow.

User-specified optimization slices to improve summary tables

This release of DB2 Cube Views now includes a mechanism for you to specify the type of queries that will be used and what region of the cube they will reference. The Optimization Advisor can use this additional metadata to focus the optimization on the appropriate regions of a cube. Any type of queries can benefit, although report queries are likely to have the most significant improvement.

Multiple XML schema versions supported by the metadata API

The DB2 Cube Views metadata API fully supports only XML schema version number 8.2.0.1.0, including the new DESCRIBE and TRANSLATE operations.

All of the XML documents passed to and from the metadata API are required to have a version number. The version number enables the stored procedure to identify what XML schema the client is using.

Attributes and measures nullability

Metadata API attributes and measures have been extended to include a read-only property called nullable, which is a value set comprised of *yes*, *no*, or *unknown* values.

Nullability of *yes*
The attribute or measure might contain null values.

Nullability of *no*
The attribute or measure can never contain the null value.

Nullability of *unknown*
The nullability could not be determined by the API, or could not be determined by the migration utility when the attribute or measure was migrated from V8.1 to V8.2.

Modeling of hierarchies using levels

In previous releases of DB2 Cube Views, hierarchies were modeled from an ordered list of attributes, which consisted of attributes and objects called attribute relationships. In this release, hierarchies are modeled from an ordered list of level objects with each level referencing one or more attributes.

This release includes a new Level Wizard and level properties window, which allow you to create and modify the new level objects.

Functional dependencies for level objects

When you create a level, you can request that DB2 Cube Views attempt to create a functional dependency object.

A functional dependency indicates that a level object's default attribute and related attributes are functionally determined by the level's key attributes. The Optimization Advisor can then use the functional dependency to minimize the size of the summary tables that it recommends.

If a functional dependency cannot be created, a warning message is returned. For example, functional dependencies cannot point to columns that span more than one table.

Restriction: Query results could be incorrect if the underlying columns do not comply with the relationship that is defined in the functional dependency for the level object. Because DB2 UDB does not check the validity of the data with regard to any functional dependencies, you must ensure that data in the table columns are functionally dependent in the way that you specified.

For more information about functional dependencies, see "Levels" on page 26.

Automated operation of the Optimization Advisor

This release includes the new API ADVISE operation. This operation recommends summary tables that should be built to improve query performance for a cube model. This operation has arguments that restrict how long the Optimization Advisor can run and how much disk space it can use for the summary tables.

Updated sample database called CVSAMPLE

A new and improved sample database is provided called CVSAMPLE. The CVSAMPLE database is a more robust snowflake schema. DB2 Cube Views metadata is also provided that reflects the metadata changes in this release. For information about how to set up the CVSAMPLE database, see "Setting up the CVSAMPLE sample database" on page 8.

Introduction to DB2 Cube Views

DB2 Cube Views is an add-on feature of DB2 Universal Database that improves the ability of DB2 UDB to perform OLAP processing. You can use DB2 Cube Views to streamline the deployment and management of OLAP solutions, and improve the performance of OLAP tools and applications.

With DB2 Cube Views, you can describe the dimensional structure of your relational tables and create OLAP constructs. You can store the structural information and the OLAP constructs as multidimensional metadata in the DB2 database.

The new multidimensional metadata in DB2 UDB provides two major benefits:

Improves the flow of the multidimensional metadata between business intelligence tools and applications
Using the OLAP Center, a graphical interface that is provided, users of warehousing and business intelligence tools can store the multidimensional metadata as part of the DB2 database, and make it available for all tools and applications.

Enhances the performance of OLAP-style queries
Based on the multidimensional metadata, you can create DB2 summary tables using the recommendations from the Optimization Advisor in the OLAP Center. The summary tables contain precalculated data that maps to your OLAP structures. Queries that are generated from the warehousing or business intelligence application with the same OLAP structure will gain performance improvement.

DB2 Cube Views exploits DB2 features such as summary tables, different index schemes, OLAP-style operators, and aggregation functions. The following components are provided:

Multidimensional metadata objects
You can create a set of metadata objects to dimensionally model your relational data and OLAP structures. DB2 Cube Views stores each of the metadata objects that you create in the DB2 catalog.

OLAP Center
With the OLAP Center, you can create, manipulate, import, or export cube models, cubes, and other metadata objects to be used in OLAP tools. The OLAP Center provides easy-to-use wizards and windows to help you work with your metadata objects. For example, the Optimization Advisor analyzes your metadata objects and recommends how to build summary tables that store and index aggregated data for your OLAP-style SQL queries. To start the OLAP Center, see Starting the OLAP Center. After you start the OLAP Center, see Optimizing a cube model to use the Optimization Advisor wizard.

Multidimensional Services
DB2 Cube Views provides an SQL-based and XML-based application programming interface (API) for OLAP tools and application developers. Using CLI, ODBC, or JDBC connections, or by using embedded SQL to DB2 UDB, applications and tools can use a single stored procedure to create, modify, and retrieve metadata objects.

Sample data
A sample application and database are available to help you learn how to use the product.

You can also exchange metadata objects between the DB2 catalog and OLAP tools. To import or export metadata objects to or from the DB2 catalog, utilities called metadata bridges are available for specific OLAP and database tools. See the documentation for your particular OLAP or database tool to determine if a metadata bridge is provided.

Installation requirements

Before installing DB2 Cube Views, ensure that you meet all of the requirements.

System requirements

You must install the appropriate components on each DB2 UDB server and client that you want to connect to. Multidimensional Services is required on the server and the client. You can also install OLAP Center on the client.

Restriction: You must install the same version of DB2 Cube Views for all client and server components. The OLAP Center, Version 8.1 client can connect to a DB2 UDB, Version 8.2 server but cannot perform any Create, Alter, or Drop operations.

You must have the following server, client, and hardware components.

- Server component:

  Microsoft® Windows®
  Windows NT® 4,Windows 2000® 32-bit, Windows XP Professional 32-bit, Windows Server 2003 32-bit or Windows Server 2003 64-bit

  AIX AIX Version 4.3.3 32-bit, AIX 5L™ 32-bit, or AIX 5L 64-bit

  Linux®:
  Linux Red Hat™ 8 (kernel 2.4.18, glibc 2.2.93-5) 32-bit, Linux SuSE 8.0 (kernel 2.4.18, glibc 2.2.5) 32-bit, Linux SLES 8 SP3 (kernel 2.4.21, glibc 2.2.5) 32-bit, or Linux RHEL 3 Update 2 (kernel 2.4.21-9, glibc 2.3.2.) 32-bit.

  For the latest information about distribution and kernel levels supported by DB2 UDB, go to: www.ibm.com/db2/linux/validate

  On Sun Solaris™ Operating System
  Solaris 8 32-bit, Solaris 8 64-bit, Solaris 9 32-bit, or Solaris 9 64-bit

  On HP-UX
  HP-UX 11i v2 64-bit for Intel™ Itanium
- Client component: Windows NT 4, Windows 2000 32-bit, Windows XP 32-bit, Windows XP 64-bit, Windows Server 2003 32-bit, or Windows Server 2003 64-bit
- Hardware components:
  - 500 MB disk space
  - 256 MB RAM

Prerequisites to installing DB2 Cube Views

Before installing DB2 Cube Views, install the following components.

DB2 Information Center

To access the online help for the OLAP Center and the online version of the *DB2 Cube Views Guide and Reference*, you must install the DB2 Information Center. Install the DB2 Information Center from the DB2 Information Center CD.

DB2 Universal Database, Version 8.2

You must install the DB2 UDB, Version 8.2 before you can install DB2 Cube Views.

Note: Remove all previous versions of DB2 UDB before you install DB2 UDB, Version 8.2.

Installing DB2 Cube Views

Install DB2 Cube Views on the Windows, AIX, Linux, and Solaris Operating System platforms using the Installation Launchpad.

Installing DB2 Cube Views on Windows

Before you install DB2 Cube Views, ensure that:

- You installed the DB2 Information Center.
- You completed a clean installation of DB2 Universal Database, Version 8.2.
- You meet all of the installation requirements.

To install DB2 Cube Views on Windows:

1. Insert the DB2 Cube Views CD. The installation program starts automatically.
2. On the Launchpad, click **Release Notes** for the latest installation, disk, and memory requirements. Additionally, check the readme.txt file in the root of the DB2 Cube Views CD for any additional instructions.
3. Click **Install Products** to begin the installation, and follow the prompts.

Installing DB2 Cube Views on AIX, Linux, or Solaris Operating System

Before you install DB2 Cube Views, ensure that:

- You completed a clean installation of DB2 Universal Database, Version 8.2.
- You meet all of the installation requirements.

To install the DB2 Cube Views on AIX, Linux, or Solaris Operating System:

1. Insert the DB2 Cube Views CD.
2. Switch to the directory for your UNIX operating system, and launch the db2setup file.
3. On the Launchpad, click **Release Notes** for the latest installation, disk, and memory requirements. Additionally, check the readme.txt file in the root of the DB2 Cube Views CD for any additional instructions.
4. Click **Install Products** to begin the installation, and follow the prompts.

Migrating DB2 Cube Views XML files from V8.1 to V8.2

You can migrate your DB2 Cube Views-enabled database from Version 8.1 to Version 8.2 by manually running a script or by using the OLAP Center.

Migrating with a DB2 script

Recommendation: Back up and export your Version 8.1 XML metadata prior to running this migration script.

Use the db2mdmigrate.sql file in the sqllib\misc directory to create the new metadata tables and the additional SQL that is necessary to migrate metadata from V8.1 to V8.2. To use this file:

1. Connect to the database that you want to migrate by entering the following command:

   ```
   db2 connect to database_name
   ```

2. Enter the following command from the DB2 command window to run the db2mdmigrate.sql file.

```
db2 -tvf db2mdmigrate.sql
```

This script has no error handling. The script proceeds through a set of DDL and SQL statements. If any of the statements fail or if you abort the script before it completes, the migration will be only partially done, and DB2 Cube Views might not work properly.

3. If you encounter errors do the following tasks.
   a. Drop all DB2INFO.* tables.
   b. Drop the DB2INFO.MD_MESSAGE stored procedure.
   c. Re-create the metadata tables using the file db2mdapi.sql in the sqllib\misc directory.

Migrating with the OLAP Center

When the OLAP Center connects to a DB2 UDB database, the OLAP Center automatically detects the current version of DB2 UDB and determines if the metadata tables need to be migrated. If the OLAP Center determines that you need to migrate, it will display an error message to inform you.

If you accept the OLAP Center's recommendation to migrate, the OLAP Center will connect to the database and migrate the metadata tables to V8.2. If you decline to migrate, the OLAP Center will not connect to the database.

If any errors occur during the migration progress, the OLAP Center will roll back the transaction, and the database will not be migrated.

Setting up a database for DB2 Cube Views

You can set up a new database for DB2 Cube Views to use.

Setting up a database includes:

- Registering the DB2 Cube Views stored procedure with the database
- Creating metadata catalog tables for DB2 Cube Views

When you first log on to a database that is not configured for DB2 Cube Views by using the OLAP Center, the OLAP Center sets up the database for you. Alternatively, you can set up the database using the db2mdapi.sql file.

Important: Do not alter the db2mdapi.sql file or your results will be unpredictable.

To set up a database using the db2mdapi.sql file:

1. Open the DB2 Command window and connect to your database.
2. Change to the SQLLIB\misc directory and enter the following command in the DB2 Command window:

   ```
   db2 -tvf db2mdapi.sql
   ```

   Run the db2mdapi.sql script only once for a database. If you encounter errors, fix the problem then do the following tasks.
   a. Drop all DB2INFO.* tables.
   b. Drop the DB2INFO.MD_MESSAGE stored procedure.
   c. Re-create the metadata tables using the file db2mdapi.sql in the sqllib\misc directory.

Setting up the CVSAMPLE sample database

DB2 Cube Views provides sample data that you can use to create a sample database called CVSAMPLE.

The sample data includes a set of tables that contain data about a fictional company that sells beverages. A set of metadata objects that describe the sample data tables is also included. The CVSAMPLE data that is provided is an improved and expanded version of the MDSAMPLE database that was provided in Version 8.1. Most of the examples in the *DB2 Cube Views Guide and Reference* are based on the CVSAMPLE database and corresponding cube model.

Create and populate the sample CVSAMPLE database by opening the DB2 Command window and entering the following commands:

1. Create a sample database called CVSAMPLE:
   ```
   db2 create db cvsample
   ```
2. Connect to the database:
   ```
   db2 connect to cvsample
   ```
3. Run the db2mdapi.sql script to set up the database for DB2 Cube Views. Change to the SQLLIB\misc directory and then enter the following DB2 command:
   ```
   db2 -tvf db2mdapi.sql
   ```
4. Change to the SQLLIB\samples\olap\cvsample directory and then enter the following DB2 command to create the CVSAMPLE tables:
   ```
   db2 -tvf CVSampleTables.sql
   ```

When you have created the CVSAMPLE database, you can create the DB2 Cube Views metadata objects by importing the definitions from an XML file that is exported from a business intelligence application.

For example, the following procedure populates the DB2 Cube Views catalog tables with a complete description of the CVSAMPLE database.

To import the CVSAMPLE metadata:

1. Start the OLAP Center and connect to the CVSAMPLE database.
2. Click **OLAP Center –> Import**.
3. Browse for the CVSampleMetadata.xml file that is located in the SQLLIB/samples/olap/cvsample directory. Click **Next**.

You can browse the metadata objects in the OLAP Center. For information about using the OLAP Center, see “Creating DB2 Cube Views metadata objects” on page 46.

Troubleshooting a database connection

If you cannot connect to a database using the OLAP Center, check that the version of DB2 Cube Views matches the version of the metadata tables in the DB2 catalog.

The following table shows how the OLAP Center behaves when the versions of DB2 Cube Views and the metadata tables in the DB2 catalog are mismatched.

Version of DB2 Cube Views	Version of the metadata tables in the DB2 catalog	Behavior of the OLAP Center
Not installed	Not applicable	Connection fails and the OLAP Center displays an error message
Version 8.1	None	Connection fails and the OLAP Center displays an error message
Version 8.1	Version 8.1	Connection fails and the OLAP Center displays an error message
Version 8.2	None	The OLAP Center can configure the database for use with DB2 Cube Views, Version 8.2
Version 8.2	Version 8.1	The OLAP Center can migrate the database for use with DB2 Cube Views, Version 8.2
Version 8.2	Version 8.2	Connection is successful.

Chapter 2. About DB2 Cube Views metadata objects

DB2 Cube Views metadata objects describe relational tables as OLAP structures, but these metadata objects are different from traditional OLAP objects. Metadata objects store metadata about the data in the base tables, they describe where pertinent data is located, and they describe relationships within the base data.

DB2 Cube Views stores information about your relational data in metadata objects that provide a new perspective from which to understand your data. DB2 Cube Views extends the DB2 catalog so that in addition to storing information about tables and columns, the DB2 catalog contains information about how the tables and columns relate to OLAP objects and the relationships between those metadata objects.

Some metadata objects act as a base to directly access relational data by aggregating data or directly corresponding to particular columns in relational tables. Other metadata objects describe relationships between the base metadata objects and link the base metadata objects together. All of the metadata objects can be grouped by their relationships to each other into a metadata object called a cube model. Essentially, a cube model represents a particular grouping and configuration of relational tables.

DB2 Cube Views manages the following metadata objects and stores them in the DB2 catalog:

- Cube models
- Facts objects
- Dimensions
- Hierarchies
- Levels
- Measures
- Attributes
- Attribute relationships
- Joins
- Cubes
- Cube facts objects
- Cube dimensions
- Cube hierarchies
- Cube levels

Metadata objects that map to relational tables

A cube model can be constructed in many ways, but is often built to represent a relational star schema or snowflake schema. A cube model contains metadata objects that describe relationships in the relational data.

A star schema has a fact table at the center and one or more dimension tables joined to the fact table, and a snowflake schema is an extension of a star schema such that one or more dimensions are defined by multiple tables. A cube model that is based on a simple star schema is built around a central facts object. The

facts object contains a set of measures that describe how to aggregate data from the fact table across dimensions. Measures describe data calculations from columns in a relational table and are joined to create the facts object. Figure 1 shows how measures and a facts object relate to relational data.

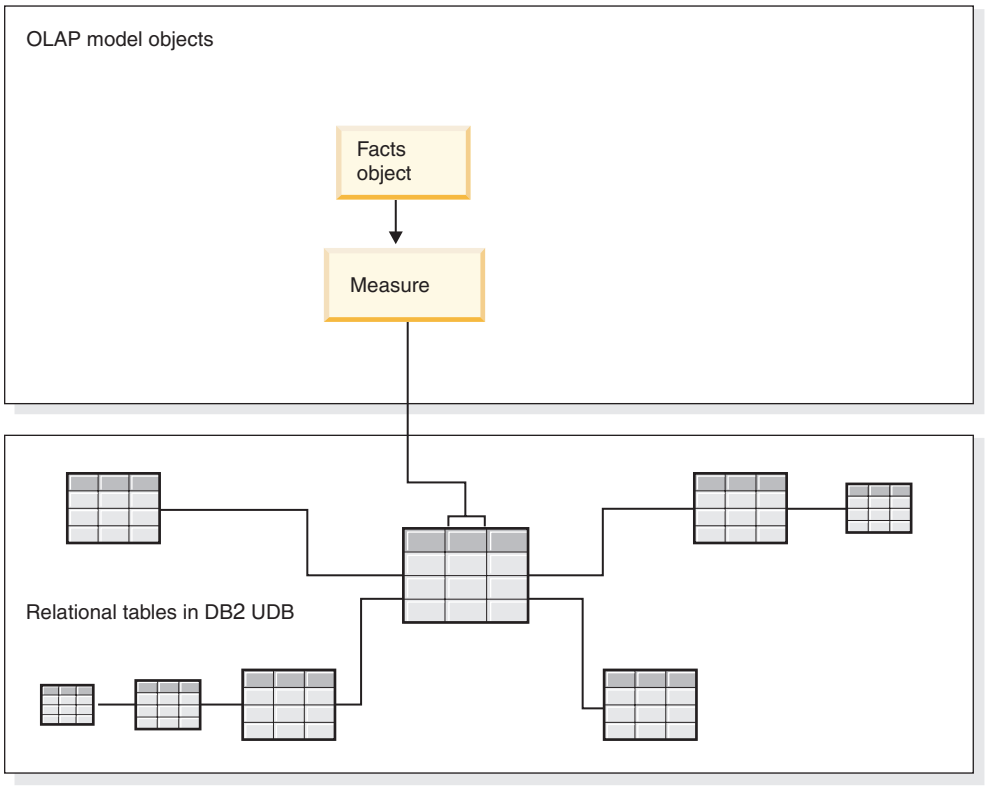

Figure 1. Facts object. How a facts object and measures relate to relational data

Dimensions are connected to the facts object in a cube model like the dimension tables are connected to the fact table in a star schema. Columns of data from relational tables are represented by attributes that are organized to make a dimension.

Figure 2 on page 13 shows how dimensions are built from relational tables. Hierarchies store information about how the levels within a dimension are related to each other and are structured. A hierarchy provides a way to calculate and navigate across the dimension. Each dimension has a corresponding hierarchy that contains levels with sets of related attributes. In a cube model, each dimension can have multiple hierarchies.

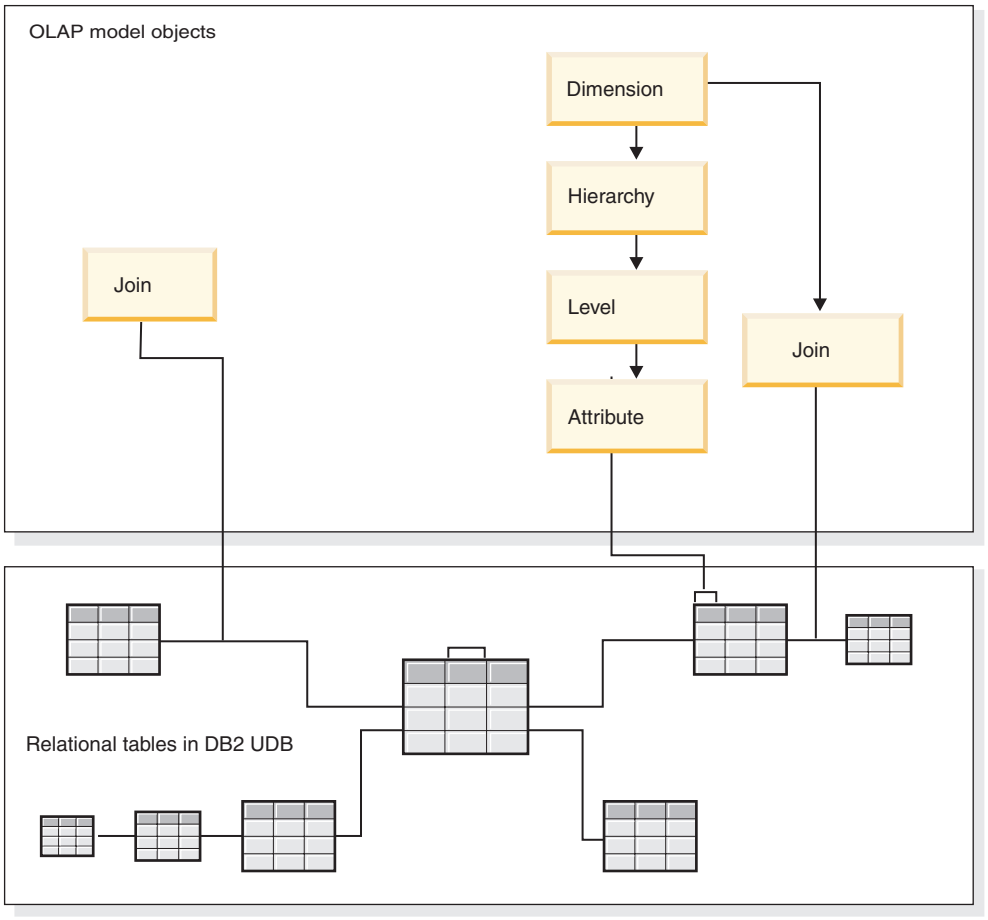

Figure 2. Dimension. How dimensions are built from relational tables

All of the dimensions are connected to a facts object in a cube model based on a star schema or snowflake schema. Joins can connect tables to create a facts object or a dimension. In a cube model, joins can connect facts objects to dimensions. The dimensions reference their corresponding hierarchies, levels, attributes, and related joins. Facts objects reference their measures, attributes, and related joins. Figure 3 on page 14 shows how the metadata objects fit together in a cube model and map to a relational snowflake schema.

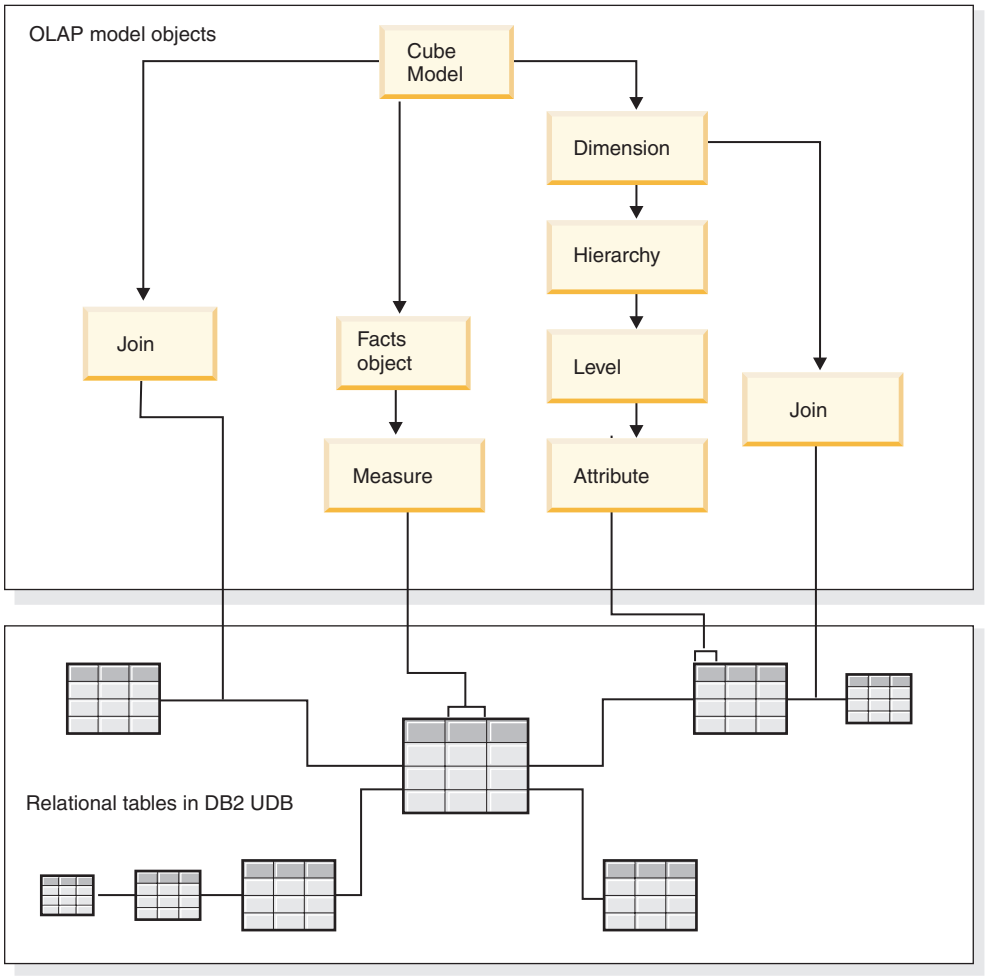

Figure 3. Cube model. How metadata objects fit together and map to a relational snowflake schema

You can reuse the components of a cube model to create more precise cubes for specific applications. A cube is the most precise metadata object and is the closest object to an OLAP conceptual cube. A cube is a specific instance or subset of a cube model. A cube has a specific set of similar but more restrictive metadata objects derived from the parent cube model, including cube dimensions, cube hierarchies, cube levels, and a cube facts object. A cube can have only one cube hierarchy defined for each cube dimension, but a dimension can have many hierarchies that are defined for the cube model. Because of this structural difference between a cube and a cube model, you can retrieve most cubes with a single SQL statement.

Example of a cube model that maps to relational tables

A cube model and corresponding set of metadata is built for the CVSAMPLE database that is based on a snowflake schema.

Figure 4 on page 15 shows a snowflake schema with a Sales fact table, with Store, Location, Time, Product, Line, and Family dimension tables. The Market dimension has two dimension tables with Store as the primary dimension table and Location as the outrigger dimension table. The Production dimension has three dimension tables with Product as the primary dimension table and the Line and Family tables are the outrigger dimension tables.

The primary key in each primary dimension table (Store, Time, and Product) is joined to the corresponding foreign key in the Sales fact table. For example, Store.StoreID = Sales.StoreID, Time.TimeID = Sales.TimeID, and Product.ProductID = Sales.ProductID.

In the snowflake dimensions, the primary key in each dimension table is joined to the corresponding foreign key in another dimension table. For example, Location.PostalcodeID = Store.PostalcodeID, Family.FamilyID = Line.FamilyID, and Line.LineID = Product.LineID.

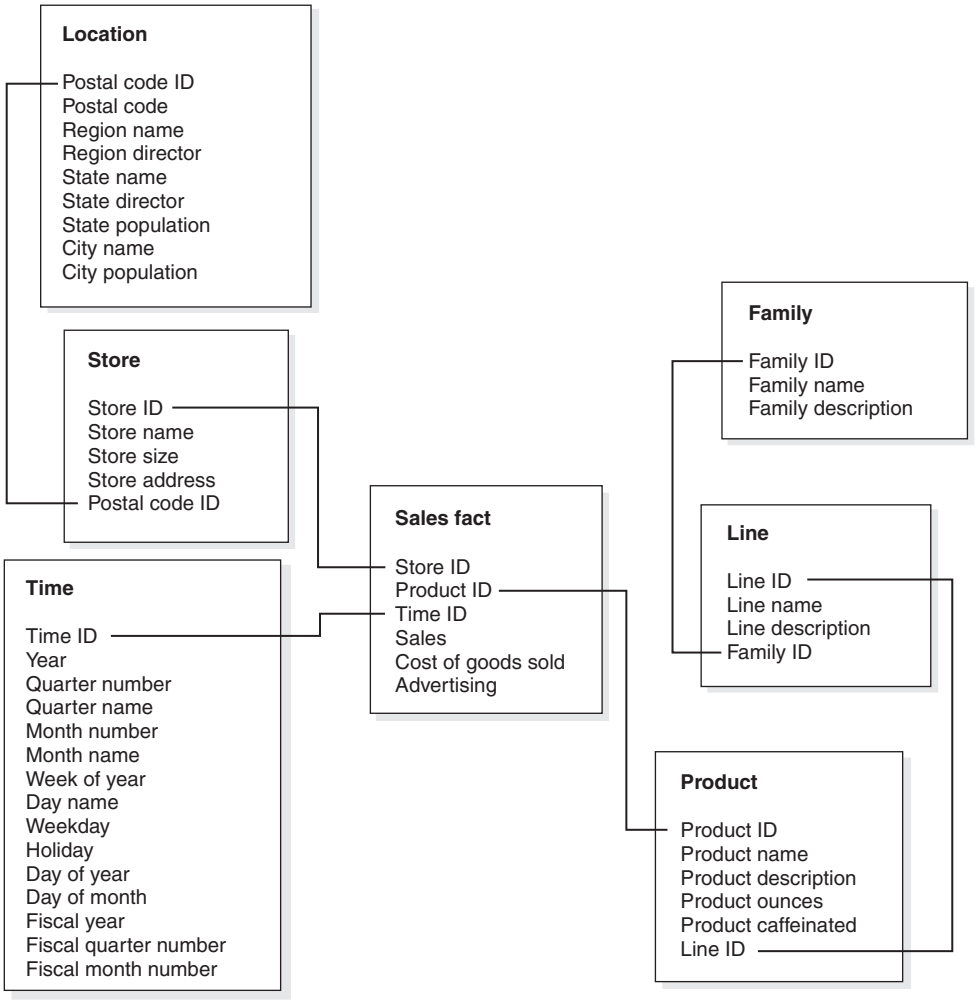

Figure 4. Snowflake schema. Example based on the CVSAMPLE snowflake schema

The cube model based on the CVSAMPLE snowflake schema is built around the Sales facts object that describes aggregated relational data from the Sales fact table. Measures describe how to calculate data from columns in the Sales table. The facts object also includes attributes that correspond to the foreign keys in the fact table that are used to join the dimensions to the facts object. In this example, the Sales facts object has seven measures: Sales, Cost of goods sold, Advertising, Total expense, Advertising-sales correlation, Profit, and Profit margin. The Sales facts object has three attributes: Store ID (Salesfact), TimeID (Salesfact) and ProductID (Salesfact).

Dimensions are connected to the facts object in a cube model like the dimension tables are connected to the fact table in a star schema. Columns of data from relational tables are represented by attribute objects referenced by the dimension.

The Product dimension references the following attributes:

- Family ID
- Family name
- Family description
- Line ID
- Line name
- Line description
- Product ID
- Product name
- Product description
- Product ounces
- Product caffeinated

The Time dimension references the following attributes:

- Year
- Quarter name
- Quarter number
- Month name
- Month number
- Time ID
- Day of month
- Day name
- Day of week
- Holiday
- Weekday
- Fiscal year
- Fiscal quarter name
- Fiscal quarter number
- Fiscal month.

The Product dimension references the following attributes:

- Region name
- Region director
- State name
- State director
- State population
- City name
- City population
- Postal code ID
- Postal code
- Store ID
- Store name

- Store size
- Store address.

A join is created to connect each dimension to the facts object. The three joins in this example are Product, Time, and Store. Figure 5 shows the cube model that is described in this example.

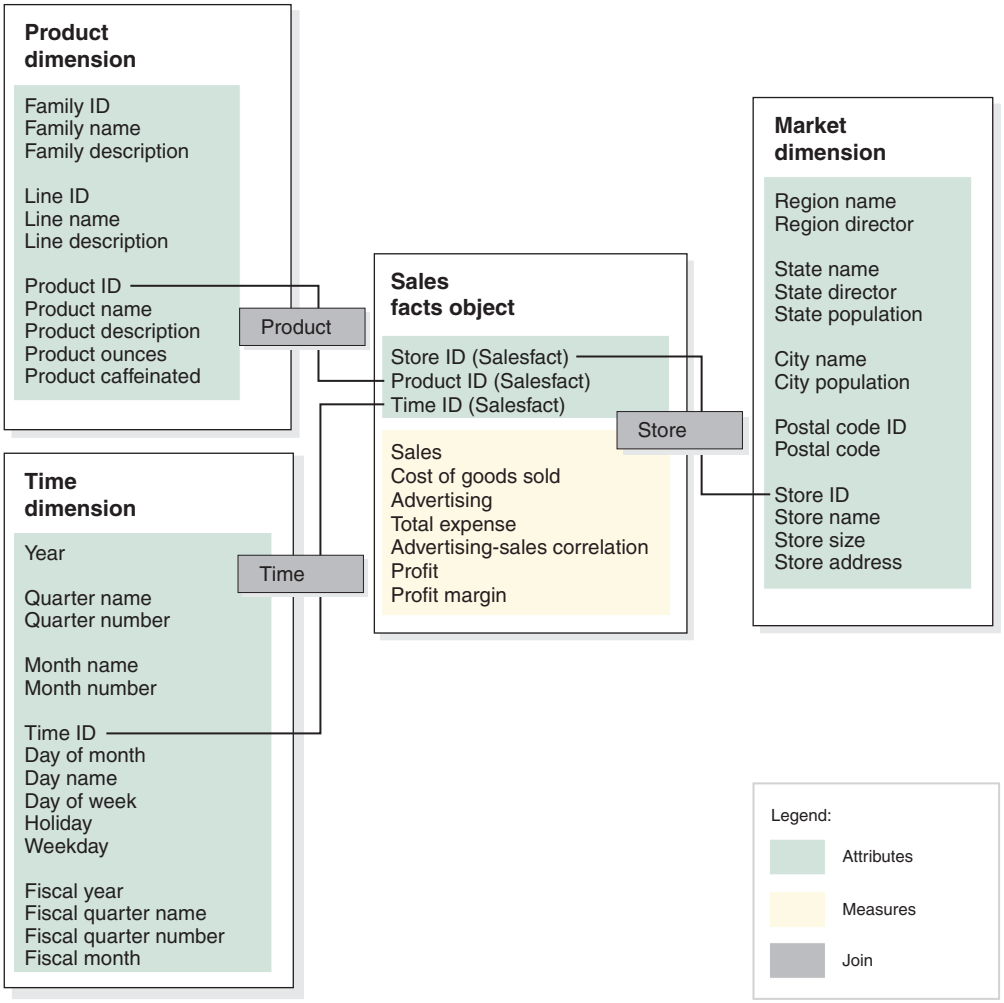

Figure 5. Cube model. Example cube model based on CVSAMPLE star schema

Hierarchies store information about how the attributes grouped into levels within a dimension are related to each other and structured. As a metadata object, a hierarchy provides a way to calculate and navigate across the dimension. Each dimension has a corresponding hierarchy with levels that group related attributes. In a cube model, each dimension can have multiple hierarchies.

The Product hierarchy includes all of the attributes in the Product dimension, as shown in Figure 6 on page 18. The attributes in the Product dimension are grouped into three levels. The Family level is the top level of the Product hierarchy. The Family level includes Family ID as the level key attribute, Family name as the default attribute, and Family description as the relate attribute. The second level, the Line level, includes Line ID as the level key attribute, Line name as the default attribute, and Line description as the related attribute. The bottom level, the Product level, includes Product ID as the level key attribute, Product name as the

default attribute, and Product description, Product ounces, and Product caffeinated as related attributes.

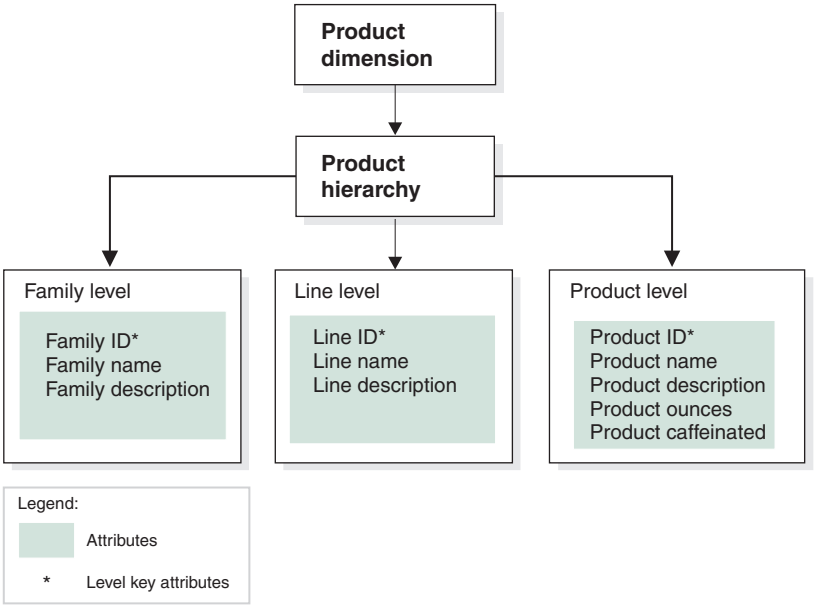

Figure 6. Dimension. Example dimension based on the Product dimension in the CVSAMPLE database

You can also build one or more cubes for the cube model. The CVSAMPLE database has two cubes, but only the General sales cube is described here. The General sales cube is shown in Figure 7 on page 19. The cube facts object references a subset of the measures (Sales, Cost of goods sold, Advertising, and Total expense) from the cube model facts object. The cube has three cube dimensions, and each cube dimension references one of the three dimensions in the cube model. The Product cube dimension has a Product cube hierarchy with cube levels that reference the Family, Line, and Product levels. The Market cube dimension has a Market cube hierarchy with cube levels that reference the Region, State, City, Postal code, and Store levels. The Time cube dimension has a Time cube hierarchy with cube levels that reference a subset of levels including Year, Quarter, and Month. The cube has only one cube hierarchy that is defined per cube dimension. (A cube can have only one cube hierarchy per cube dimension.)

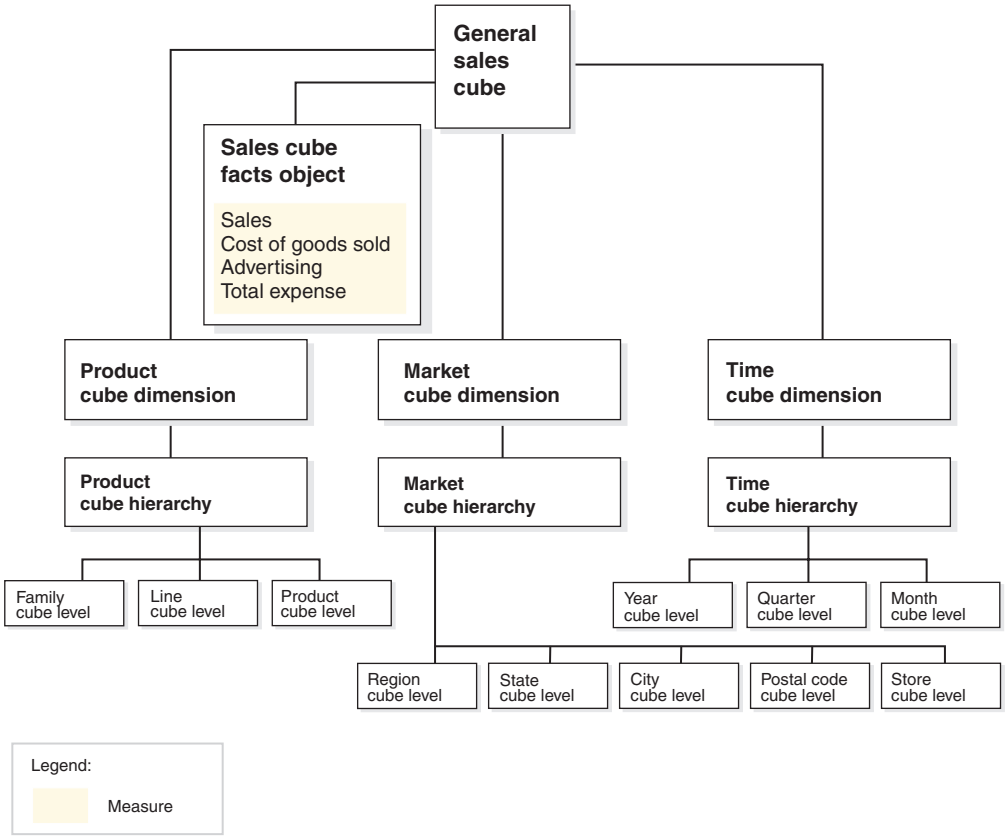

Figure 7. Cube. Example cube based on the General sales cube in the CVSAMPLE database

Common metadata object properties

Each metadata object has a set of common properties and metadata object-specific properties. The common properties are used to identify the metadata object instances, to describe the usage or role of the metadata object instances, and to track metadata object instance changes. The metadata objects are named by using a schema in the same way that other DB2 objects are named. If you do not want to use the default user name schema for a metadata object, you need to fully qualify the metadata object with the schema name that you want.

The following table describes the common properties that exist for all metadata objects.

Table 1. Common metadata object properties

Property	Description
Name	Name of the metadata object.
Schema	Schema that owns the metadata object.
Business name	Name presented to the user. This name can be used in graphic user interfaces as a name more meaningful to the user.
Comments	Textual description or comment on the nature or usage of the metadata object.

Table 1. Common metadata object properties (continued)

Property	Description
Create time	Time that the metadata object was created.
Creator	User (schema) that defined the metadata object.
Modify time	Time the metadata object was last modified.
Modifier	User (schema) that performed the modification.

In addition to a common set of properties, each metadata object has a set of specific properties. These specific properties describe the components and qualities that define the metadata object. For information about properties that are specific to each metadata object, see the topic for that metadata object.

Metadata object naming conventions

DB2 UDB provides two different naming conventions to name objects: ordinary and delimited. For metadata objects, the delimited convention is used when naming objects and referring to DB2 tables and columns. The delimited convention allows mixed case names, spaces, and special characters, such as national language characters. The complete set of characters is determined by the code page of the database in which the metadata objects are stored.

The following conventions apply to the metadata objects:

Table 2. Naming conventions for metadata objects

Object	Convention
Schema	• Length: 1-30 bytes • Restricted names: schema names must not be *SESSION* or begin with *SYS*. Only the uppercase names are restricted.
Name of metadata object	• Length: 1-128 bytes • No other restrictions.
Business name of metadata object	• Length: 1-128 bytes • No other restrictions
Comments for metadata objects	• Length: 0-254 bytes • No other restrictions
Table schema that is used in referencing columns	• Length: 1 to 128 bytes • No other restrictions
Table name that is used in referencing columns	• Length: 1 to 128 bytes • No other restrictions
Column Name that is used in referencing columns	• Length: 1-128 bytes • No other restrictions

Cube models

The DB2 Cube Views cube model is a representation of a logical star schema or snowflake schema and groups relevant dimension objects around a central facts object.

Each dimension can have multiple hierarchies. The structural information about how to join the tables that are used by the facts object and dimensions is referenced by the cube model. Also stored in the cube model is enough information to construct SQL queries and to retrieve OLAP data. Other reporting and OLAP tools that understand the cube model and that can display multiple views of a specific dimension can benefit from using the cube model.

Cube models define a complex set of relationships and can be used to selectively expose relevant facts objects and dimensions to an application. Each join object that connects a dimension to the central facts object is stored with the corresponding dimension as a set. Subsets of cube model components can be used by many cubes for different analysis purposes.

You can create an empty cube model in the OLAP Center by using the Cube Model wizard. An empty cube model does not have a facts object or any dimensions. With the wizards in the OLAP Center, you can complete the cube model by creating the facts object and one or more dimensions. You can also create a complete cube model using the Quick Start wizard. DB2 Cube Views will validate your cube model when open the Optimization Advisor. To optimize a cube model, the cube model must contain the following mandatory components:

- A facts object
- At least one dimension
- A hierarchy defined for at least one dimension
- Joins between the existing facts object and dimensions
- Attributes that reference existing table columns

The properties that are specific to cube models are described in the following table.

Table 3. Cube model properties

Property	Description
Facts object	Facts object that is used in the cube model
Set of (dimension, join)	Dimensions that are used in the cube model and their corresponding joins

Facts objects

A facts object is used in a cube model as the center of a star schema and groups related measures that are interesting to a particular application.

The facts object references the attributes that are used in facts-to-dimension joins, and the attributes and joins that are used to map the additional measures across multiple database tables. Multiple relational fact tables can be joined on specific attributes to map additional related measures. Therefore, in addition to a set of measures, a facts object stores a set of attributes and a set of joins.

You can use the Facts wizard in the OLAP Center to create a facts object. In the Facts wizard you specify one or more fact tables and any necessary joins, measures, and aggregations for the measures.

The specific properties of a facts object are described in the following table.

Table 4. Facts object properties

Property	Description
Set of measures	Set of all related measures in the facts object
Set of attributes	Set of all attributes that are used in the facts object
Set of joins	Set of all joins that are needed to join all of the specified measures and attributes

Dimensions

Dimensions provide a way to categorize a set of related attributes that together describe one aspect of a measure. Dimensions are used in cube models to organize the data in the facts object according to logical categories such as Region, Product, or Time.

Dimensions reference zero or more hierarchies. Hierarchies describe the relationship and structure of the referenced attributes that are grouped into levels, and provide a navigational and computational way to traverse the dimension.

Related attributes and the joins that are required to group these attributes are defined in the properties of the dimension.

Dimensions also have a type that describes if the dimension is time-oriented. For example, a dimension called Time that contains levels like Year, Quarter, and Month is a Time type. Another dimension called Market that contains levels like Region, State, and City is a Regular type. Type information can be used by applications to intelligently and appropriately perform time-related functions.

You can use the Dimension wizard in the OLAP Center to create a new dimension in the context of a cube model or without a reference to a cube model. You can share dimensions across cube models by adding an existing dimension to a cube model with the Add Dimension wizard.

The specific properties of dimensions are described in the following table.

Table 5. Dimension properties

Property	Description
Set of attributes	Set of all attributes that are used in the dimension.
Set of joins	Set of all joins that are required to join all of the specified attributes. Only the joins that are required to join the dimension tables are specified here.
Set of hierarchies	Set of hierarchies that apply to the dimension.
Set of levels	Set of levels that are referenced by the dimension.
Type	Dimension type that can be Regular or Time

Hierarchies

A hierarchy defines relationships among a set of attributes that are grouped by levels in the dimension of a cube model. These relationships provide a navigational and computational means of traversing dimensions. Multiple hierarchies can be defined for a dimension of a cube model.

The hierarchy type describes the relationship among the levels within the hierarchy. The following four hierarchy types are supported:

Balanced

A hierarchy with meaningful levels and branches that have a consistent depth. Each level's logical parent is in the level directly above it. A balanced hierarchy can represent time where the meaning and depth of each level, such as Year, Quarter and Month, is consistent. They are consistent because each level represents the same type of information, and each level is logically equivalent. Figure 8 shows an example of a balanced time hierarchy.

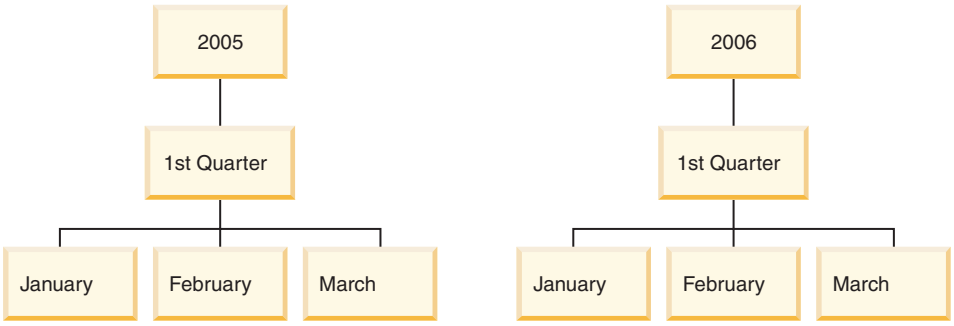

Figure 8. Balanced hierarchy. Example of a balanced hierarchy

Unbalanced

A hierarchy with levels that have a consistent parent-child relationship but have a logically inconsistent levels. The hierarchy branches also can have inconsistent depths. An unbalanced hierarchy can represent an organization chart. For example, Figure 9 on page 24 shows a chief executive officer (CEO) on the top level of the hierarchy and at least two of the people that might branch off below including the chief operating officer and the executive secretary. The chief operating officer has more people branching off also, but the executive secretary does not. The parent-child relationships on both branches of the hierarchy are consistent. However, the levels of both branches are not logical equivalents. For example, an executive secretary is not the logical equivalent of a chief operating officer.

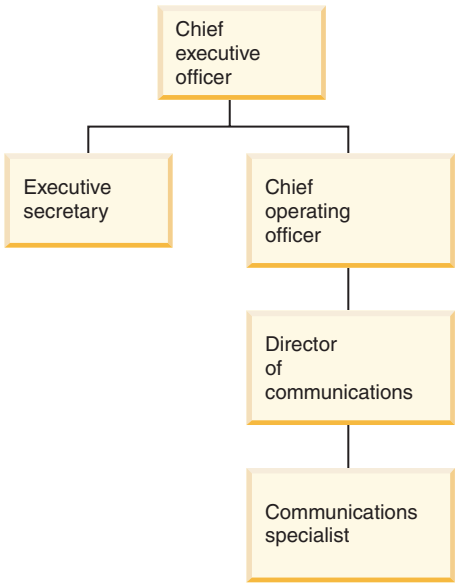

Figure 9. Unbalanced hierarchy. Example of an unbalanced hierarchy

Ragged

A hierarchy in which each level has a consistent meaning, but the branches have inconsistent depths because at least one member attribute in a branch level is unpopulated. A ragged hierarchy can represent a geographic hierarchy in which the meaning of each level such as city or country is used consistently, but the depth of the hierarchy varies. Figure 10 shows a geographic hierarchy that has Continent, Country, Province/State, and City levels defined. One branch has North America as the Continent, United States as the Country, California as the Province/State, and San Francisco as the City. However, the hierarchy becomes ragged when one member does not have an entry at all of the levels. For example, another branch has Europe as the Continent, Greece as the Country, and Athens as the City, but has no entry for the Province/State level because this level is not applicable to Greece for the business model in this example. In this example, the Greece and United States branches descend to different depths, creating a ragged hierarchy.

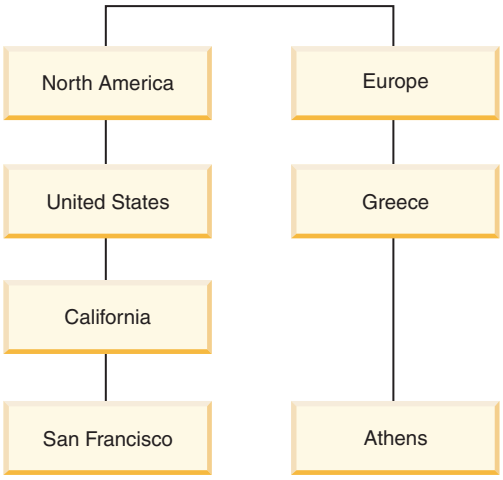

Figure 10. Ragged hierarchy. Example of a ragged hierarchy

Network

A hierarchy in which the order of levels is not specified, but in which levels do have semantic meaning. For example, Figure 11 shows a network hierarchy that describes product attributes such as Color, Size, and PackageType. Because the levels do not have an inherent parent-child relationship, the order of the levels is not important. A widget company might have member entries like white for Color, small for Size, and shrink wrap for PackageType. A second member entry might be red for Color, large for Size, and box for PackageType.

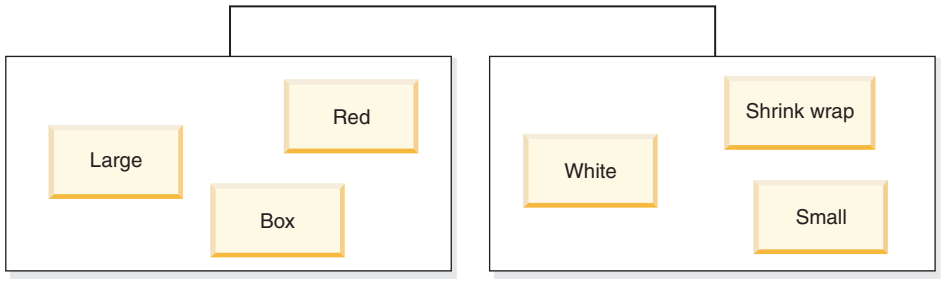

Figure 11. Network hierarchy. Example of a network hierarchy

A hierarchy also specifies the deployment mechanisms for the hierarchy. A deployment mechanism defines how to interpret the data in levels of a hierarchy. The following two deployment mechanisms are supported:

Standard

Uses the relationship of the level definitions of the hierarchy, where each level in the hierarchy is used as one item in the deployment. For example, a balanced hierarchy for a Time dimension would be organized by each defined level including Year, Quarter, and Month. Standard deployment can be used with all four hierarchy types. Table 6 shows how some of the balanced hierarchy attributes for a Time dimension are organized using a standard deployment

Table 6. Standard deployment. Standard deployment of a balanced hierarchy for a Time dimension

Year	Quarter	Month
2003	Qtr 1	Jan
2003	Qtr 1	Feb
2003	Qtr 1	Mar
2004	Qtr 1	Jan
2004	Qtr 1	Feb
2004	Qtr 1	Mar

Recursive

Uses the inherent parent-child relationships between the levels of the hierarchy. An unbalanced hierarchy using a recursive deployment is represented as parent-child level pairs. For example, Table 7 on page 26 shows the level pairs for the unbalanced hierarchy describing an organization chart shown in Figure 9 on page 24. The parent-child level pairs include: chief executive officer and executive secretary, chief executive officer and chief operating officer, chief operating officer and director of communications, director of communications and communications

specialist. Recursive deployment can be used only with an unbalanced hierarchy.

Table 7. Recursive deployment. Recursive deployment of an unbalanced hierarchy for an Organization dimension

Parent attribute	Child attribute
Chief executive officer	Executive secretary
Chief executive officer	Chief operating officer
Chief operating officer	Director of communications
Director of communications	Communications specialist

You can create a hierarchy in the OLAP Center using the Hierarchy wizard. You can define a hierarchy for a dimension after you create the dimension.

The properties of a hierarchy object are described in the following table.

Table 8. Hierarchy properties

Property	Description
Ordered set of levels	Ordered list of one or more levels from the top to the bottom of a hierarchy.
Type	Hierarchy type can be Balanced, Unbalanced, Ragged, Network
Deployment	Hierarchy deployment can be Standard, Recursive

Levels

A level consists of a set of attributes that work together as one logical step in a hierarchy's ordering. A level contains one or more attributes that are related and can function in one ore more roles in the level. The relationship between the attributes in a level is usually defined with a functional dependency.

A level can use an attribute in one of three ways:

Level key attributes

One or more attributes whose values uniquely identify each of the instances of the level. For example, for City level, the simplest way to define a level key attribute is to use an ID column, such as City ID, that is guaranteed to be unique. A City name attribute cannot be a level key attribute by itself because city names are not necessarily unique across states and countries. However, you can include the set of Country name, State name, and City name attributes as the level key attributes because the set of these three attributes can uniquely define a city. You must ensure that the level key attributes uniquely define the level because DB2 Cube Views creates functional dependencies to improve optimization based on the levels that you define.

Default attribute

An attribute whose value can be displayed by a reporting application to provide meaningful names for each instance of a level in the data. The default attribute is required, and must be functionally determined by the level key attributes. For example, if the level key attribute is defined as

City ID, the values for the City ID column would not be very useful in reports. You can define City name as the default attribute to be shown in reports so that the data that is displayed is useful. You can use an attribute from the set of level key attributes as the default attribute. For example, if you define the level key attribute for the City level as the set of Country name, State name, and City name, you can define the default attribute as City name.

Related attributes

A collection of zero or more attributes that provide additional information about the instances of the levels that are defined as level key attributes. All related attributes must be functionally determined by the level key attributes. For example, a City level might have related attributes such as City mayor, City population, City location, and City description.

You can use the Level wizard in the OLAP Center to create a new level independently or in the context of a hierarchy. You can share levels across cube models in dimensions. When you create a level, DB2 Cube Views creates a functional dependency by default for the attributes in the level, such that the default attributes and related attributes are functionally dependent on the level key attributes within the level. The Optimization Advisor uses the functional dependencies to recommend the best summary tables and the DB2 optimizer uses the functional dependencies to correctly route SQL queries.

There are many ways to model a hierarchy using levels. Whether you follow ideal or nonideal modeling techniques, it is important that you define each level so that the level key attributes functionally determine the default attribute and related attributes. The level key attributes in a level must uniquely identify the values in that level. You must define that a functional dependency should be created between the attributes in each level. Functional dependencies are used by both the Optimization Advisor to recommend summary tables and the DB2 optimizer to correctly route your SQL queries. The functional dependencies enable the Optimization Advisor wizard to recommend smaller summary tables that can return query results more quickly.

Ideal modeling

Ideally, each dimension's relational data is stored in a single dimension table that contains ID columns for each of the levels in the dimension, and each ID column uniquely identifies the values in the level. For example, you might have a single dimension table for the Region dimension that contains the following columns:

Table 9. Ideal modeling of a dimension table

Columns in an ideal Region dimension table
City ID (Primary key)
City name
City mayor
State ID
State name
State governor
Country ID
Country name

You can create a hierarchy that contains Country, State, and City levels. In each of the levels, you can define a functional dependency between the ID column that is defined as the level key attribute and any related attributes, as shown in Table 10.

Table 10. Ideal modeling of a hierarchy

Level	Level key attribute	Level related attributes
Country	Country ID	Country name
State	State ID	State name
		State governor
City	City ID	City name
		City mayor

Functional dependencies are created between the following attribute pairs:

- Country ID –> Country name
- State ID –> State name, State governor
- City ID –> City name, City mayor

DB2 Cube Views does not create a functional dependency for City ID and the related attributes because City ID is the primary key and should already have a constraint defined.

Nonideal modeling

If you do not have unique ID data columns for each level in your hierarchy, you must be more careful when you define the level key attributes for each level so that you create a valid functional dependency. For example, you might have a single dimension table for the Region dimension that contains the following columns:

Table 11. Nonideal modeling of a dimension table

Columns in a non-ideal Region dimension table
City ID (Primary key)
City name
City mayor
State name
State governor
Country name

You can create a hierarchy that contains Country, State, and City levels, like in the ideal modeling example. However, the level key attributes must be more carefully defined to ensure that each row in the level can be uniquely defined. For example, City name does not uniquely define the City level because there is a city named Leicester in the United States and in England. The only way to uniquely define the City level is with the combination of the Country name, State name, and City name attributes, as shown in Table 12.

Table 12. Nonideal modeling of a hierarchy

Level	Level key attributes	Level related attributes
Country	Country name	

Table 12. Nonideal modeling of a hierarchy (continued)

Level	Level key attributes	Level related attributes
State	Country name	State governor
	State name	
City	Country name	City mayor
	State name	
	City name	

Functional dependencies are created between the following two attribute combinations:

- Country name, State name, and City name –> City mayor
- Country name and State name –> State governor

The properties of a level object are described in the following table.

Table 13. Level properties

Property	Description
Set of level key attributes	Set of one or more attributes that together, uniquely define the level.
One default attribute	One required default attribute that is functionally determined by the level key attributes and can be used by reporting applications to display meaningful data.
Set of related attributes	Set of zero or more attributes that contain more information about the level and are functionally determined by the level key attributes.
Functional dependency	Boolean (Yes or No) specifying if the level has a corresponding DB2 UDB functional dependency.
Name of the functional dependency	If Functional dependency is set to Yes, this property contains the name (up to 18 bytes in length) of the DB2 UDB functional dependency. If Functional dependency is set to No, this property is ignored.

Measures

Measures define a measurement entity and are used in facts objects.

Measures become meaningful within the context of a set of dimensions in a cube model. For example, a revenue of 300 is not meaningful by itself. When you put a revenue measure in the context of dimensions, such as Region and Time, the measure becomes meaningful: the revenue for New York in January is 300. Common examples of measures are Revenue, Cost, and Profit.

A measure is defined by a combination of two properties: an SQL expression list and an aggregation list. Table columns, attributes, and measures are mapped to a template to build SQL expressions. The resulting SQL expressions are then used as input for the first aggregation function of the measure. If a measure has more than one aggregation, the aggregation functions are performed in the order that they are listed, with each subsequent aggregation taking the previous aggregation's result

as its input. If the SQL expression for the measure only references other measures, the aggregation function is optional. The aggregation function is optional because the referenced measures provide the aggregation.

An SQL expression for the measure is created by the combination of two properties: a template and a list of columns, attributes, and measures. The template uses a token notation where {$$n} is the token and n references a specific column, attribute, or measure from the list. The list of columns, attributes, and measures is ordered and the position of a column, attribute, or measure in the list corresponds to the token n value.

SQL expressions are used as input to the first aggregation. Each aggregation specifies a function that is applied to a corresponding list of dimensions. The aggregation function can be any aggregation function that is supported by the underlying database. The following aggregation functions are supported in DB2 Cube Views:

- AVG
- CORRELATION
- COUNT
- COUNT_BIG
- COVARIANCE
- MAX
- MIN
- REGRESSION functions (all 9 types)
- STTDEV
- SUM
- VARIANCE

Each dimension can be aggregated only once by the measure object. A measure must have one aggregation with an empty list of dimensions, and any other aggregations must each have an explicit list of dimensions. The aggregation for an empty list of dimensions is applied to all dimensions in the cube model that are not specifically being used by another aggregation.

An example of a simple measure that directly maps to a column is Revenue. The Revenue measure can be created for a cube model with three dimensions: Product, Market, and Time. Revenue has an SQL expression template of `template = "{$$1}"` that represents a simple mapping to the column specified in the one-item list of columns, attributes, and measures where `list = "Column Fact.Rev"`. The aggregation list is `(SUM, <NULL>)` where `SUM` is the aggregation function, and `<NULL>` is an empty list of dimensions. The SQL expression is used as input for the `SUM` aggregation function, which results in the SQL expression: `SUM(Fact.Rev)`.

A more complicated measure, Profit, might have an SQL expression template of `template = "{$$1} - {$$2}"`, where the list of attributes, columns, and measures is `list = "Measure Revenue, Column Fact.Cost"`. When the tokens are replaced with the correct references, the SQL expression becomes `"Revenue - Fact.Cost"`. When the revenue measure reference is expanded to its column reference, the SQL expression becomes: `"Fact.Rev - Fact.Cost"`. The Profit measure's aggregation list is : `(SUM, <NULL>)`. The profit SQL expression is used as input for the `SUM` aggregation function, and so the Profit measure's SQL is: `SUM(Fact.Rev - Fact.Cost)`.

If the measure has an aggregation function, such as CORRELATION, that requires two or more parameters, the measure will have two or more SQL expressions.

Measures also have a data type that is based on SQL data types. DB2 Cube Views automatically determines the data type of a measure. Each name, when fully qualified by a schema, must be unique among measures and attributes.

The OLAP Center hides much of the complexity of the metadata object definition. In the OLAP Center, you do not need to explicitly define the measure's list of SQL expressions or aggregation list. If you want to create a measure that directly maps to a column, attribute, or other measure, you select the source when you create the measure in the Facts wizard or the Facts Properties window. If you want to create a calculated measure, you can use the SQL Expression Builder window to create the source expression. The SQL Expression Builder provides lists of available columns, attributes, and measures, operators, and functions and constants. In the Measure Properties window, you can view the data type of the source data for the measure, and the data type of the measure after the source data is aggregated.

The following table describes the specific properties that define a measure. The OLAP Center defines each of these for you when you create a measure.

Table 14. Measure properties

Property	Description
List of SQL expressions (template, [(list of columns, attributes and measures])	List of SQL expressions used as input for the first aggregation function of the measure. Each SQL expression has a template and an ordered list of columns, attributes, and measures.
List of aggregations (function, list of dimensions)	List of aggregations that specify how to calculate the measure. Each aggregation has an SQL aggregation function and an optional list of dimensions to apply the function to.
Data type (schema, name, length, scale, nullable)	Determines the data type of the measure. The data type is based on SQL data types, and composed by data type schema, name, length, scale, and nullable. The OLAP Center displays the schema only if it is a schema other than SYSIBM.

Attributes

An attribute represents the basic abstraction of a database table column. An attribute contains an SQL expression that can be either a simple mapping to a table column or a more complex expression. These more complex expressions can combine multiple columns or attributes and can use all SQL functions including user-defined functions, if necessary.

The OLAP Center for DB2 Cube Views hides much of the complexity of the attribute object definition. In the OLAP Center, you do not need to explicitly define the expression template or parameter list of the attribute. If you want to create an attribute that directly maps to a column, you select the source column when you create the attribute in the Dimension wizard or the Dimension Properties window. If you want to create a calculated attribute, you can use the SQL Expression Builder window to create the source expression. The SQL Expression Builder provides lists of available attributes, columns, operators, functions, and constants.

If you want to create an attribute without using the OLAP Center, you must create the attribute's SQL expression definition as a combination of two properties including a template and a list of columns and attributes. The template uses a token notation where `{$$n}` is the token and `n` references a specific column or attribute from the list. The list of columns and attributes is ordered, and the position of a column or attribute in the list corresponds to the token `n` value. For example, the template of `template = "{$$1} || ' ' || {$$2}"` can be used with a corresponding list such as `list = "Column CUSTOMER.FIRSTNAME, Attribute LastName"` to concatenate the first name and last name of customers with a space between the names. When the template tokens are replaced with the correct list references, the SQL expression is `"Customer.FirstName || ' ' || LastName"`. The attribute reference is further expanded to a column reference to form the SQL expression: `"Customer.FirstName || ' ' || Customer.LastName"`.

When other attributes are used in the SQL expression that defines an attribute, the other attributes cannot form attribute reference loops. For example, if Attribute A references Attribute B, then Attribute B cannot reference Attribute A.

Each name, when fully qualified by a schema, must be unique from the names of all other attributes and measures in the database.

The following table describes the specific properties that define an attribute. The OLAP Center defines each of these for you when you create an attribute object.

Table 15. Attribute properties

Property	Description
SQL expression template	SQL expression that defines the attribute. The template references columns and attributes by using a `{$$n}` notation, where `n` is an ordinal number that corresponds to the list of columns and attributes.
List of columns and attributes for SQL expression	Ordered list of all columns and attributes that compose the attribute. These columns and attributes are applied as specified in the SQL expression template.
Data type (schema, name, length, scale, nullable)	Determines the data type of the attribute. The data type is based on SQL data types, and composed by data type schema, name, length, scale, and nullable. The OLAP Center displays the schema only if it is a schema other than SYSIBM.

Attribute relationships

An attribute relationship describes relationships of attributes in general, but attribute relationships are not part of cube model.

The relationships consist of the following properties:

- A left and a right attribute
- A type
- A cardinality
- A possible functional dependency

Type describes what the role of the right attribute is with respect to the left attribute. There are two possible types: Descriptive and Associated.

Descriptive
Specifies that the right attribute is a descriptor of the left attribute. For example, a ProductName right attribute describes a ProductCode left attribute.

Associated
Specifies that the right attribute is associated with the left attribute, but is not a descriptor of the left attribute. For example, a CityPopulation right attribute is associated with but not a descriptor of CityID.

Cardinality describes how the instances of the left and right attributes are related. You can use the following cardinalities for attribute relationships:

1:1 At most one left-attribute instance exists for each right attribute instance, and at most one right attribute instance exists for each left-attribute instance.

1:Many
At most one left-attribute instance exists for each right-attribute instance, and any number of right-attribute instances exist for each left-attribute instance.

Many:1
Any number of left-attribute instances exist for each right-attribute instance, and at most one right-attribute instance exists for each left-attribute instance.

Many:Many
Any number of left-attribute instances exist for each right-attribute instance, and any number of right-attribute instances exist for each left-attribute instance.

The functional dependency property tells whether the attribute relationship defines a functional relationship between two attributes. Specifying that an attribute relationship is a functional dependency means that you guarantee that every instance of the left attribute will determine the instance of the right attribute. DB2 Cube Views will not create a functional dependency between the attributes described by the attribute relationship, regardless of how you set the functional dependency property.

You can define several attribute relationships that indicate a functional dependency can exist between CountryID and Country, StateID and State, CityID and City and CityID and City_Population.

You can explicitly create an attribute relationship object in the OLAP Center. You can explicitly create an attribute relationship using the Attribute Relationship wizard. Open the Attribute Relationship wizard from the Relational objects view and specify all of the object definition properties.

The specific properties that define an attribute relationship object are described in the following table.

Table 16. Attribute relationship properties

Property	Description
Left attribute	Left attribute used in the relationship.
Right attribute	Right attribute used in the relationship.

Table 16. Attribute relationship properties (continued)

Property	Description
Type	Type of relationship described by the attribute relationship. The type is used to determine what role an attribute serves: Descriptive, Associated
Cardinality	Cardinality expected in the join: 1:1, 1:Many, Many:1, Many:Many
Functional dependency	Determines if the attribute relationship is also a functional dependency: Yes, No. **Note:** DB2 Cube Views will not create a functional dependency for attribute relationships, regardless of whether or not you specify Yes or No for this property.

Joins

A join is a metadata object that describes a join between two relational tables. A join references attributes that reference columns in the tables being joined.

The simplest form of a join references two attributes: one that maps to a column in the first table and one that maps to a column in the second table. You also specify an operator to indicate how the columns will be compared.

A join can also be used to model composite joins where two or more columns from the first table are joined to the same number of columns in the second table. A composite join uses pairs of attributes to map corresponding columns together. Each pair of attributes has an operator that indicates how that pair of columns will be compared.

A join also has a type and cardinality. The join types map to relational join types. Joins can be used in dimensions to join dimension tables, or in a cube model to join the dimensions of a cube model to its facts object, or within a facts object to join multiple fact tables together. You can use the Join wizard in the OLAP Center to create a join.

The specific properties that define a join are described in the following table.

Table 17. Join properties

Property	Description
List of (left attribute, right attribute, operator)	Left attribute: The attribute on the left side of the join. Right attribute: The attribute on the right side of the join. Operator: Operator expected in the join =, <, >, <>, >=, <=.
Type	Type of join expected: Inner, Full outer, Left outer, Right outer
Cardinality	Cardinality expected in the join: 1:1, 1:Many, Many:1, Many:Many

Cubes

A cube is a precise definition of an OLAP cube that sometimes can be delivered using a single SQL statement. A cube, which is derived from a cube model, contains a subset of metadata objects that are based on the metadata objects in the cube model.

The cube facts object and list of cube dimensions are subsets of those in the referenced cube model. Cubes are appropriate for tools and applications that do not use multiple hierarchies because cube dimensions allow only one cube hierarchy per cube dimension.

Cubes can be used when optimizing a cube model to specify the regions of the cube model that are the most active and the most important. You can specify optimization slices that define specific regions of the cube that are most often queried.

You can use the Cube wizard in the OLAP Center to create a cube. You must have a complete cube model to create an associated cube. The properties of a cube are described in the following table.

Table 18. Cube properties

Property	Description
Cube model	Cube model from which the cube is derived.
Cube facts object	Cube facts object that is used in the cube. The cube facts object is derived from the facts object in the cube model.
List of cube dimensions	Ordered list of cube dimensions that are used in the cube. Each cube dimension is derived from a corresponding dimension in the cube model. One cube hierarchy is associated with each cube dimension.
List of optimization slices	Set of zero or more optimization slices. Each optimization slice includes: • Type can be Drill-down, Report, MOLAP extract, Hybrid extract, or Drill through. • Set of optimization levels where one optimization level exists for each cube dimension in the cube. An optimization level references: – A cube dimension and its corresponding cube hierarchy – A cube level, Any, or All

Cube facts object

A cube facts object has a subset of measures in an ordered list from a specific facts object. A cube facts object allows a cube the flexibility to scope a cube model's facts object. The structural information, such as the joins and attributes, is referenced from the parent facts object.

In the OLAP Center, you create a cube in the context of a cube model, using the Cube wizard. You do not need to explicitly define the cube facts object because the

OLAP Center knows that the cube facts object is derived from the facts object in the associated cube model. You select which measures from the cube model facts object that you want to use in the cube.

The specific properties that define a cube facts object are described in the following table.

Table 19. Cube facts object properties

Property	Description
Facts object	Facts object from which the cube facts object is derived.
List of measures	Ordered list of measures that is used in a cube. All measures must be part of the facts object from which the cube facts object is derived.

Cube dimensions

A cube dimension is used to scope a dimension for use in a cube. The cube dimension references the dimension from which it is derived and the relevant cube hierarchy for the given cube.

Only one cube hierarchy can be applied to a cube dimension. The joins and attributes that apply to the cube dimension are referenced from the dimension definition.

In the OLAP Center, you create a cube in the context of a cube model, using the Cube wizard. You select which of the cube model dimensions that you want to have in your cube. For each dimension that you include as a cube dimension, you can select which levels to include in the cube hierarchy.

The specific properties that define a cube dimension object are described in the following table.

Table 20. Cube dimension properties

Property	Description
Dimension	Dimension from which the cube dimension is derived.
Cube hierarchy	Cube hierarchy that applies to the cube dimension.

Cube hierarchies

A cube hierarchy is a subset of a hierarchy, and it is used in a cube. A cube hierarchy references the hierarchy from which it is derived (parent hierarchy), and it can have a set of cube levels that are a subset of the parent levels from the parent hierarchy.

A cube dimension can have only one cube hierarchy. In general, a cube hierarchy has the same hierarchy type and deployment mechanism as the hierarchy from which it is derived. If the hierarchy is the network type, the cube hierarchy is balanced if no members are missing and ragged if members are missing.

In the OLAP Center you create a cube in the context of a cube model by using the Cube wizard. You select which of the cube model dimensions you want to have in your cube. For each dimension you include as a cube dimension, you can select which cube levels to include in the cube hierarchy.

The specific properties that define a cube hierarchy are described in the following table.

Table 21. Cube hierarchy properties

Property	Description
Hierarchy	Hierarchy from which the cube hierarchy is derived.
Ordered set of cube levels	Ordered set of one or more cube levels that is a subset of the levels included in the parent hierarchy. The order of the cube levels should be the same as in the parent hierarchy.

Cube levels

A cube level is a subset of a level, and it is used in a cube. A cube level references the level from which it is derived (parent level) and inherits the level key attributes and default attribute that are defined for the parent level.

A cube level can have a set of attributes that are a subset of the related attributes from the parent level.

The properties of a cube level are described in the following table.

Table 22. Cube level properties

Property	Description
Set of attributes	A set of zero or more related attributes that are a subset of the parent level's related attributes.

Metadata object rules

Three types of rules apply to metadata objects: base rules, cube model completeness rules, and optimization rules. These rules ensure that each object is valid both in and out of the context of a cube model and that effective SQL queries can be written and optimized.

Base rules

Base rules define a metadata object's validity outside the context of its use. Every metadata object has its own set of rules. A metadata object is valid if it follows all of the base rules.

Completeness rules

Completeness rules apply only to cube models and extend the base rules to ensure that a cube model links to other metadata objects appropriately and that effective SQL queries can be written.

Optimization rules

Optimization rules further extend the base rules and cube model completeness rules. These rules ensure that the SQL queries that are created for the metadata objects can be optimized successfully.

Cube model rules

Base rules:

- Dimension—join pairs must have both a dimension and a join.
- All attributes on one side of a dimension to fact join must exist in the dimension's attribute list and all attributes on the other side of the join must exist in the facts object's attribute list.
- A cube model must reference all the explicit dimensions referenced by the aggregations of the measures from the cube model's facts object. If a cube model has dimensions, an aggregation with empty list of dimensions must match to at least one dimension from the cube model. Ensure that the dimension is not referenced in other aggregations of the same measure. However, if a cube model has no dimensions, all the measures must have only aggregations with an empty list of dimensions.

Completeness rules:

- A cube model must refer to one facts object.
- A cube model must refer to at least one dimension.

Optimization rules:

- The join used to join the facts object and dimension must have a cardinality of 1:1 or Many:1 and must join a fact table to a dimension's primary table.
- At least one dimension in the cube model must have at least one hierarchy.

Facts object rules

Base rules:

- A facts object must refer to at least one measure.
- All attributes and measures referenced by a facts object must be able to be joined within the facts object. Only the joins of the facts object are considered.
- Exactly one join can be defined between any two tables within the facts object.
- Join loops are not allowed within a facts object.
- Joins referenced by a facts object must refer to the attributes of the facts object.

Dimension rules

Base rules:

- A dimension must refer to at least one attribute.
- All attributes referenced by a dimension must be able to be joined within the dimension. Only the joins of the dimension are considered.
- Join loops are not allowed within a dimension.
- Exactly one join can be defined between any two tables within the dimension.
- Joins referenced by a dimension must refer to the attributes of the dimension.
- Levels referenced by a dimension must refer to the attributes of the dimension.
- Hierarchies referenced by a dimension must refer to the levels of the dimension.

Optimization rule:

- A dimension must have one primary table to which joins attach with a 1:1 or Many:1 cardinality.

Level rules

Base rules:

- A level must refer to at least one level-key attribute.
- A level must refer to at least one default attribute.

Hierarchy rules

Base rules:

- A hierarchy must refer to at least one level.
- Exactly two levels must exist for a recursive deployment.
- Standard deployment can be used for all types of hierarchies and a recursive deployment can only be used for unbalanced hierarchies.

Measure rules

Base rules:

- Each SQL expression template can have zero or more of the following parameters: attributes, columns, and measures.
- Attributes and measures used as parameters for an SQL expression template cannot form a dependency loop.
- The SQL template of a measure cannot be an empty string.
- An SQL template cannot use aggregation functions.
- If at least one measure, and only measures are referenced, defining an aggregation is optional.
- The number of SQL templates must match the number of parameters of the first aggregation function, if an aggregation exists.
- A measure with multiple SQL templates must define at least one step in the aggregation script.
- If a measure refers to a second measure that defines multiple SQL templates, then the referring measure cannot have an aggregation script.
- A multi parameter aggregation function can only be used in the first aggregation.
- If a measure defines one or more aggregations, one aggregation must specify an empty list of dimensions.
- A measure can reference each dimension only once either within an aggregation or across aggregations.
- In an SQL template, token indicators must begin numbering with 1 and must continue numbering consecutively.
- Within an SQL expression, every column, attribute, and measure must be referenced at least once.

Attribute rules

Base rules:

- Each SQL template has zero or more of the following parameters attributes and columns.
- Attributes used as parameters for an SQL expression template cannot form a dependency loop.
- The SQL template of an attribute cannot be an empty string.

- The SQL template cannot have aggregation functions.
- In an SQL template, token indicators must begin numbering with 1 and must continue numbering consecutively.
- Within an SQL expression, every column and attribute must be referenced at least once.

Attribute relationship rules

Base rules:

- An attribute relationship must refer to two attributes.
- An attribute relationship cannot be defined as a functional dependency with a Many:Many cardinality.

Join rules

Base rules:

- A join must refer to at least one triplet (left attribute, right attribute, operator).
- A valid operation must be defined for each join triplet. The data types of the left and right attributes should be compatible with each other and with the specified operation.
- All left attributes must resolve into one or more columns of a single table.
- All right attributes must resolve into one or more columns of a single table.

Optimization rules:

- A constraint must be defined for the columns that participate in a join. If the same set of columns are used on both sides of the equality, a primary key must be defined that matches the set of columns. If different sets of columns are used on each side of the equality, one side must have a matching primary key, and the other side must have a matching foreign key and reference the primary key.
- The join cardinality must be 1:1, Many:1, or 1:Many. In a join with the same set of columns on both sides of the equality, the cardinality must be 1:1. For all other joins, the cardinality must have 1 on the side with the primary key defined, and N on the side with the foreign key defined. If the foreign key side also has a primary key defined, a 1 must be used for the cardinality of that side.
- All attributes used in the join must resolve to non-nullable SQL expressions.
- The join type must be an inner join.

Cube rules

Base rules:

- A cube must refer to one cube model.
- A cube must refer to one cube facts object.
- A cube must refer to at least one cube dimension.
- A cube facts object must be derived from the facts object used in the referenced cube model.
- All cube dimensions must be derived from the dimensions used in the referenced cube model.
- A cube must have zero or more optimization slices.
- An optimization slice must have one optimization layer for each cube dimension in the cube.
- An optimization slice must have one or more optimization levels.

- An optimization level must have one cube dimension reference and one hierarchy reference.
- An optimization level must have an **All**, **Any**, or cube level reference.
- In an optimization level object, the referenced cube dimension and cube hierarchy must be derived from objects in the cube object. The cube hierarchy must belong to the cube dimension.
- In an optimization level object, if the cube level reference is not **All** or **Any**, then the cube level must belong to the cube hierarchy.
- A cube dimension or cube hierarchy reference in an optimization level should not repeat within an optimization slice.
- An optimization slice with type molap-extract must not exist if there is another optimization slice object with type hybrid-extract, and vice-versa.
- Each cube has a maximum of one optimization slice with type molap-extract.
- Each cube has a maximum of one optimization slice with type hybrid-extract.
- An optimization slice with type drill-through can exist if there is another optimization slice with type hybrid-extract.

Cube facts object rules

Base rules:

- A cube facts object must refer to one facts object.
- A cube facts object must refer to at least one measure.
- All measures referenced by a cube facts object must also be referenced in the corresponding facts object.

Cube dimension rules

Base rules:

- A cube dimension must refer to one dimension.
- A cube dimension must refer to one cube hierarchy.
- The referenced cube hierarchy must be derived from a hierarchy used in the referenced dimension.

Cube level rules

Base rules:

- A cube level must refer to one level.
- A cube level must refer to at least one related attribute.
- All referenced attributes must be related attributes in the level.

Cube hierarchy rules

Base rules:

- A cube hierarchy must refer to one hierarchy.
- A cube hierarchy must refer to at least one cube level.
- Cube levels referenced by a cube hierarchy must be derived from levels that are part of the corresponding hierarchy.
- Cube levels in the cube hierarchy must be listed in the same order as in the corresponding levels in the referenced hierarchy. Cube level order does not apply to network hierarchies.

Chapter 3. Designing DB2 Cube Views metadata object models

This section describes the following topics:

Starting the OLAP Center
Start the OLAP Center so that you can manage your metadata objects.

OLAP Center and API compatibility
DB2 Cube Views Version 8.2 has limited backward compatibility support for OLAP Center and the metadata API.

Authorities and privileges for DB2 Cube Views
To perform tasks with the OLAP Center, you must contact your DB2 database administrator (DBA) for help in obtaining the required authorities and privileges for your operating system.

Creating DB2 Cube Views metadata objects
You can create your DB2 Cube Views metadata objects using the OLAP Center.

Removing a dimension from a cube model
You can remove a dimension from a cube model if you no longer need that dimension. You might remove a dimension without dropping the dimension, if that dimension is used by another cube model.

Dropping a metadata object from a database
You can drop a metadata object if you no longer use the metadata object in a cube model in this database.

Starting and refreshing the OLAP Center

Start the OLAP Center so that you can manage your metadata objects.

To start the OLAP Center:

1. Click **Start –> Programs –> IBM DB2 –> Business Intelligence Tools –> OLAP Center**. A database connection window opens.
2. In the database connection window, log on to the database that you want to manage metadata for.

The OLAP Center shows a snapshot in time of the metadata objects in the database. Although DB2 Cube Views always ensures the integrity of the metadata objects that it manages, the contents of the OLAP Center window are not automatically updated when metadata objects are created in the database by another OLAP Center user or by an API application. If another user or an API application changes the metadata objects, you can select **View —> Refresh** to see the new state of the database. Multiple users who work on the same metadata objects at the same time can experience errors because they might not see the most recent data in the database. Multiple users should not work on the same set of metadata objects at the same time.

OLAP Center and API Version compatibility

DB2 Cube Views Version 8.2 has limited backward compatibility support for OLAP Center and the metadata API.

DB2 Cube Views API supports the following:

- Version 8.2 API supports Describe requests from a Version 8.1 client. Describe is the only Version 8.1 operation that is supported from the previous release.

DB2 Cube Views OLAP Center supports the following:

- Version 8.2 OLAP Center does not support Version 8.1 API or Version 8.1 metadata tables.
- Version 8.1 OLAP Center does not support Version 8.2 API or Version 8.2 metadata tables.
- Version 8.1 OLAP Center cannot connect to a Version 8.2 DB2 database.
- OLAP Center supports the import of Version 8.1 XML. When you import a Version 8.1 XML file, OLAP Center migrates the XML to Version 8.2, using the Translate operation, before importing the metadata.
- OLAP Center can export both Version 8.2 and Version 8.1 XML.

DB2 Cube Views db2mdapiclient supports the following:

- Version 8.1 db2mdapiclient supports Version 8.2 API if you use Version 8.2 XML.
- Version 8.2 db2mdapiclient supports Version 8.1 API if you use Version 8.1 XML.

Authorities and privileges for using DB2 Cube Views

To perform tasks with the OLAP Center, you must contact your DB2 database administrator (DBA) for help in obtaining the required authorities and privileges for your operating system.

All references in the following tables to SELECT, INSERT, UPDATE, and DELETE privileges, unless otherwise specified, apply to tables in the DB2INFO schema for the database in which DB2 Cube Views is installed.

Windows authorities and privileges

For Windows operating systems, authorities for OLAP Center tasks apply to the user ID that is used to connect to the database.

Table 23. Windows general authorities and privileges

Task	Authorities and privileges
All OLAP Center tasks	• SYSADM or DBADM for the database in which DB2 Cube Views is installed • EXECUTE for the stored procedure DB2INFO.MD_MESSAGE within the database in which DB2 Cube Views is installed (DB2 UDB V8 only) • CONNECT for target databases
Create metadata objects	SELECT, INSERT
Modify metadata objects	SELECT, INSERT, UPDATE, DELETE
Drop metadata objects	SELECT, DELETE
Export metadata objects to an XML file	SELECT

Table 23. Windows general authorities and privileges (continued)

Task	Authorities and privileges
Import metadata objects to DB2 UDB from an XML file	SELECT, INSERT, UPDATE
Run the optimization advisor	SELECT for system tables and base tables
Run the DB2 commands that are generated by the Optimization Advisor	• CREATEIN, DROPIN on schema DB2INFO • SELECT and ALTER (or CONTROL) on base tables

Some setup and installation tasks can be done using the OLAP Center.

Table 24. Windows setup and installation authorities and privileges

Task	Authorities and privileges
All setup and installation tasks	• SYSADM for the database in which DB2 Cube Views is installed, to create the schemas for metadata objects • CONNECT, CREATETAB, IMPLICIT_SCHEMA for the database into which DB2 Cube Views is to be installed • CREATEIN, DROPIN for the schema DB2INFO within the database in which DB2 Cube Views is installed • SELECT, INSERT, UPDATE, DELETE, CONTROL for all tables in the DB2INFO schema for the database in which DB2 Cube Views is installed

AIX authorities and privileges

On AIX, there are two different types of user IDs, with their own distinct set of authorities. One type of user ID should be set up to connect to a database and run the DB2INFO.MD_MESSAGE stored procedure. This type of user ID is referred to below as the connection user ID.

The other type of user ID should be set up to run all other OLAP Center tasks. This type of user ID is referred to below as the execution user ID. This user ID is a special user ID that is set up on AIX to run FENCED stored procedures. FENCED stored procedures run under this user ID, which is set to the owner of the `.fenced` file in `sqllib/adm`. This user ID controls the system resources available to the stored procedure.

Table 25. AIX general authorities and privileges for the connection user ID

Task	Authorities and privileges
All OLAP Center tasks	• SYSADM or DBADM for the database in which DB2 Cube Views is installed • EXECUTE for the stored procedure DB2INFO.MD_MESSAGE within the database in which DB2 Cube Views is installed (DB2 UDB V8 only) • CONNECT for target databases

Table 26. AIX authorities and privileges for the execution user ID

Task	Authorities and privileges
Create metadata objects	SELECT, INSERT
Modify metadata objects	SELECT, INSERT, UPDATE, DELETE
Drop metadata objects	SELECT, DELETE
Export metadata objects to an XML file	SELECT
Import metadata objects to DB2 UDB from an XML file	SELECT, INSERT, UPDATE, DELETE
Run the optimization advisor	SELECT for system tables and base tables
Run the DB2 commands that are generated by the Optimization Advisor	• CREATEIN, DROPIN on schema DB2INFO • SELECT and ALTER (or CONTROL) on base tables

Some setup and installation tasks can be performed using the OLAP Center. These tasks require additional authorities for the connection user ID.

Table 27. AIX setup and installation authorities and privileges for the connection user ID

Task	Authorities and privileges
All setup and installation tasks	• SYSADM for the database in which DB2 Cube Views is installed, in order to create the schemas for metadata objects • CONNECT, CREATETAB, IMPLICIT_SCHEMA for the database into which DB2 Cube Views is to be installed • CREATEIN, DROPIN for the schema DB2INFO within the database in which DB2 Cube Views is installed • SELECT, INSERT, UPDATE, DELETE, CONTROL for all tables in the DB2INFO schema for the database in which DB2 Cube Views is installed

Creating DB2 Cube Views metadata objects

You can create your DB2 Cube Views metadata objects using the OLAP Center.

Exchanging metadata between DB2 Cube Views and OLAP tools

Use the OLAP Center to connect to a DB2 UDB database and import or export metadata objects.

Connecting to a DB2 database

You can connect to or disconnect from a DB2 database.

Before you connect to a DB2 database using OLAP Center, you must set up the database for use with DB2 Cube Views metadata. If the database that you try to connect to is not set up for use with DB2 Cube Views metadata, you will see a message when you try to connect. OLAP Center can perform set up tasks if the user ID that you are using to connect has the proper authorities and privileges.

Note: If you are already connected to a database and then you connect to another database, you are disconnected from the original database. All metadata objects in the original database are no longer shown in the OLAP Center object tree.

To connect to a DB2 database:

1. Open the database connection window by clicking **OLAP Center –> Connect**, and specify the necessary information.
   - In the **Database name** field, select the database that you want to connect to.
   - In the **User name** field, type the user ID for the database that you specified.
   - In the **Password** field, type the password for the user ID that you specified.
2. Click **OK**. The metadata objects in the DB2 database that you connected to are displayed in the OLAP Center object tree.

To disconnect from a DB2 database:

Click **OLAP Center–> Disconnect** from the OLAP Center main window. All metadata objects are removed from the OLAP Center object tree.

Importing metadata

You can import metadata objects into DB2 UDB so that you can manage the metadata objects using DB2 Cube Views.

Before you import metadata objects to DB2 UDB, you make sure that you have a DB2 Cube Views XML file. You can create a DB2 Cube Views XML file by using a metadata bridge to a vendor application, by exporting metadata from an existing DB2 UDB database, or by using an XML editor.

You can import Version 8.1 and Version 8.2 DB2 Cube Views XML files. The Import wizard can convert the Version 8.1 metadata objects to Version 8.2 metadata objects when the Import wizard creates the objects.

To import metadata:

1. Open the Import wizard by clicking **OLAP Center –> Import**. The Import wizard opens.
2. On the Source page, specify the XML file with the metadata information that you want to import.
3. On the Import Options page, review the list of metadata objects that will be imported, and specify import options. You can see if the object currently exists or will be created after you import the objects.
4. On the Summary page, review the decisions that you made for your metadata import.
5. Click **Finish** to import the metadata objects. The objects that you imported appear in the OLAP Center object tree.

Exporting metadata

You can export DB2 Cube Views metadata objects so that you can use the metadata objects in business partner applications.

Before you export metadata objects to a DB2 Cube Views XML file, you must create the metadata objects that you want to export.

To export metadata:

1. Open the Export window by clicking **OLAP Center –> Export**.

2. Select the version of metadata to export. If you are using a bridge to a vendor tool, check which version of the metadata that the vendor tool supports.
3. Select **All metadata objects** to export all metadata objects in the database or **Selected metadata objects** to specify a specific set of metadata objects to export. You can select one cube model, or one or more cubes with the same parent cube model. If you select one or more cubes, the parent cube model is automatically exported.
4. Specify the name of the XML file to export to. If the file already exists, it will be overwritten.
5. Click **OK**. A DB2 Cube Views XML file is created that contains information about the metadata objects that you specified. You can use the DB2 Cube Views XML file with business partner applications.

Creating a cube model using the Quick Start wizard

You can create a cube model and the corresponding facts object, measures, dimensions, attributes, and joins all at once based on your relational schema.

Before you can create a cube model and its corresponding metadata, you must define the referential integrity constraints for the tables in your database.

The Quick Start wizard will create the metadata objects that it can logically infer from the schema. You specify the fact table and measure columns, and the wizard will detect the corresponding dimensions, joins, and attributes. After you complete the Quick Start wizard, you can add, drop, and modify the metadata objects as needed.

To create a cube model and its corresponding metadata:

1. Open the Quick Start wizard by right–clicking the database or the **Cube Models** folder in the OLAP Center object tree, and clicking **Create Cube Model – Quick Start**. The Quick Start wizard opens. If the object tree does not contain a **Cube Models** folder, make sure that you are in the OLAP object view by clicking **View –> Show OLAP Objects**.

   The Introduction page describes what the Quick Start wizard can do.
2. On the Fact Table page, select the schema and fact table for your cube model. The fact table that you select directly maps to the facts object in your cube model. Columns used as foreign keys in the specified fact table will map to attribute objects in your cube model. The wizard cannot detect implicit dimensions (dimensions whose columns exist in the fact table). You can add implicit dimensions to the cube model after you complete the Quick Start wizard.
3. On the Measures page, select columns from the specified fact table that you want to map directly to measures. Primary keys and foreign keys cannot be mapped to measures and are not listed. The default aggregation function is SUM for measures based on numeric columns and COUNT for measures based on character columns. You cannot create calculated measures in this wizard; you can create calculated measures after you complete the Quick Start wizard.
4. On the Summary page, view the metadata objects that will be created. Click **Finish** to create the cube model and corresponding metadata objects. When you click **Finish**, the Quick Start wizard creates the following metadata objects in the DB2 catalog:
   - Cube model that contains all of the other metadata objects.
   - Facts object that corresponds to the fact table that you specified.
   - Measures that correspond to the fact table columns that you specified.

- Dimensions that correspond to each dimension table that is joined to the facts table. Outrigger tables are tables joined to a dimension table and are included in the appropriate dimension object.
- Attributes that correspond to each column in the dimension and outrigger tables, and to any foreign keys in the facts table.
- Join objects that serve as facts-dimension joins and joins within a dimension object that join the dimension table and any corresponding outrigger tables.

You can add hierarchies, calculated measures, and cubes to the cube model.

Creating a complete cube model

To create a complete cube model, you create an empty cube model then add a facts object, dimensions, and hierarchies and levels for each dimension in that cube model. Cube models define the relationships between the relational data in your star schema or snowflake schema so that you can optimize queries that are issued to your relational data.

You can create an empty cube model, or you can follow the steps below to create all of the objects necessary to complete the cube model. After you complete the cube model, you can create cube objects that can contain all or a subset of the cube model objects' properties.

The DB2 Cube Views cube model is a representation of a logical star schema or snow flake schema. The cube model is a grouping of relevant dimension objects around a central facts object. Each dimension can have multiple hierarchies, which increases the cube model's flexibility. The structural information about how to join the tables that are used by the facts and dimension objects is stored in the cube model. The cube model also stores enough information to retrieve OLAP data. Other reporting and OLAP tools that understand the cube model and can support multiple views of a specific dimension can benefit from multiple hierarchies defined for a dimension.

To create a complete cube model:

1. Create a cube model.
2. Create a facts object.
3. Create a dimension.
4. Create a hierarchy for a dimension.
5. Create levels for each hierarchy.

Creating a cube model

You create a cube model to group relevant dimension objects around a central facts object.

The structural information about how to join the tables that are used by the facts and dimension objects is stored in the cube model. You can create cubes with subsets of objects from a cube model to selectively expose relevant facts and dimensions to an application.

The Cube Model wizard guides you through the steps to create an empty cube model. After you create a cube model by using the wizard, add a facts object and dimensions with corresponding join objects. Dimensions can be shared among different cube models.

To create a cube model:

1. Open the Cube Model wizard by right-clicking the database or the **Cube Models** folder in the OLAP Center object tree, and clicking **Create Cube Model**. If the object tree does not contain a **Cube Models** folder, make sure that you are in the OLAP object view by clicking **View –> Show OLAP Objects**.
2. On the **Name** page, specify a name, business name, schema, and comment for the cube model.
3. Click **Finish** to create the cube model.

Creating a facts object

You create a facts object as part of an existing cube model. A facts object groups related measures that are interesting to a given application. A facts object is used in a cube model as the center of a star schema. You can create simple and calculated measures.

Before you can create a facts object, you must create a cube model.

To create a facts object:

1. Open the Facts wizard by right-clicking a cube model, and clicking **Create Facts**. If the **Create Facts** option is not available, then a facts object is already defined for the selected cube model.
2. On the Name page, specify a name, business name, and schema for the facts object. You can also type comments about the facts object. All calculated and non–calculated measures that you create in the Facts wizard will have the same schema name that you specify for the facts object.
3. On the Tables page, select one or more fact tables from the list of available tables.
4. On the Joins page, specify joins between the tables that you selected. You can select an existing join, or you can click **Create Join** to open a wizard where you can create a new join. All of the selected tables must be joined without loops. You can specify only one join between each pair of tables. If you selected only one table on the Tables page, you do not need to specify joins.
5. On the Measures page, create simple measures, calculated measures, or both.

   Simple measures
   Measures that map directly to a column. Moving a column to the selected measures list will create a simple measure that maps directly to the column.

   Calculated measures
   Measures that refer to an expression and are created from one or more columns, attributes, or other measures. To create a calculated measure, click **Create Calculated Measure**.
6. On the Aggregations page, specify a simple aggregation function for each measure. The default aggregation function is SUM for measures with a numeric data type. For measures with a character data type, the default aggregation function is COUNT. To change the aggregation, click the aggregation in the table and select a different function or none.
7. Click **Finish** to create the facts object.

Creating a dimension

You can create a dimension as part of a cube model, or you can create a dimension that you want to use at a later time.

Before you create a dimension for a cube model, you must create a facts object.

A dimension object defines a set of related attributes and joins among the attributes that together describe some aspect of a measure. When joins are necessary to group attributes, such as with a snowflake schema, the necessary joins and the attributes used in the joins are listed in the dimension definition. A dimension called Region might contain attributes like State, Country, City, and Population. Dimensions reference hierarchies that can be used to drive navigation and calculation of the dimension. Dimensions have a type that describes the nature of the dimension.

To create a dimension:

1. Open the Dimension wizard by right-clicking a cube model and clicking **Create Dimension**.
2. On the Name page, specify a name, business name, schema, and comments for the dimension. All attributes that you create in the Dimension wizard will have the same schema name that you specify for the dimension.
3. On the Tables page, select source tables from which to create the dimension. You must select at least one table.
4. On the Joins page, specify joins between the tables that you selected. You can select an existing join, or you can click **Create Join** to open a wizard so that you can create a join. All of the selected tables must be joined without loops. You can specify only one join between each set of two tables. If you selected only one table on the **Tables** page, you do not need to specify joins.
5. On the Attributes page, specify attributes to include in the dimension. You must specify at least one attribute. You can select an attribute from the list, or click **Create Calculated Attribute** to open a window so that you can create an expression. The object tree shows source tables, with their available columns and attributes. If an attribute already exists in the database that maps to one of the columns in the table, then the attribute is shown. If a column is not mapped to an attribute, then the column itself is shown. If a column is selected and moved to the **Selected attributes** list, an attribute is created that maps to the column. Select all of the attributes you want to use in the hierarchy or hierarchies of your dimension. OLAP Center will automatically add to your dimension any ID attributes used by the specified joins.
6. On the Type page, specify the type of dimension that you want to create. Select **Time** to specify that the dimension is a time dimension, or select **Regular** to specify that the dimension is not a time dimension.
7. On the Facts-to-Dimension page, specify a join object to join the dimension to the facts object. You can also create joins with the Join wizard. This page is displayed only if you are creating a dimension as part of a cube model.
8. Click **Finish** to create the dimension.

Creating a hierarchy for a dimension

A hierarchy defines relationships between two or more levels within a given dimension of a cube model. You can also define a hierarchy that uses only one level. Defining these relationships provides a navigational and computational means of traversing the specified dimension. For example, a CalendarYear hierarchy might contain levels like Year, Quarter, and Month.

Before you create a hierarchy for a dimension, you must create a dimension.

You can define multiple hierarchies for a dimension in a cube model. The relationship among the levels is determined by the hierarchy type.

To create a hierarchy for a dimension:

1. Open the Hierarchy wizard by expanding the **Dimensions** folder to see the existing dimensions. Right-click a dimension, and click **Create Hierarchy**.
2. On the Name page, specify a name, business name, schema, and any comments for the hierarchy.
3. On the Levels page, specify levels for the hierarchy and the type of hierarchy:
   a. If no levels exist in the **Available Levels** list, click **Create Level** and use the Create Level wizard to create levels.
   b. Select the levels that you want. You must select at least one level.
   c. Specify the type and deployment for the hierarchy. If you specify a **Recursive** hierarchy type, you can select only two levels.

   After you select at least one level, you can click **Show Sample** to view sample data in the hierarchy.
4. Click **Finish** to add the new hierarchy to the cube model.

Creating levels for each hierarchy

Levels define relationships between a set of related attributes. When possible, DB2 Cube Views creates functional dependencies that define the relationship between the level attributes.

1. Open the Level wizard by expanding a dimension under the **Dimensions** folder for a cube model in the OLAP Center object tree. Right-click the **Levels** folder and click **Create Level**.
2. On the Name page, specify a name, business name, schema, and comment for the cube model.
3. On the Level key attributes page, select one or more attributes that together uniquely define the level.
4. On the Default attribute page, select one attribute that describes the level's data.
5. On the Related attributes page, select zero or more attributes that provide more information about the level.
6. On the Optimize using functional dependency page, specify if you want applicable functional dependencies to be created between the level key attributes and the default and related attributes. You should select this check box only if you are sure that the level key attributes functionally determine both the default attribute and any related attributes.
7. Click **Finish** to create the level.

Adding an existing dimension to a cube model

You can add an existing dimension to a cube model. A dimension might already exist if you use the dimension in another cube model or if you used the dimension for another cube model but later removed the dimension.

Before you add a dimension, you must create a cube model and a facts object. You must also have an existing dimension that is not part of your cube model.

To add an existing dimension:

1. Open the Add Dimension wizard by right-clicking on a cube model and click **Add Dimensions**..
2. On the Dimensions page, select one or more existing dimensions to add to your cube model. You must select at least one dimension.
3. On the Facts-to-Dimension Joins page, specify a join object to join the added dimension to the facts object of your cube model.

If there is one appropriate, existing join for a dimension, that join is shown for the corresponding dimension. Make sure that the default join makes sense, that the attributes on one side of the join refer to columns in the appropriate dimension table, and the attributes on the other side of the join refer to columns in the fact table.

If there are no appropriate, existing joins or there are more than one appropriate, existing joins, you must specify a join. To specify a different join or to create a join for a specific dimension, select the dimension and click **Specify Join**.

4. Click **Finish** to add the dimension to your cube model.

Specifying a facts-to-dimension join for an existing dimension

You can specify the facts-to-dimension join for an existing dimension to a cube model.

- To specify an existing join:
  1. Select a join from the list of candidate joins.
  2. Click **OK**.
- To create a join:
  1. Click **Create Join**. The Join wizard opens.
  2. Create the join that you want. The new join will appear in the list of existing candidate joins.
  3. Select the join that you created from the list of candidate joins.
  4. Click **OK**.

Creating a join

You can create a join to join a dimension to a facts object, as part of a facts object, or as part of a dimension.

A join object joins two relational tables together. A join references attributes that then reference columns in the tables being joined. A join also has a type and a cardinality.

The simplest form of a join references two attributes; one that maps to a column in the first table and one that maps to a column in the second table, along with an operator to indicate how the columns should be compared.

A join object can also be used to model composite joins, where two or more columns from the first table are joined to the same number of columns in the second table. A composite join uses pairs of attributes to map corresponding columns together. Each pair of attributes has an operator that indicates how that pair of columns should be compared.

Join objects are primarily used in a cube model to join the cube model's dimensions to its facts object. Joins can also be used to join dimension tables together in snowflake schema or are sometimes used within a facts object to join multiple fact tables together.

To create a join:

1. Open the Join wizard.
2. On the Name page, specify a name, business name, schema, and any comments for the join.

Tip: Use a name for the join that includes the names of both tables being joined. For example, if you are joining the SalesFact table and the Product table, name your join: SalesFact–Product.

3. On the Join page, add one or more attribute pairs and select the join type and cardinality for the join. Create an attribute pair by selecting a left attribute and right attribute and then clicking **Add**. The attribute pair appears in the attribute pair table. The default join operator is =. You can change the operator by clicking the current operator in the table and selecting a new operator. The default join type is Inner, and the default cardinality is 1:1.
4. Click **Finish** to create the join.

Creating a cube

You can create a cube to specify regions of your cube model that are significant. You can also use a cube to define a subset of your data for a business-partner application.

Before you create a cube, you must create or import a cube model. The cube model must have a facts object and at least one dimension, and each dimension must have a hierarchy.

A cube is a precise definition of an OLAP cube that can be queried by using a single SQL statement. A cube is derived from an existing cube model. The cube facts and list of cube dimensions are subsets of the facts and dimensions that are in the referenced cube model. Cubes are appropriate for tools and applications that do not use multiple hierarchies because cube dimensions allow only one cube hierarchy per cube dimension.

To create a cube:

1. Open the Cube wizard by right-clicking a cube model and clicking **Create Cube**.
2. On the Name page, specify a name, business name, schema, and comments for the cube.
3. On the Measures page, select measures to include in the cube. You must select at least one measure.
4. On the Dimensions page, select cube dimensions to include in the cube. You must select at least one cube dimension. Select a cube dimension, and click the push button next to the selected cube dimension to open a window so that you can specify hierarchy information for the cube dimension.
5. On the Specify query type page, specify how you expect to use this cube. If you select **Advanced settings**, click **Specify** to specify the specific optimization slices that you expect to query most often for this cube.
6. Click **Finish** to create the cube.

Specifying a cube hierarchy and cube levels for a cube dimension

You can create customized cubes for different applications by specifying the cube levels and related attributes to include in a cube hierarchy for each cube dimension. You can specify a subset of cube levels and related attributes to reference in the cube hierarchy.

To specify a cube hierarchy for a cube dimension from Select Attributes for Cube Hierarchy window that you open from the Cube wizard:

1. Select a cube hierarchy from the **Possible hierarchies** list.

2. In the **Levels and attributes** list, ensure that the check box next to the cube level or related attribute that you want is selected. Level key attributes and default attributes are included when you select a level.
3. Click **OK** to add the cube levels and related attributes to the cube hierarchy and return to the Cube wizard.

Specifying cube optimization slices

You can specify optimization slices for a cube as an optional, but powerful way to guide the Optimization Advisor in providing summary tables that are focused on the most important regions of your cube model.

Optimization slices indicate the regions of the cube that you expect to query most often. The type of the optimization slice indicates how you expect to query that area. For example, if 50% of your queries include Month, you can define an optimization slice at the Any-Month-Any slice. The Optimization Advisor can use the optimization slices that you provide to recommend summary tables that closely match your needs.

To specify optimization slices for a cube:

1. From the Cube wizard or the Cube properties window, open the Query Type page and select **Advanced settings** then click **Specify**.
2. Define one or more optimization slices for your cube.

   Add a slice

   Click **New** to add a slice. The new slice appears in the interactive graphic and as a row in the **List of optimization slices** table. After you add a slice, modify the slice to specify the type of query and the cube level in each cube dimension that you expect to query most often.

   Modify a slice

   You can modify a slice in the interactive graphic or in the **List of optimization slices** table that is below the graphic.

   To modify a slice in the table, select the row in the table that represents the slice. Click the type or cube dimension that you want to modify and select an option.

   To modify a slice in the interactive graphic, change the levels by dragging the node in the cube dimension to a level and change the type by right-clicking on the slice and selecting the query type.

   You can select one of the following options for each cube dimension:

   - Select **Any** if you do not have a preference as to which cube level the slice should be at. The Optimization Advisor will determine which cube level to optimize for.
   - Select **All** if you often query your data at the highest aggregation level, such as for all products or all regions.
   - Select a specific cube level that is defined for the cube dimension if you know which cube level many of your queries use. For example, you can define an optimization slice at the Any-Month-Any cube levels, where you select **Any** in the Market cube dimension, **Month** in the Time cube dimension, and **Any** in the Product cube dimension.

   Recommendation: For best results, define only a small number (three or less) of focused slices per cube.

Remove a slice
You can select a slice in the table or in the interactive graphic and click **Remove** to delete the slice.

Removing a dimension from a cube model

You can remove a dimension from a cube model if you no longer need that dimension. You might remove a dimension without dropping the dimension, if that dimension is used by another cube model.

When a dimension is removed from a cube model, the following actions apply:

- The dimension is removed only from the cube model that you selected, but it is kept in any other cube models that are referencing it.
- Any cube dimensions based on this dimension in the cube model are removed from corresponding cubes.
- The dimension is not dropped from the database.
- The dimension is available in the **All Dimensions** folder.

To remove a dimension from a cube model:

1. In the OLAP Center object tree, expand the **Cube Models** folder to see the existing cube models. If the object tree does not contain a **Cube Models** folder, make sure that you are in the OLAP object view by clicking **View —> Show OLAP Objects**.
2. Expand a cube model node to see the object categories that are contained in the cube model.
3. Expand the **Dimensions** folder to see the existing dimensions.
4. Right-click a dimension, and click **Remove**.

Dropping a metadata object from a database

You can drop a metadata object if you no longer use the metadata object in a cube model in this database.

You can drop most objects from the OLAP object view or the Relational object view. You can drop objects only if they are not being referenced by other objects.

To drop a metadata object from a database:
Select one or more objects in the OLAP Center object tree and right-click the selected objects, then click **Drop**. If the menu does not contain a **Drop** menu item, you cannot drop the selected object. When you drop a parent object, all of child objects are dropped. For example, dropping a dimension will also drop all of the corresponding hierarchies.

Chapter 4. DB2 Cube Views business modeling scenarios

This section describes the following topics:

Calculating the flow and value of the stock in a warehouse over time
A retail business, XYZ Retail, keeps its stock in a warehouse before the stock is sent to a particular store to be sold. XYZ Retail maintains data about the state of the stock in the warehouse over time, and wants to analyze this data.

Correlating advertising costs to sales
A car dealership is considering increasing its advertising spending. To make an educated decision, the dealership first wants to analyze the historical relationship between advertising spending and sales. The dealership is interested in determining if varying levels of advertising have affected sales, and in particular if increased advertising is closely associated with increased sales.

Calculating the profit and profit margin of a store
The general manager of a toy store wants to be able to analyze how various factors, such as time of year and type of product, affect the profit and profit margin.

Counting the number of Internet orders
A retail company expanded its business by adding Internet sales a few years ago. Now the company wants to analyze the impact of the Internet sales. One of the first calculations that the company needs is the number of orders completed over the internet.

Ranking sales figures
An office supply store chain has expanded rapidly over the last several years. The business executives are considering closing some of the lowest performing stores to cut costs and increase profits.

Using time data stored in the fact table to create a Time dimension
A retail business, XYZ Retail, is modeling their sales transaction data in DB2 Cube Views so that they can analyze their data more effectively. However, because of the transactional nature of the data, the only time information that is available is a date that is associated with each transaction.

Calculating the flow and value of the stock in a warehouse over time

A retail business, XYZ Retail, keeps its stock in a warehouse before the stock is sent to a particular store to be sold. XYZ Retail maintains data about the state of the stock in the warehouse over time, and wants to analyze this data.

In particular, the company wants to examine two aspects of its warehouse:

- The flow of merchandise in to and out of the warehouse
- The value of the merchandise in the warehouse at a given time

The first aspect, the flow of the merchandise involves looking at data over time. The second aspect, the value of the merchandise takes a snapshot of the warehouse at a particular moment in time.

Details of the scenario

XYZ Retail has a fact table with the following warehouse-related columns: QUANTITY_IN, QUANTITY_OUT, CURRENT_QUANTITY, PRODUCT_VALUE, PRODUCT_ID, and TIME_ID. This data is entered in the table on a weekly basis. The database also has a Product table and Time table. For example, a set of sample fact table data is shown in Table 28.

Table 28. Sample fact table data

PRODUCT _ID	TIME_ID	QUANTITY _ID	QUANTITY _OUT	CURRENT _QUANTITY	PRODUCT _VALUE
1234	1	5	0	5	5
1234	2	20	10	15	5
1234	3	10	20	5	5

The PRODUCT_ID value for each of the three sample data entries is the same because one product type can move in and out of the warehouse multiple times.

The DBA for XYZ Retail must create three different measures:

Flow In
Models the flow of merchandise into the warehouse.

Flow Out
Models the flow out of the warehouse.

Current Value
Models the value of the merchandise at a given time.

To create the first two measures, Flow In and Flow Out, the DBA creates measures that map to the QUANTITY_IN and QUANTITY_OUT columns respectively, and sum the data across all dimensions. This is known as a fully additive measure because the data is aggregated using only the SUM function across all dimensions. For example, Table 29shows a set of sample data for the QUANTITY_IN and QUANTITY_OUT columns for three months for the product with a PRODUCT_ID of 1234. The Flow In and Flow Out measures sum these monthly values to calculate the total quantities that came in and out of the warehouse over the quarter.

Table 29. Calculation of the sample data for the Flow In and Flow Out fully additive measures for PRODUCT_ID 1234

	January	February	March	Quarter 1
QUANTITY_IN	5	20	10	35
QUANTITY_OUT	0	10	20	30

Fully additive measures are the simplest and most common measures to create and are often used as the building blocks for more complex measures. For measures based on numeric source data, the OLAP Center creates a fully additive measure by default.

To create the third measure, Current Value, the DBA creates a calculated measure that computes the value by multiplying PRODUCT_VALUE by CURRENT_QUANTITY. For example, if the value of the product with PRODUCT_ID=1234 is 5, then the Current Value measure for the sample data is

shown in Table 30.

Table 30. Calculation of the sample data for the Current Value measure for PRODUCT_ID 1234

	January	February	March
CURRENT_QUANTITY	5	10	20
Current Value	25	50	100

This data must then be aggregated across the dimensions. However, because this measure is calculating the value at a specific point in time, it does not make sense to sum across the time dimension. Instead, the aggregation will sum the data across the Product dimension, and find the average of the data over time. This is known as a semi-additive measure because only part of the aggregation involves the SUM function.

Measures that calculate snapshot data, data that represents a particular moment in time such as month inventory data, are often semi-additive measures because it does not make sense to add the months into quarters. If a product remains in the warehouse for an entire quarter, that product is included in the CURRENT_QUANTITY snapshot data of the warehouse inventory each of the three months of the quarter. If the CURRENT_QUANTITY data is summed over time, the product that sat in the warehouse for three months is counted three times. As shown in Table 31, the value 25 for Quarter 1 has no significance to the warehouse's activities. The table shows that the warehouse never had 25 products in the warehouse, so calculating the value of 25 products has no meaning.

Table 31. Calculation of the sample data for the CURRENT_QUANTITY column using the SUM function for the time dimension for PRODUCT_ID 1234

	January	February	March	Quarter 1
SUM(CURRENT_ QUANTITY)	5	15	5	25

Instead of using the SUM function across all dimensions, you can perform other aggregation functions such as AVG, MIN, and MAX for the time dimension. For example, with the same set of sample data for January, February, and March, you can use a second aggregation function for the time dimension as shown in Table 32 to create meaningful values for the Quarter. The Current Value measure can represent the average total value of merchandise stored in the warehouse over the quarter, or the maximum or minimum value at any point in time during the quarter.

Table 32. Calculation of the sample data for the CURRENT_QUANTITY column using the AVG, MAX, and MIN functions for the time dimension for PRODUCT_ID 1234

	January	February	March	Quarter 1
AVG(CURRENT_ QUANTITY)	5	15	5	8.3
MAX(CURRENT_ QUANTITY)	5	15	5	15
MIN(CURRENT_ QUANTITY)	5	15	5	5

Steps to create measures

The following steps explain how you could use the OLAP Center Facts Properties window to create the Flow In, Flow Out, and Current Value measures in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. Create the Flow In measure:
   a. On the Measures page, click **Create Calculated Measure** to create the Flow In measure. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `FLOW IN` in the **Name** field.
   c. To create the flow in expression, complete the following steps:
      - Expand the **Columns** folder and the fact table in the Data list.
      - Double-click the **QUANTITY_IN** column to add it to the expression.
      - Click **OK** to close the SQL Expression Builder window. You do not need to change the default aggregation function, SUM, on the Aggregations page. The SUM function is the default for the Flow In measure because the data source is numeric and the measure refers to a column, not to only existing measures.
3. Create the Flow Out measure:
   a. On the Measures page, click **Create Calculated Measure** to create the Flow Out measure. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `FLOW OUT` in the **Name** field.
   c. To create the flow out expression, complete the following steps:
      - Expand the **Columns** folder and the fact table in the **Data** list.
      - Double-click the **QUANTITY_OUT** column.
   d. Click **OK** to close the SQL Expression Builder window. You do not need to change the default aggregation function, SUM, on the Aggregations page. The SUM function is the default for the Flow Out measure because the data source is numeric and the measure refers to a column, not to only existing measures.
4. Create the Current Value measure:
   a. On the Measures page, click **Create Calculated Measure** to create the Current Value measure. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `CURRENT VALUE` in the **Name** field.
   c. To create the Current Value expression, complete the following steps:
      - Expand the **Columns** folder and the fact table in the **Data** list.
      - Double-click the **PRODUCT_VALUE** column in the **Data** list.
      - Double-click the * operator in the **Operators** list.
      - Double-click the **CURRENT_QUANTITY** column in the **Data** list.

      Figure 12 on page 61 shows the Current Value expression that you can create in the SQL Expression Builder window.

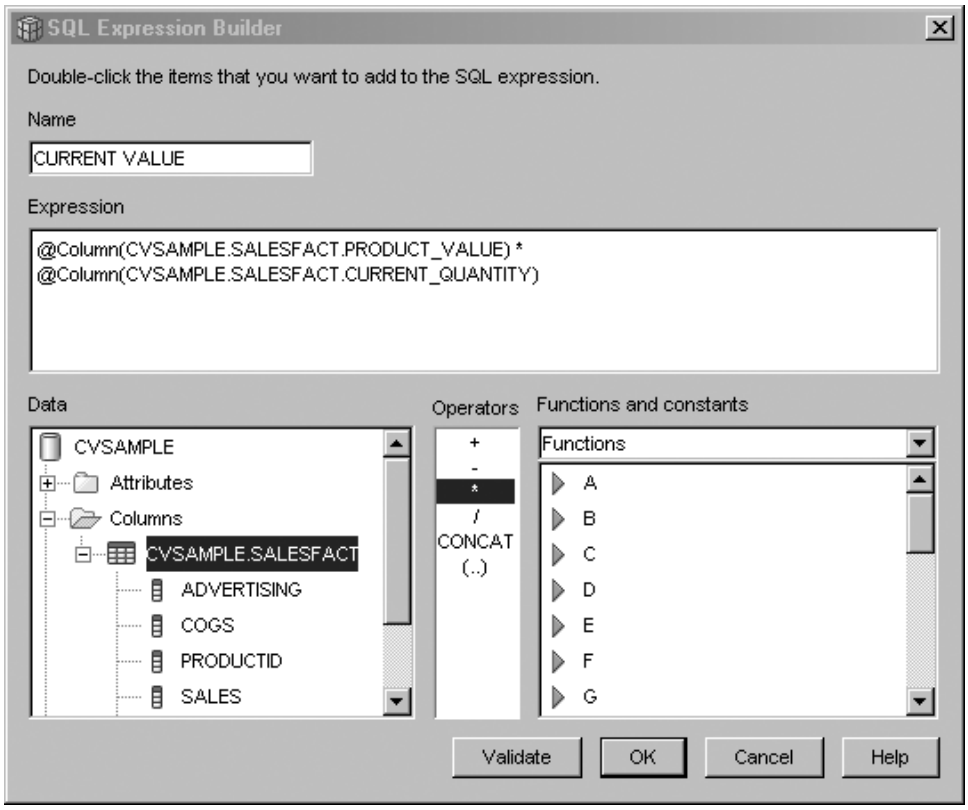

Figure 12. Complete the Current Value expression in the SQL Expression Builder window

d. Click **OK** to close the SQL Expression Builder window.

e. On the Aggregations page, click the aggregation for the Current Value measure, and click **Aggregation script** from the list. The Aggregation Script Builder window opens. The default aggregation script has the SUM function used for all dimensions.

f. If necessary, move the Time dimension down by selecting **Time** and clicking the  move down push button, so that it is the last dimension listed in the script. Then with the Time dimension selected, double-click the **AVG** function in the **Column functions** list. The aggregation script, as shown in Figure 13 on page 62, sums the data across all of the dimensions except Time, over which the data is averaged.

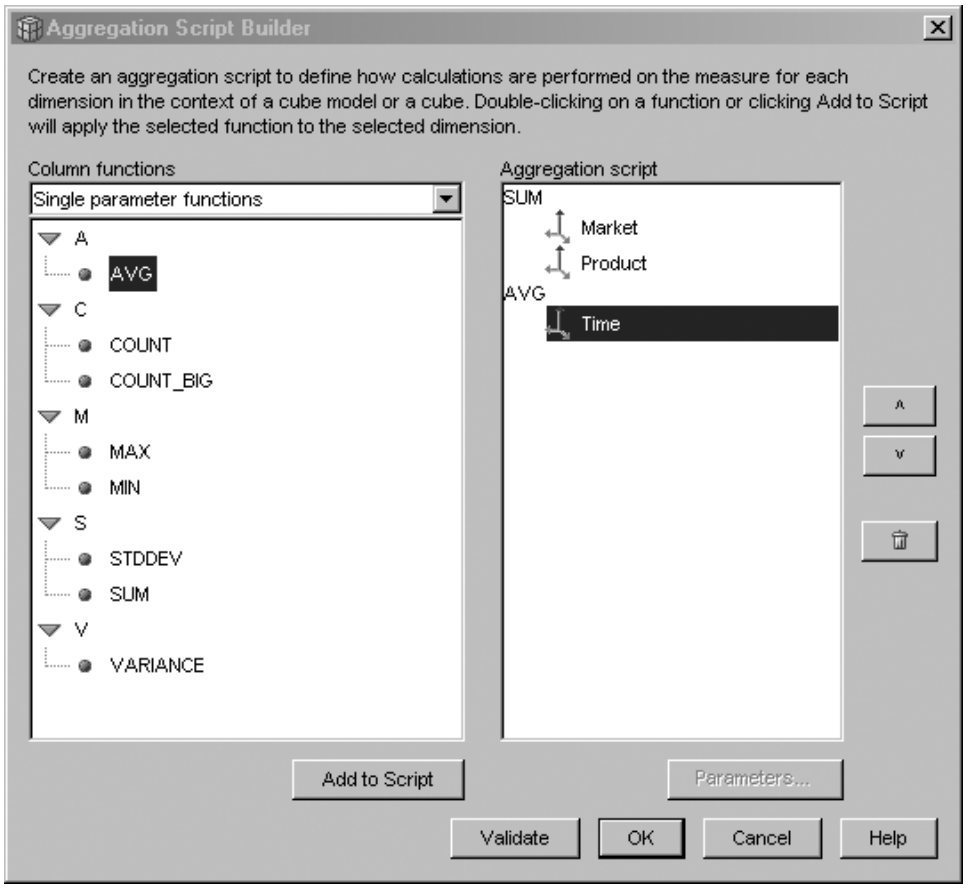

Figure 13. Aggregation script for the Current Value measure

g. In the Aggregation Script Builder window, click **Validate** to verify that the aggregation script is valid. Click **OK** to save the aggregation script and close the window.

5. Click **OK** to save the changes to the facts object and to close the Facts Properties window.

You now have three calculated measures for the stock in the warehouse. You can use these measures to analyze the patterns of product flow in and out of your warehouse.

Correlating advertising costs to sales

A car dealership is considering increasing its advertising spending. To make an educated decision, the dealership first wants to analyze the historical relationship between advertising spending and sales. The dealership is interested in determining if varying levels of advertising have affected sales, and in particular if increased advertising is closely associated with increased sales.

Details of the scenario

The dealership's database has a fact table with Sales and Ad Costs columns. The database also has several other dimension tables. The DBA can create a measure that uses the DB2 CORRELATION function to perform correlation calculations between the costs and the sales. The CORRELATION function is a multiparameter

function that requires two input parameters. In this case, the DBA will use the Sales and Ad Costs columns as the two input parameters.

The DBA must apply the multiparameter aggregation function first in the aggregation script. The multiparameter function can be applied across all of the dimensions, or it can be applied first to all dimensions except for the Time dimension, and a second function, such as the MAX function, can be applied to the Time function. The DBA defines the SQL expression for the measure so that it maps directly to the Ad Costs column. The SQL expression is the first of the two parameters used in the multiparameter function. The DBA defines the second parameter as an SQL expression that maps directly to the Sales column. The CORRELATION function is defined as the only aggregation function so that the measure can calculate the statistical correlation between advertising costs and sales results across all dimensions.

Steps to create measure

The following steps explain how you could use the OLAP Center Facts Properties window to create the Advertising-Sales Correlation measure in an existing facts object:

1. Open the Facts Properties window by right-clicking the facts object in the OLAP Center object tree, and clicking **Edit Measures**.
2. Click the **Create Calculated Measure** push button. The SQL Expression Builder window opens.
3. In the SQL Expression Builder window, type `ADVERTISING-SALES CORRELATION` in the **Name** field.
4. Define the measure's expression that will also be used as the first parameter of the multiparameter CORRELATION function in the aggregation script. To define the expression, expand the **Measures** folder in the **Data** list and double-click the **AD COSTS** measure to add it to the **Expression** list. Figure 14 on page 64 shows that the expression that you create in the SQL Expression Builder window.

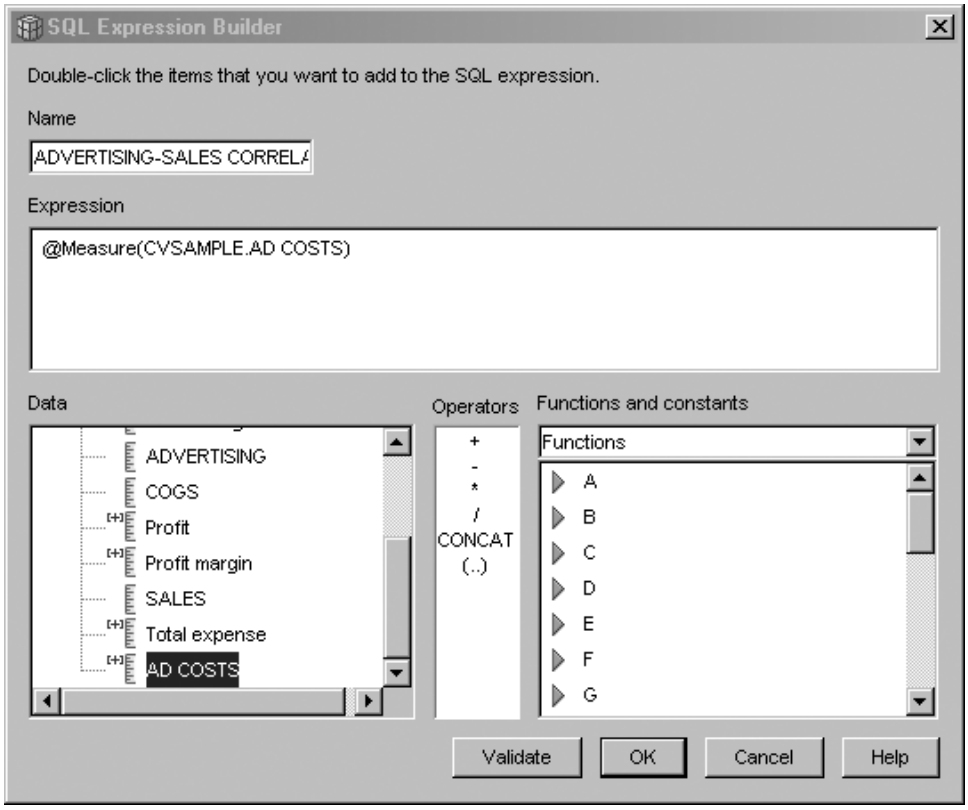

Figure 14. Complete advertising-sales correlation expression in the SQL Expression Builder window

5. On the Aggregations page, click the aggregation function for the **ADVERTISING-SALES CORRELATION** measure and select **Aggregation script** as shown in Figure 15 on page 65. The Aggregation Script Builder window opens.

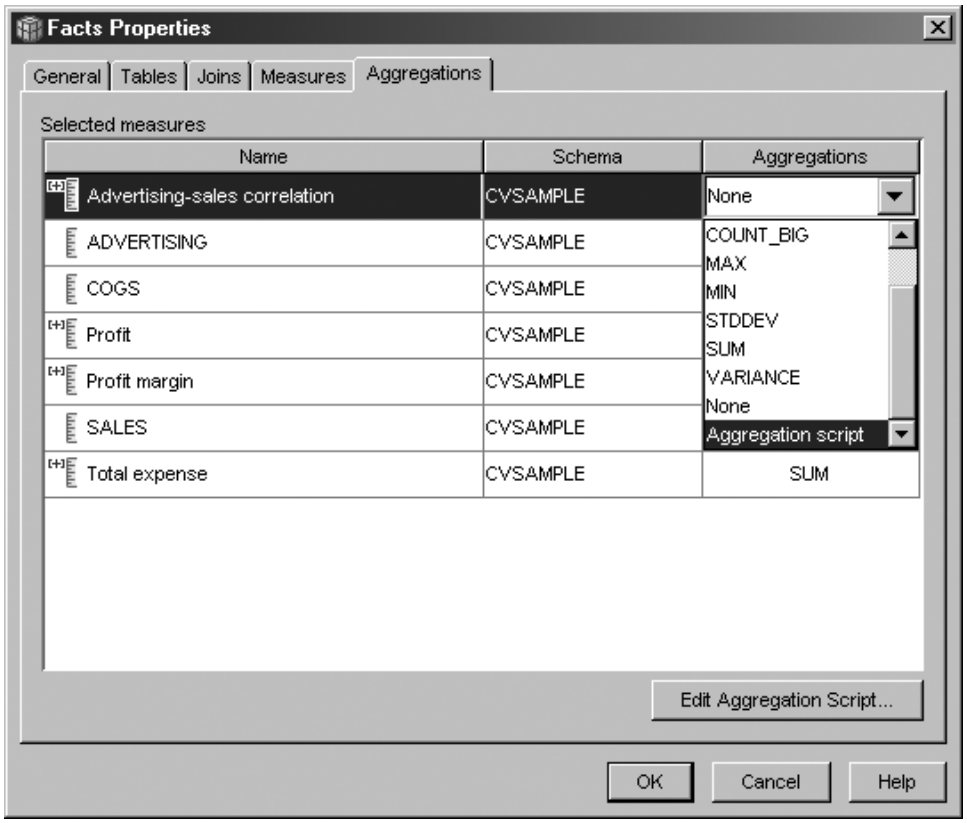

Figure 15. Aggregations page of the Facts Properties window

6. In the **Column functions** field, select **Multiparameter functions**. In the list of multiparameter functions, select the **CORRELATION** function and click **Add to Script**. The Function Parameters window opens.
7. Select **Using existing measure** and select **SALES** from the list as shown in Figure 16 on page 66.

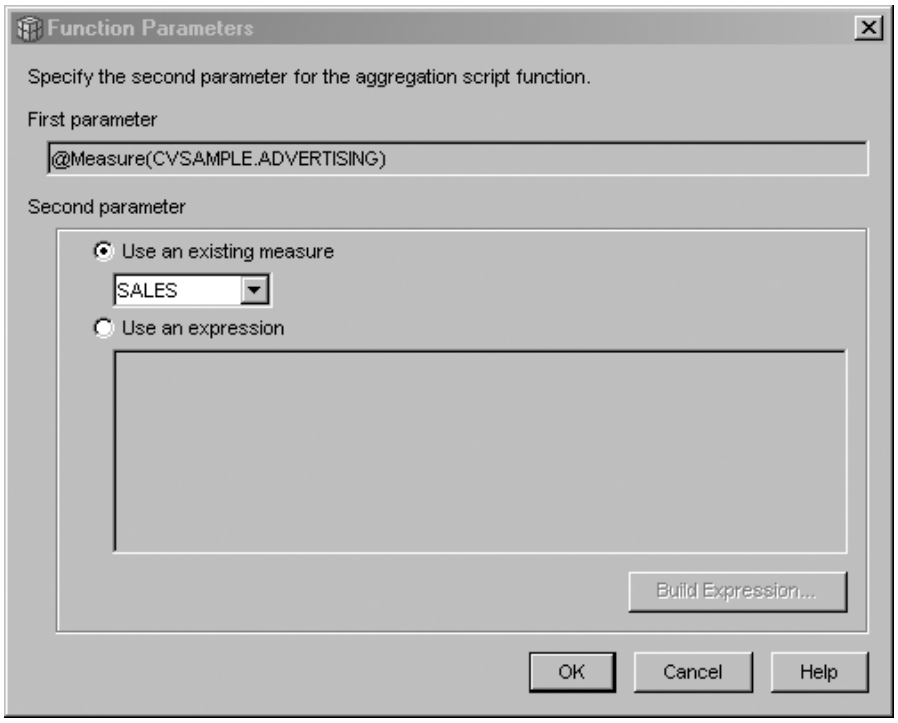

Figure 16. Sales measure specified as the second parameter in the Function Parameters window

8. Click **OK** to save your selection and close the Function Parameters window.
9. Figure 17 on page 67 shows that the CORRELATION function is at the top of the list of dimensions in the script.

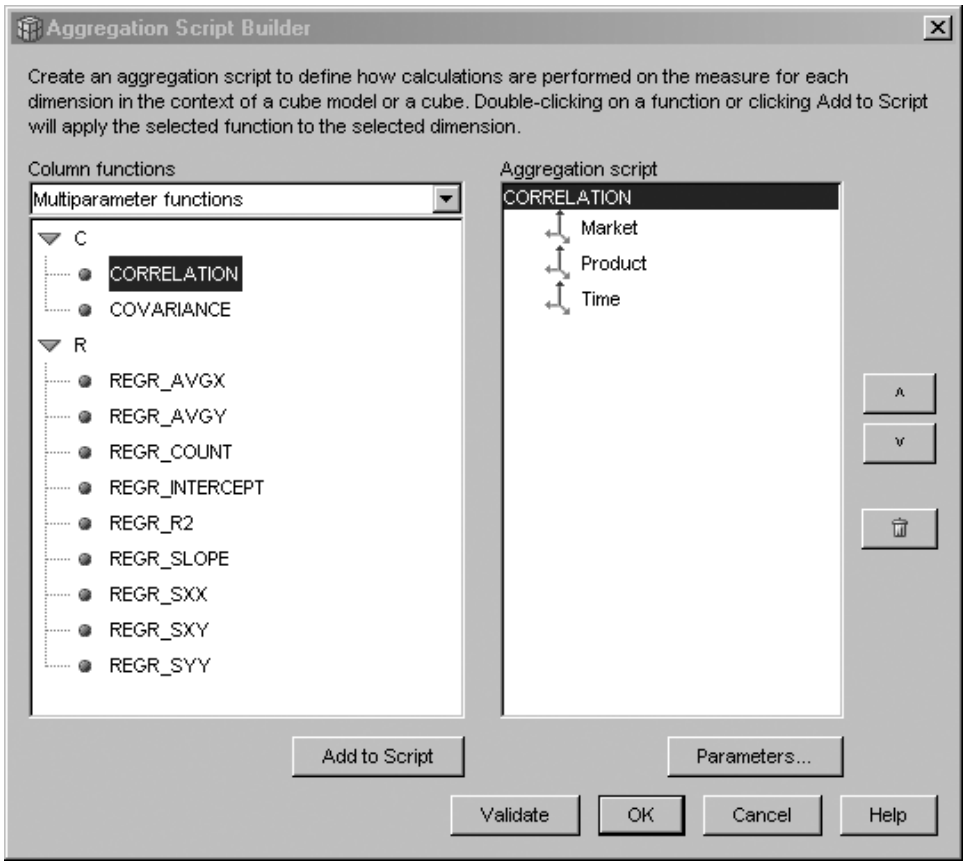

Figure 17. Aggregation script for the Advertising-Sales Correlation measure

10. In the Aggregation Script Builder window, click the **Validate** push button to verify that the aggregation script is valid.
11. Click **OK** to save the aggregation script and close the window.
12. Click **OK** to save the changes to the facts object and to close the Facts Properties window.

You now have a measure that correlates two types of data in your database. You can use this measure to make decisions on future advertising spending based on historical trends in results.

Calculating the profit and profit margin of a store

The general manager of a toy store wants to be able to analyze how various factors, such as time of year and type of product, affect the profit and profit margin.

Before the more advanced analysis can be completed, the DBA for the toy store must first create profit and profit margin measures. Then the DBA can create additional measures that correlate and compare different factors to the profit and profit margin measures.

Details of the scenario

The toy store's database has a fact table with Sales, Costs of Goods Sold (COGS), and Expense columns, in addition to corresponding foreign key columns for each

of the several dimension tables. The DBA already created Sales, COGS, and Expense measures that map to the Sales, COGS, and Expense columns, respectively. The Profit and Profit Margin measures can be created entirely from these existing measures.

To create the Profit measure, the DBA creates a measure that calculates SALES-(COGS+EXPENSE) in the SQL Expression, and sums the calculated data across all dimensions. The Profit measure can be created by referencing existing measures, or columns, or a combination of both.

After the Profit measure is created, the DBA can create the Profit Margin measure. The Profit Margin measure is a ratio of two existing measures expressed as a percentage, (Profit / Sales)*100, and does not require its own aggregation function. An aggregation function is not required because the measure refers only to other measures whose data is already aggregated. If the DBA uses a composite measure, a measure that only references other measures, to calculate a ratio, the DBA does not need to define an additional aggregation. Most aggregation functions, such as SUM, do not make sense with ratios. For example, if the toy store has profit margins of 40%, 32%, 28%, and 37% for four consecutive quarters, summing the ratios over time, would result in a profit margin of 137% for the year, which does not make sense.

Steps to create the measures

The following steps explain how you could use the OLAP Center Facts Properties window to create the Profit and Profit Margin measures in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. Create the Profit measure:
   a. On the Measures page, click the **Create Calculated Measure** push button. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `PROFIT` in the **Name** field.
   c. To create the Profit expression, expand the **Measures** folder in the **Data** list and complete the following steps:
      - Double-click the **SALES** measure in the **Data** list to add it to the expression.
      - Double-click the **-** operator in the **Operators** list.
      - Double-click the **COGS** measure in the **Data** list.
      - Double-click the **+** operator in the **Operators** list.
      - Double-click the **EXPENSE** measure in the **Data** list.
      - In the **Expression** field, highlight the part of the expression that reads: `@Measure(CVSAMPLE.COGS)+@Measure(CVSAMPLE.EXPENSE)` and double-click the **(..)** operator from the **Operators** list to enclose the selected part of the expression in parenthesis.

      Figure 18 on page 69 shows the profit expression that you can create in the SQL Expression Builder window.

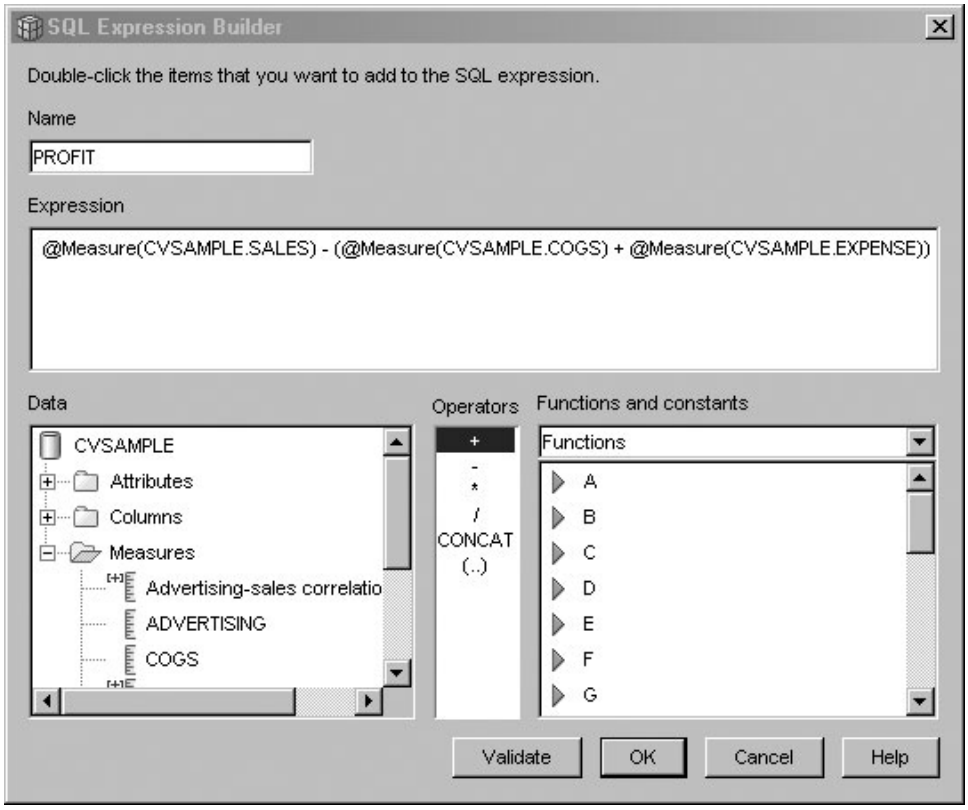

Figure 18. Complete profit expression in the SQL Expression Builder window

d. Click **OK** to create the Profit measure and close the SQL Expression Builder window.

e. On the Aggregations page, click the aggregation for the Profit measure, and select the SUM function. The Profit measure is complete.

3. Create the Profit Margin measure:

   a. On the Measures page, click **Create Calculated Measure**. The SQL Expression Builder window opens.

   b. In the SQL Expression Builder window, type `PROFIT MARGIN` in the **Name** field.

   c. To create the Profit Margin expression, expand the **Measures** folder in the **Data** list and complete the following steps:

      - Double-click the **PROFIT** measure in the **Data** list to add it to the expression.
      - Double-click the **/** operator from the **Operators** list.
      - Double-click the **SALES** measure in the **Data** list.
      - Enclose the entire expression in parentheses by typing in the **Expression** field.
      - Position the cursor at the end of the expression and double-click the * operator from the **Operators** list.
      - Type `100` at the end of the expression in the **Expression** field.

      Figure 19 on page 70 shows the profit margin expression that you can create in the SQL Expression Builder window.

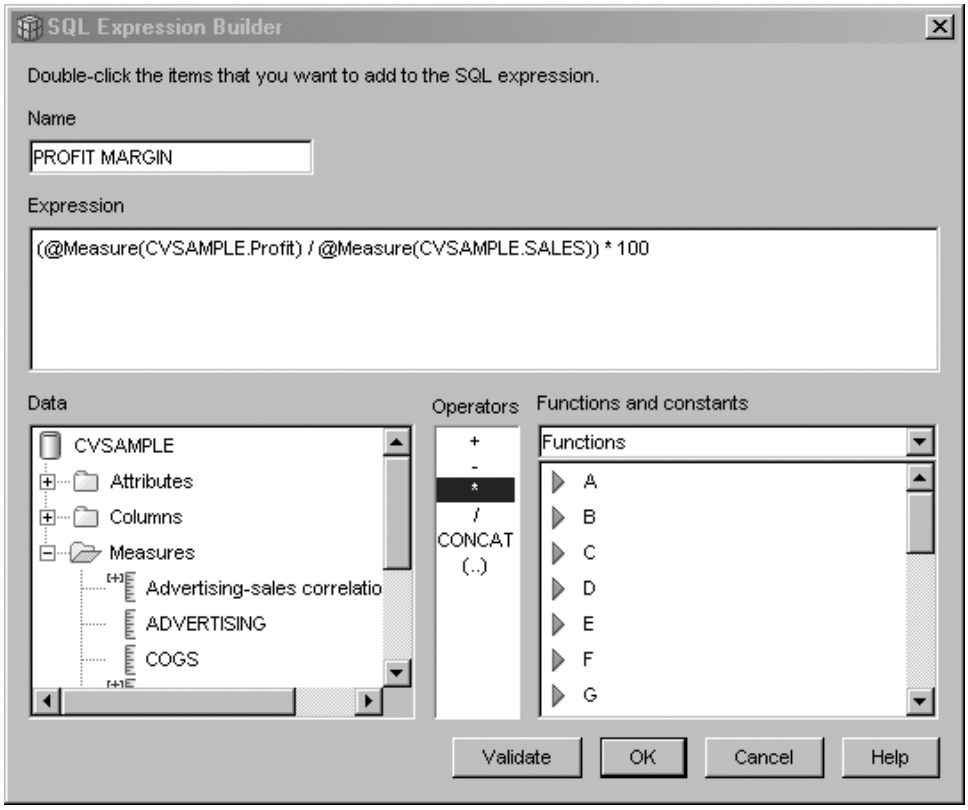

Figure 19. Complete profit margin expression in the SQL Expression Builder window

d. Click **OK** to create the Profit Margin measure and close the SQL Expression Builder window.

   On the Aggregations page, the OLAP Center sets the aggregation function to NONE by default for composite measures, so you do not need to change the aggregation function.

4. Click **OK** to close the Facts Properties window and save the two new measures that you added to the facts object.

After the DBA creates these two measures, additional analysis can be completed with respect to these important measures.

Counting the number of Internet orders

A retail company expanded its business by adding Internet sales a few years ago. Now the company wants to analyze the impact of the Internet sales. One of the first calculations that the company needs is the number of orders completed over the internet.

Details of the scenario

The company's database has a fact table for Internet orders with ORDER_ID, PRODUCT_ID, QUANTITY, and TIME_ID columns. The PRODUCT_ID column includes each product sold in a corresponding order, and the QUANTITY column stores the quantity of the product purchased in the order. Orders with more than one product have as many row entries as the number of unique products sold in the order. For example, Table 33 on page 71 shows three orders, where Order 1 included three Product As, one Product O and one Product G.

Table 33. Partial fact table contents

ORDER_ID	PRODUCT_ID	QUANTITY
1	A	3
1	O	1
1	G	1
2	L	1
2	Q	2
3	P	5

The DBA can create an Order Count measure that counts each unique entry in the ORDER_ID column. The Order Count measure is defined using the DISTINCT keyword in the SQL expression, and the COUNT function for the aggregation across all of the dimensions. The measure's SQL expression will create a list of distinct orders, which are counted during the aggregation. Because the measure does not involve any summing, it is called a non-additive measure.

Non-additive measures are also useful when you have character data or other data that you want to count. For example, you might use non-additive measures to count the number of postal codes that you shipped products to.

In this example, the DBA decided to define an Order ID measure that maps directly to the ORDER_ID column. However, you can choose to use the ORDER_ID column in the same way. The default aggregation is different based on if a column or a measure is used in the SQL expression, but in either case, you need to change the default aggregation to the COUNT function, as described in "Steps to create measures."

Steps to create measures

The following steps explain how you could use the OLAP Center Facts Properties window to create the Order Count measure in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. On the Measures page, click the **Create Calculated Measure** push button. The SQL Expression Builder window opens.
3. In the SQL Expression Builder window, type `ORDER COUNT` in the **Name** field.
4. To create the order count expression, expand the **Measures** folder in the **Data** list and complete the following steps:
   - In the **Functions and constants** filed, select **Miscellaneous**. In the list of miscellaneous functions and constants, double-click the **DISTINCT** keyword.
   - Double-click the **ORDER ID** measure in the **Data** list.

   Figure 20 on page 72 shows the order count expression that you can create in the SQL Expression Builder window.

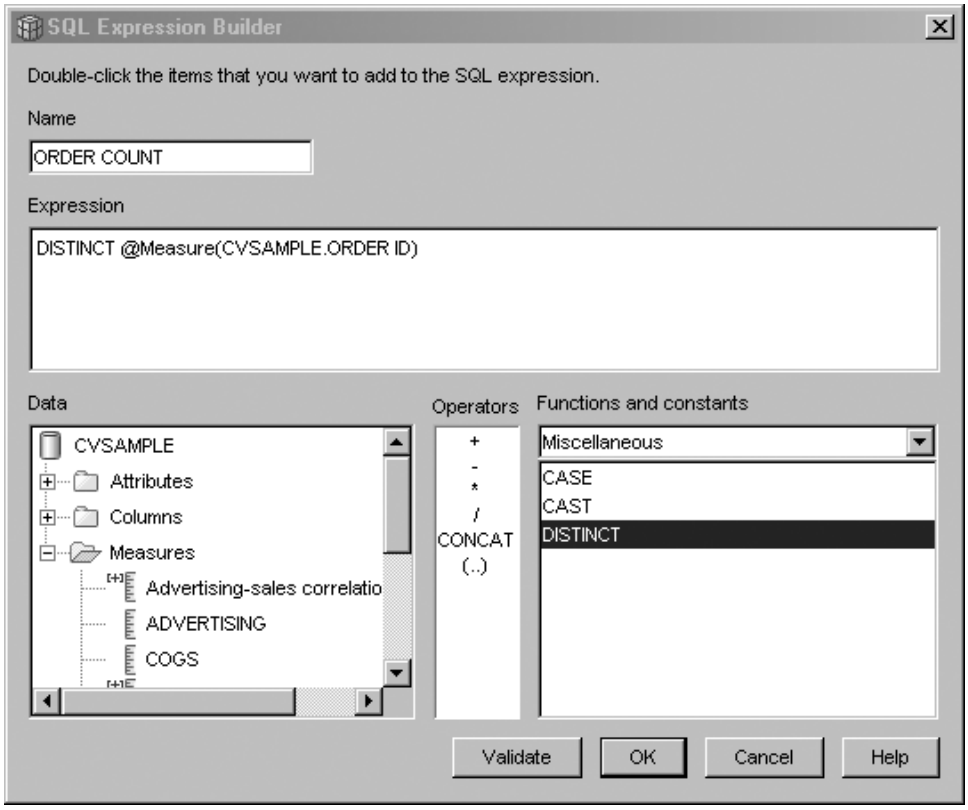

Figure 20. Complete order count expression in the SQL Expression Builder window

5. Click **OK** to close the SQL Expression Builder window.
6. On the Aggregations page, click the aggregation for the Order Count measure, and select the **COUNT** function.
7. Click **OK** to save the changes to the facts object and to close the Facts Properties window.

You now have the measure that counts the number of distinct Order ID row values. You can use this measure in conjunction with other measures to further analyze your data.

Ranking sales figures

An office supply store chain has expanded rapidly over the last several years. The business executives are considering closing some of the lowest performing stores to cut costs and increase profits.

The sales history for a store is an important factor in deciding to close a store. Analysts need to be able to rank the sales figures and drill-down across the dimensions to complete their analysis.

Details of the scenario

The office supply store's database has a fact table with a Sales column in addition to other columns. The database also has several dimension tables. The DBA can create a Sales Rank measure that uses the RANK function, which is an OLAP function provided by DB2 Universal Database (DB2 UDB).

DB2 Cube Views supports the following OLAP functions that are provided by DB2 UDB:

RANK

Orders the rows and assigns a ranking to each row. The rank is defined as 1 plus the number of preceding rows that are distinct with respect to the ordering. If the relative order of two or more rows cannot be determined because they have duplicate row values, the same rank number is assigned. The ranking results might have gaps in the numbers if there are duplicate row values. Table 34 on page 74 shows an example of what the ranking results are from the RANK function for a set of sample row values.

The typical syntax for the RANK function is:

```
RANK ( ) OVER (ORDER BY sort-key-expression expression-order)
```

where *sort-key-expression* is the set of data to be ranked, and *expression-order* is a keyword, **ASC** or **DESC**, that orders the values of the sort-key-expression in ascending or descending order. DB2 Cube Views requires that the *sort-key-expression* be an existing measure, not a column or attribute. Also, DB2 Cube Views does not support the PARTITION BY clause that is provided by DB2 UDB with this function. More information about the RANK function is available in the DB2 UDB Information Center.

DENSERANK

Orders the rows and assigns a ranking to each row. The rank of the row is defined as 1 plus the number of rows that strictly precede the row. Therefore, the ranking results will be sequential and without gaps in the rank numbering. Table 34 on page 74 shows an example of what the ranking results are from the DENSERANK function for a set of sample row values.

The typical syntax for the DENSERANK function is:

```
DENSERANK ( ) OVER (ORDER BY sort-key-expression expression-order)
```

where *sort-key-expression* is the set of data to be ranked, and *expression-order* is a keyword, **ASC** or **DESC**, that orders the values of the sort-key-expression in ascending or descending order. DB2 Cube Views requires that the *sort-key-expression* be an existing measure, not a column or attribute. Also, DB2 Cube Views does not support the PARTITION BY clause that is provided by DB2 UDB with this function. More information about the DENSERANK function is available in the DB2 UDB Information Center.

ROWNUMBER

Computes the sequential row number of the row by the ordering, starting with 1 for the first row. If the ORDER BY clause is not specified, the row numbers are assigned to the rows in arbitrary order.

The typical syntax for the ROWNUMBER function is:

```
ROWNUMBER ( ) OVER ([ORDER BY sort-key-expression expression-order])
```

where *sort-key-expression* is the set of data to be ranked, and *expression-order* is a keyword, **ASC** or **DESC**, that orders the values of the sort-key-expression in ascending or descending order. DB2 Cube Views requires that an existing measure, not a column or attribute, be used as the data source for this function. Also, DB2 Cube Views does not support the

PARTITION BY clause that is provided by DB2 UDB with this function. More information about the ROWNUMBER function is available in the DB2 UDB Information Center.

These OLAP functions are not listed in the SQL Expression Builder Functions and constants list.

Table 34. Ranking results for a sample set of values using the RANK and DENSERANK functions

Row values	Ordering	Ranking results from the RANK function	Ranking results from the DENSERANK function
100	1	1	1
35	2	2	2
23	3	3	3
8	4	4	4
8	4	4	5
6	5	6	6

Steps to create the measure

The following steps explain how you could use the OLAP Center Facts Properties window to create the Sales Rank measure in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. On the Measures page, click **Create Calculated Measure** to create the Sakes Rank measure. The SQL Expression Builder window opens.
3. In the SQL Expression Builder window, type `SALES RANK` in the **Name** field.
4. To create the Sales Rank expression, complete the following steps:
   - Type the following function syntax in the Expression field: `RANK ( ) OVER (ORDER BY measure DESC)`.
   - Expand the **Measures** folder in the **Data** list.
   - Highlight the word `measure` in the function syntax in the **Expression** field and double-click the **SALES** measure to add the SALES measure to the expression.

   The final expression is shown in Figure 21 on page 75.

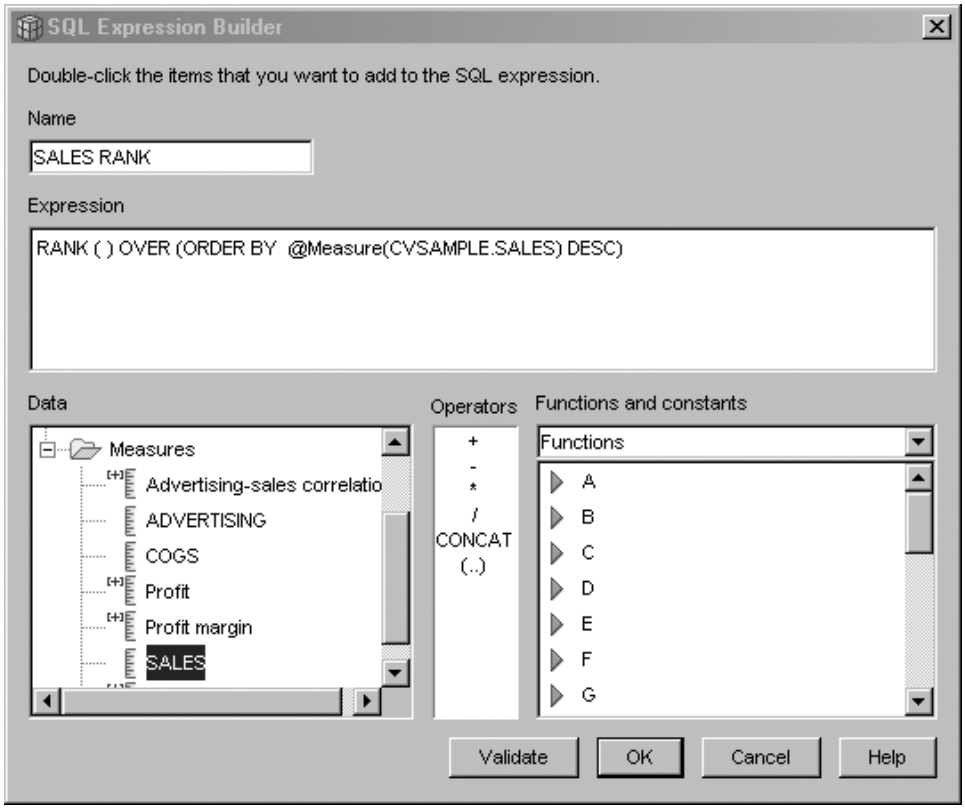

Figure 21. Complete Sales Rank expression in the SQL Expression Builder window

5. Click **Validate** to ensure that the expression is valid. Click **OK** to close the SQL Expression Builder window.

   You do not need to change the default aggregation, None, on the Aggregations page. The None option is the default for the Sales Rank measure because the data source is numeric and refers only to existing measures.

Using the RANK function in the measure's expression to order the Sales column in descending order, the analysts can drill-down across the other dimensions to determine the store with the worst sales history over the last year, for a particular product line, or with respect to any other dimensional data stored in the database.

Using time data stored in the fact table to create a Time dimension

A retail business, XYZ Retail, is modeling their sales transaction data in DB2 Cube Views so that they can analyze their data more effectively. However, because of the transactional nature of the data, the only time information that is available is a date that is associated with each transaction.

Time information, that is modeled in a time dimension, is needed to add context to many common calculations, such as analyzing sales trends by quarter, and calculating the average inventory value for each week.

Many DBAs avoid storing time data as a date or timestamp for a transaction because if there are no transactions on one day, then there are holes in the data, which can create problems aggregating and displaying the data accurately. Usually,

modeling time data in a time table is a better choice. However, the DBA for XYZ Retail is confident that there will be at least one transaction every day and decides to keep the current structure for the data.

Details of the scenario

XYZ Retail has a fact table with measurable data about each transaction including Sales, Costs, Quantity Sold, and Date. Additionally, the database contains a Region dimension table and a Product dimension table. The problem is that time data is included in the fact table rather than being stored in a separate dimension table. The DBA must create a dimension object based on the date data in the facts object.

Creating a time dimension based on a single column of date data in the fact table has two unique requirements:

- Because all dimension objects in a valid cube model must be joined to the facts object, and the time dimension object and the facts object are based on the same fact table, the time dimension object must be joined to the facts object by using a self-join to join the fact table to itself.
- The DBA must build calculated attributes that aggregate the date data into meaningful levels such as Week, Month, Quarter, and Year.

A self-join is a type of join that joins a table to itself, in this case the table is the fact table. The self join should join one or more columns that together can uniquely identify any row in the fact table. The primary key is the best choice. However, if a primary key is not defined, a good primary key candidate is the set of columns that are used to join the fact table with the dimension tables. To optimize the cube model, a primary key must be defined. The join cardinality must be 1:1, and the join type must be inner.

Figure 22 on page 77 shows how a facts object, a dimension based on the fact table, and a facts-to-dimension join can map to the same fact table.

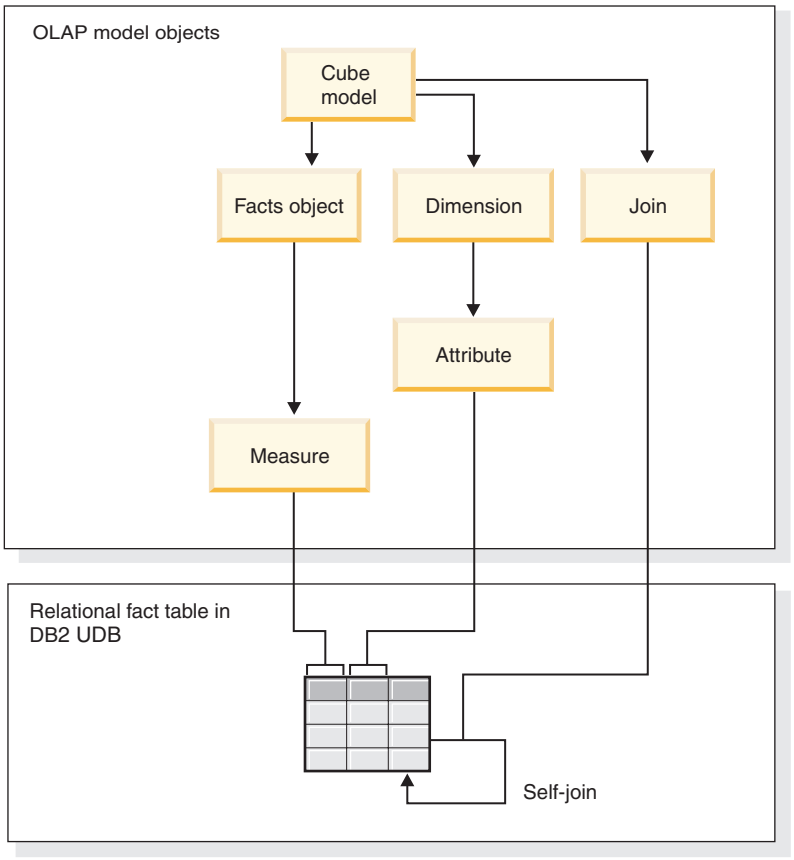

Figure 22. How a self-join joins a table to itself

Steps to create the attributes and dimension

The following steps explain how you could use the OLAP Center Dimension wizard to create the Time dimension and calculated attributes based on the fact table:

1. To open the Dimension wizard, right-click the cube model in the OLAP Center object tree, and click **Create Dimension**. The Dimension wizard opens.
2. On the Name page, type `Time` in the **Name** field. You can optionally change the business name and type a comment. Click **Next**.
3. Select the cube model's fact table. Click **Next**. You do not need to specify dimensional joins because you have only one table in your dimension. On the Dimension Joins page, click **Next**.
4. On the Dimension Attributes page, select the **Timestamp** column.
5. Optional: Create the additional calculated attributes that aggregate the timestamp data into larger chunks such Month, Quarter, and Year. To create calculated attributes, click the **Create Calculated Attribute** push button to open the SQL Expression Builder and define the expression for each attribute that calculates the source timestamp column into months, quarters, and years. After defining each calculated attribute, click the **Validate** push button to ensure that the expression is valid, and then click **OK** to close the SQL Expression Builder and return to the Dimension wizard. Click **Next** after you select and create all of the attributes that you want.
6. On the Dimension Type page, select **Time**. Click **Next**.

7. On the Fact-Dimension Join page, click **Create Join**. In the Join wizard that opens, create the self join. Type a name and click **Next**. Select the column or set of columns that uniquely define any row in the fact table, such as the primary key, for both the left and right attributes. Select one pair at a time and click **Add** to add the attribute pair to the join. Select the inner join type and 1:1 cardinality. After adding the necessary attribute pairs, click **Finish**. The Join wizard closes.
8. On the Fact-Dimension Join page, click **Finish**.

With the Time dimension defined in the cube model, XYZ Retail can add a new level of meaning to its data analysis. They can now perform time-related analyses including inventory.

Chapter 5. DB2 Cube Views cube model optimization

This section describes the following topics:

Summary tables
DB2 Cube Views uses DB2 summary tables to improve the performance of queries issued to cube models. A summary table is a special type of a materialized query table (MQT) that specifically includes summary data.

Summary tables with functional dependencies and constraints
The Optimization Advisor utilizes information about the relationships between data, such as functional dependencies and constraints, to recommend summary tables that contain aggregated measures and the level attributes that are necessary for the DB2 optimizer to efficiently answer queries.

Overview of the optimization process
Optimizing your star schema or snowflake schema with DB2 Cube Views can improve the performance of OLAP-style SQL queries. The optimization process includes creating, implementing, and maintaining the summary tables recommended by the Optimization Advisor.

Metadata design considerations for optimization
The way that you design your metadata objects, including levels and hierarchies, measures, cubes, and optimization slices, affects the summary tables that the Optimization Advisor wizard recommends.

Optimization slices for cubes
An optimization slice is an optional, but powerful aid to guide the Optimization Advisor to provide summary tables that are focused on the most important regions of your cube model.

Analyzing your queries for optimization slices
Optimization slices are a powerful tool for improving query performance, but they are effective only if they accurately reflect your users' queries.

Constraint definitions for optimization
Constraints provide valuable information to the Optimization Advisor and to the DB2 optimizer. You must define informational or enforced constraints for foreign keys and primary keys in your star schema or snowflake schema.

Parameters for the Optimization Advisor
The information that you provide to the Optimization Advisor wizard for each parameter affects the summary tables that the wizard recommends and the performance improvements that you gain. Be sure to supply accurate information and to make careful decisions between cost and performance requirements.

Optimizing a cube model
By optimizing for queries performed on a cube model, you can improve the performance of products that issue OLAP-style SQL queries.

Example of an SQL script to create summary tables
The Optimization Advisor wizard provides an SQL script to create the recommended summary tables. The SQL script contains the necessary SQL commands to build one or more summary tables.

Testing query results
You can use the db2batch Benchmark Tool in DB2 Universal Database to benchmark your query performance results before and after you create the summary tables with the Optimization Advisor.

Troubleshooting summary tables
If the performance of your queries does not improve after you create summary tables, you can use the DB2EXPLAIN facility to troubleshoot the query routing.

Summary table maintenance
When the data in your base tables changes, you need to update your summary tables. You can update your summary tables in two different ways: refresh immediate or refresh deferred.

Dropping a summary table
DB2 Cube Views does not drop the associated summary tables when you drop a cube model. If you do not use the summary tables for any other purpose, you can drop the tables to free disk space.

Summary tables

DB2 Cube Views uses DB2 summary tables to improve the performance of queries issued to cube models and cubes. A summary table is a special type of a materialized query table (MQT) that specifically includes summary data.

Because the Optimization Advisor always recommends MQTs with summarized data, the term summary table is used in the DB2 Cube Views documentation to describe the recommended MQTs.

You can complete expensive calculations and joins for your queries ahead of time and store that data in a summary table. When you run queries that can use the precomputed data, DB2 UDB will reroute the queries to the summary table. A query does not need to match the precomputed calculations exactly. If you use simple analytics like SUM and COUNT, DB2 UDB can dynamically aggregate the results from the precomputed data. Many different queries can be satisfied by one summary table. Using summary tables can dramatically improve query performance for queries that access commonly used data or that involve aggregated data over one or more dimensions or tables.

Figure 23 on page 81 shows a cube model based on a snowflake schema with a Sales facts object and Time, Market, and Product dimensions. The facts object has measures and attributes, and each dimension has a set of attributes and is joined to the facts object by a facts-to-dimension join.

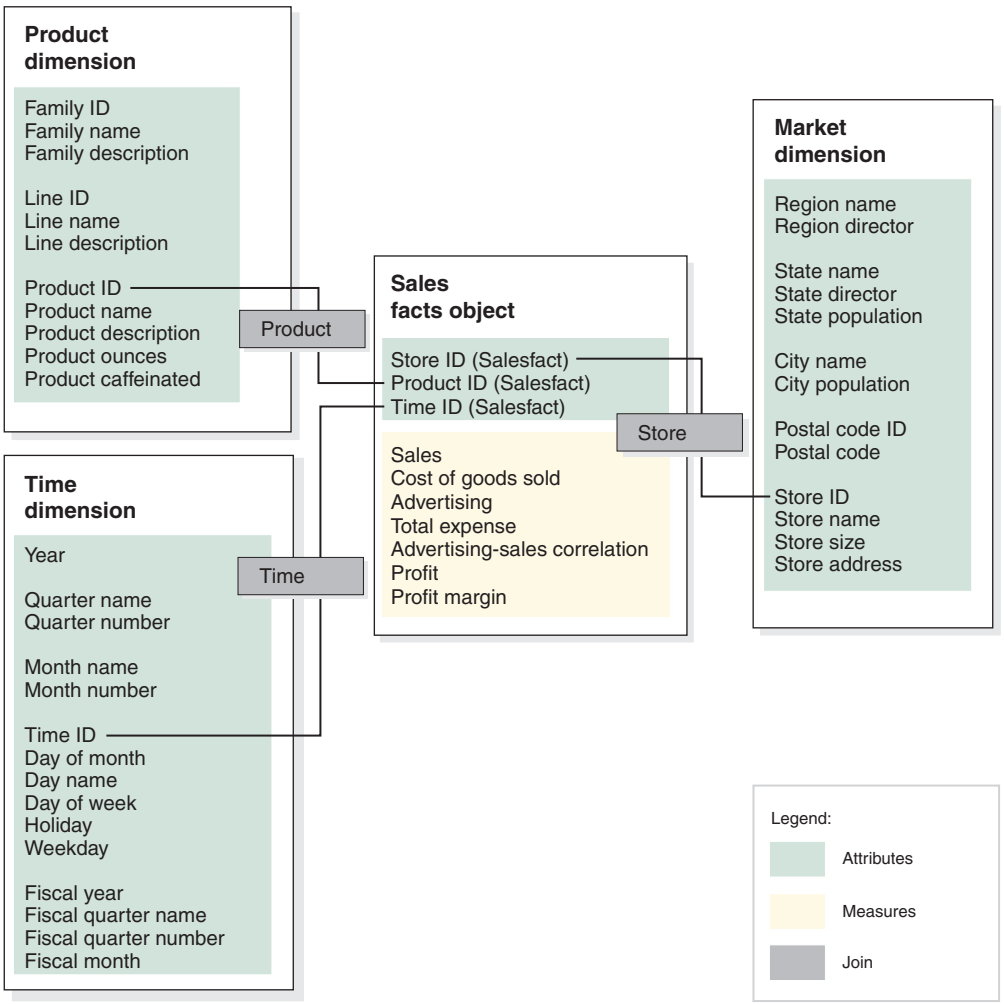

Figure 23. Cube model. Cube model with a Sales facts object and Time, Product, and Market dimensions

The hierarchy for each dimension in the cube model is shown in Figure 24 on page 82. The highlighted boxes connected by the thick dark line across the bottom of the hierarchies represent the data that actually exists in the base tables. Sales data is stored at the Day level, Store level, and Product level. Data above the base level in the hierarchy must be aggregated. If you query a base table for sales data from a particular month, DB2 UDB must dynamically add the daily sales data to return the monthly sales figures. For example, you can use the following query to see the sales data for each product line, in each region, by each month in 2004:

```
SELECT LINE_ID, REGION_NAME, MONTH_NUMBER, SUM(SALES)
FROM TIME, STORE, LOCATION, PRODUCT, LINE, SALESFACT
WHERE SALESFACT.STOREID = STORE.STOREID
  AND STORE.POSTALCODEID = LOCATION.POSTALCODEID
  AND SALESFACT.PRODUCTID = PRODUCT.PRODUCTID
  AND PRODUCT.LINEID = LINE.LINEID
  AND SALESFACT.TIMEID = TIME.TIMEID
  AND YEAR = '2004'
 GROUP BY LINEID, MONTH_NUMBER;
```

The thin line connecting the Line-Region-Month slice in Figure 24 on page 82 represents the slice that the query accesses. Line-Region-Month is a slice of the cube model and includes one level from each hierarchy. You can define summary tables to satisfy queries at or above a particular slice. A summary table can be built

for the Line-Region-Month slice that is accessed by the query. Any other queries that access data at or above that slice including All Time, Year, Quarter, All Markets, All Products, and Family can be satisfied by the summary table with some additional aggregating. However, if you query more detailed data below the slice, such as Day or City, DB2 UDB cannot use the summary table for this more granular query.

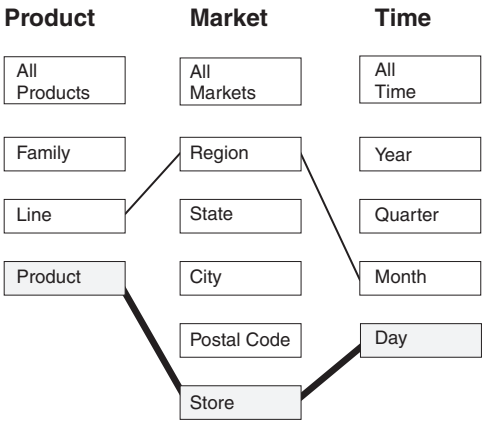

Figure 24. Query slice. Product, Market, and Time hierarchies. Shows the Line-Region-Month slice and that the base data exists in the Product-Store-Day slice.

In Figure 25, the dotted line defines the Line-State-Month slice. A summary table built for the Line-State-Month slice can satisfy any query that accesses data at or above the slice. All of the data that can be satisfied by a summary table built for the Line-State-Month slice is included in the top set of highlighted boxes.

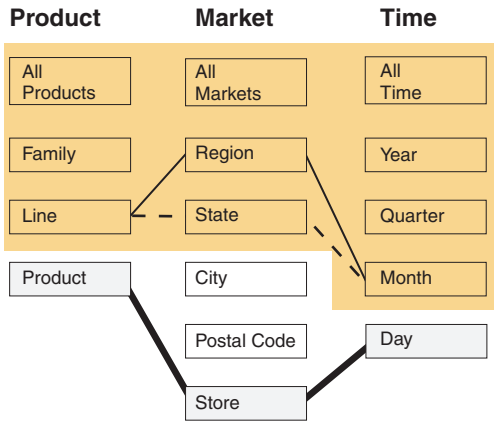

Figure 25. Summary table slice. Product, Market, and Time hierarchies. The highlighted data can be satisfied by a summary table built at the Line-State-Month slice.

The rewriter in the DB2 SQL compiler knows about existing summary tables, and can automatically rewrite queries to read from the summary table instead of the base tables. Rewritten queries are typically much faster because the summary tables are usually much smaller than the base tables and contain preaggregated data. Users continue to write queries against the base tables. DB2 UDB will decide when to use a summary table for a particular query and will rewrite the user's query to access the summary tables instead, as shown in Figure 26 on page 83. The rewritten query accesses a summary table that contains preaggregated data. A summary table is often significantly smaller, and therefore significantly faster, than the base tables and returns the same results as the base tables.

You can use the DB2 EXPLAIN facility to see if the query was rerouted, and if applicable, which table it was rerouted to.

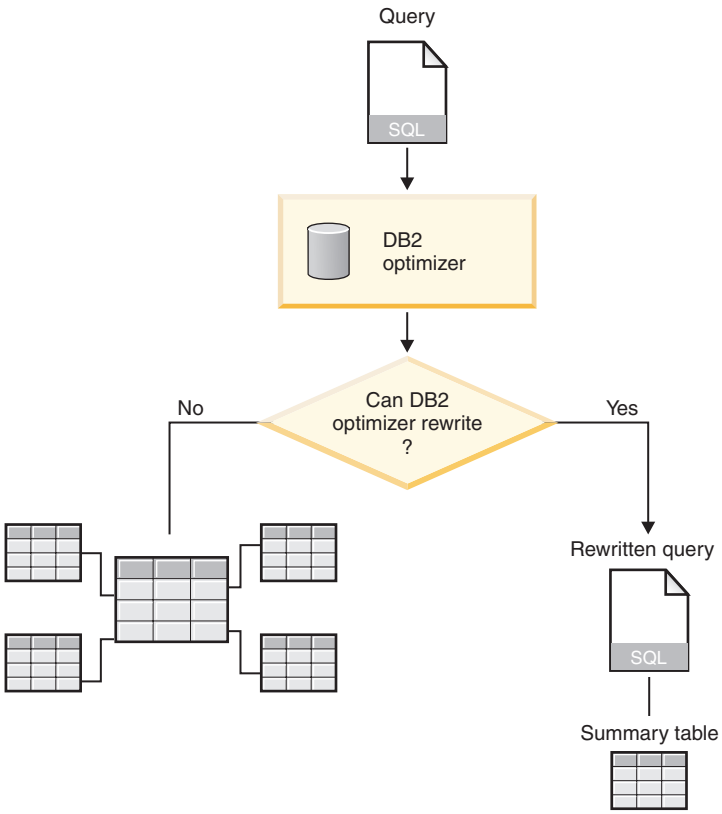

Figure 26. Query rewrite. The DB2 UDB process for rewriting a query

The query to see the sales data for each product line, in each region, by each month in 2004, can be rewritten to use the summary table built for the Line-Region-Month slice. The original query is here:

```
SELECT LINE_ID, REGION_NAME, MONTH_NUMBER, SUM(SALES)
FROM TIME, STORE, LOCATION, PRODUCT, LINE, SALESFACT
WHERE SALESFACT.STOREID = STORE.STOREID
  AND STORE.POSTALCODEID = LOCATION.POSTALCODEID
  AND SALESFACT.PRODUCTID = PRODUCT.PRODUCTID
  AND PRODUCT.LINEID = LINE.LINEID
  AND SALESFACT.TIMEID = TIME.TIMEID
  AND YEAR = '2004'
 GROUP BY LINEID, MONTH_NUMBER;
```

The rewritten query is here:

```
SELECT LINE_ID, REGION_NAME, MONTH_NUMBER, SUM(SALES)
FROM SUMMARYTABLE1
WHERE YEAR = '2004'
GROUP BY LINE_ID, REGION_NAME, MONTH_NUMBER;
```

The rewritten query is much simpler and quicker for DB2 UDB to complete because the data is preaggregated and many of the table joins are precomputed so DB2 UDB accesses one small table instead of six tables, including a large fact table. The savings with summary tables can be tremendous, especially for schemas that have large fact tables. For example, a fact table with 1 billion rows might be preaggregated to a summary table with only 1 million rows, and the calculations

involved in this aggregation occur only once instead of each time a query is issued. A summary table that is 1000 times smaller is much faster than accessing the large base tables.

In this example, Figure 27 shows the summary table for the Line-State-Month slice. DB2 UDB needs to calculate data for Region from the higher level State instead of from the lower level Store, so the summary table has fewer rows than the base tables because there are fewer states than stores. DB2 UDB does not need to perform any additional calculations to return sales data by Month and Line because the data is already aggregated at these levels. This query is satisfied entirely by the data in the summary table that joins the tables used in the query ahead of time and the joins do not need to be performed at the time the query is issued. For more complex queries, the performance gains can be dramatic.

Region name	State name	Line ID	Year	Quarter number	Quarter name	Month number	Sales	Cost of goods	Advertising	Total expense	Profit
West	Idaho	054	2004	1	Qtr 1	2	9700	2500	700	3200	6500
East	Maine	102	2004	2	Qtr 2	5	3000	500	200	700	2300
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Figure 27. Summary table. Example of the summary table that is created for the Line-Region-Month slice

In some cases, a query might access an attribute that is related to an attribute that is included in the summary table. The DB2 optimizer can use functional dependencies and constraints to dynamically join the summary table with the appropriate dimension table.

When the Optimization Advisor recommends a summary table, all of the measures in the cube model are included. In this example, the SalesFacts object has only five measures including Sales, Cost of goods, Advertising, Total expense, and Profit, which are all included in the summary table. If you define fifty measures for your cube model, all fifty measures are included in the summary table. The Optimization Advisor does not need to include all of the related attributes that are defined for a level in the summary table because DB2 Cube Views defines functional dependencies between the attributes in a level.

Summary tables with functional dependencies and constraints

The Optimization Advisor utilizes information about the relationships between data, such as functional dependencies and constraints, to recommend summary tables that contain aggregated measures and the level attributes that are necessary for the DB2 optimizer to efficiently answer queries.

DB2 Cube Views defines functional dependencies, whenever possible, between attributes in a level. When you define a level, you define a relationship between the level key attributes and the other attributes in the level (default attribute and related attributes). The relationship indicates that the level key attributes can be used together to determine the other attributes in the level. DB2 Cube Views documents the relationship between the level's attributes by defining functional

dependencies between the attributes. DB2 UDB and DB2 Cube Views can use the relationships, defined by functional dependencies, to perform intelligent optimization of your data.

You must ensure that the underlying data for the level attributes is functionally dependent in the way described by the functional dependency. DB2 UDB does not verify the validity of the functional dependencies.

If a functional dependency exists between a level key attribute and the level's related attributes, the Optimization Advisor can include the level key attribute without the related attributes in the summary table. Queries that are interested in the level's related attributes can still be routed to the summary table because the DB2 optimizer joins the summary table with the dimension table when the query is issued to create the final result set.

For example, you can use a query that is very similar to the query described in "Summary tables" on page 80, to see the sales data for each product line, in all regions, by each month in 2004. The following query is different in that it groups the results by Line name instead of Line ID and Month name instead of Month number. The result set is more usable, but relies on functional dependencies and constraints to access the data.

```
SELECT LINE_NAME, REGION_NAME, MONTH_NAME, SUM(SALES)
FROM TIME, STORE, LOCATION, PRODUCT, LINE, SALESFACT
WHERE SALESFACT.STOREID = STORE.STOREID
  AND STORE.POSTALCODEID = LOCATION.POSTALCODEID
  AND SALESFACT.PRODUCTID = PRODUCT.PRODUCTID
  AND PRODUCT.LINEID = LINE.LINEID
  AND SALESFACT.TIMEID = TIME.TIMEID
  AND YEAR = '2004'
 GROUP BY LINE_NAME, REGION_NAME, MONTH_NAME;
```

The DB2 optimizer will rewrite the query so that it joins the summary table to the appropriate dimension tables, as shown in Figure 28 on page 86.

The Line level contains the following attributes:

- Line ID as the level key attribute
- Line name as the default attribute
- Line description as the related attribute

Line ID is the primary key of the Line table, so a constraint exists on the Line ID column. Because a constraint exists for the Line ID column, DB2 Cube Views does not create a functional dependency for this level. The DB2 optimizer uses the constraint to join the summary table to the Line table and to access the Line name data for the query result set.

The Month level contains the following attributes:

- Year and Month number as the level key attributes
- Month name as the default attribute

A functional dependency exists for the Month level that specifies that Month name is functionally dependent on the combination of Year and Month number. The Year and Month number columns are not part of the primary key or foreign key for the Time table, so there is no constraint on these columns. The DB2 optimizer uses the functional dependency between Month name and the combination of Year and

Month number, to join the summary table to the Time table and access the Month name data for the query result set.

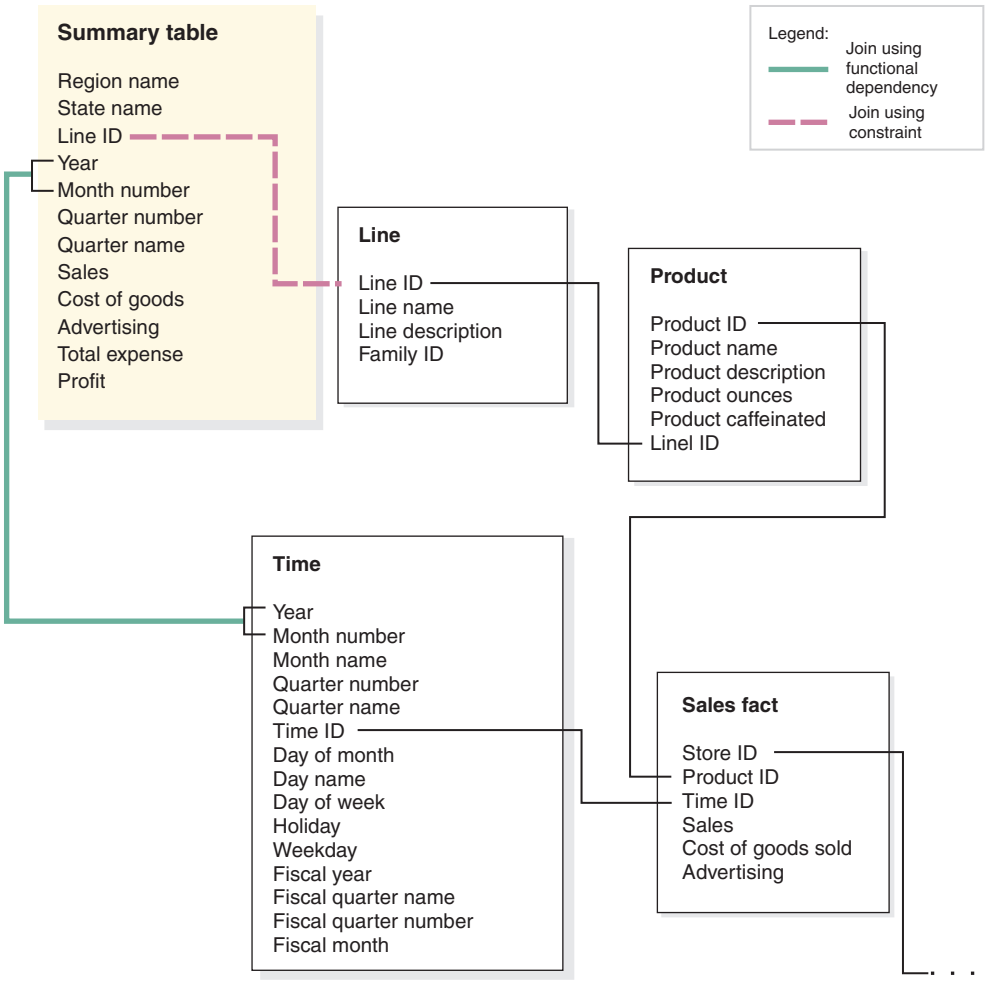

Figure 28. Joins. DB2 optimizer dynamically joins the summary table with the appropriate dimension tables when a query is issued

The DB2 optimizer rewrites the query as:

```
SELECT Q4.LINE_NAME, Q4.REGION_NAME, Q4.MONTH_NAME, SUM(Q4.SALES)
FROM (
    SELECT DISTINCT Q1.LINE_NAME, Q3.REGION_NAME, Q2.MONTH_NAME,
      Q3.SALES, Q2.YEAR, Q2.MONTH_NUMBER
    FROM LINE AS Q1, TIME AS Q2, SUMMARYTABLE1 AS Q3
    WHERE (Q3.YEAR=2004)
      AND Q3.LINEID=Q1.LINEID)
      AND (Q3.MONTH_NUMBER=Q2.MONTH_NUMBER)
      AND (2004=Q2.YEAR)
  ) AS Q4
GROUP BY Q4.LINE_NAME, Q4.REGION_NAME, Q4.MONTH_NAME
```

The resulting query joins only three tables instead of the six tables in the original query, and most importantly, the rewritten query does not have to access the large fact table that is usually quite large and slow. The rewritten query is much faster because the summary table already contains the preaggregated measure data.

Overview of the optimization process

Optimizing your star schema or snowflake schema with DB2 Cube Views can improve the performance of OLAP-style SQL queries. The optimization process includes creating, implementing, and maintaining the summary tables recommended by the Optimization Advisor.

The Optimization Advisor can help you optimize your cube models by recommending summary tables. DB2 UDB summary tables can improve query performance because they contain precomputed results from one or more tables that can be used in a query. Costly table joins and complex calculations can be computed in advance and stored in a summary table so that future queries that use these aggregations can run much faster. For information about summary tables, see "Summary tables" on page 80.

The Optimization Advisor will analyze your metadata and the information that you provide to the wizard and recommend the appropriate summary tables. After running the Optimization Advisor, you will have an SQL file that can build the set of recommended summary tables. You have the option of modifying the SQL before you run it to create the summary tables.

Running the Optimization Advisor is just one step in the optimization process. Before you begin optimizing, you need to think about several issues including, but not limited to:

- How to effectively use DB2 constraints on the base tables
- How to define your cube model so that it follows the optimization validation rules
- What types of queries you want to optimize for
- How much space you want to provide
- How you will maintain your summary tables so that the data that they contain is current

Before you can optimize, you must define constraints on your base tables. For information about the types of constraints that are required, see "Constraint definitions for optimization" on page 101.

Many parts of the optimization process are iterative and might need to be repeated to fine tune and maintain your performance gains. Figure 29 on page 88 shows an overview of the main steps in the optimization process.

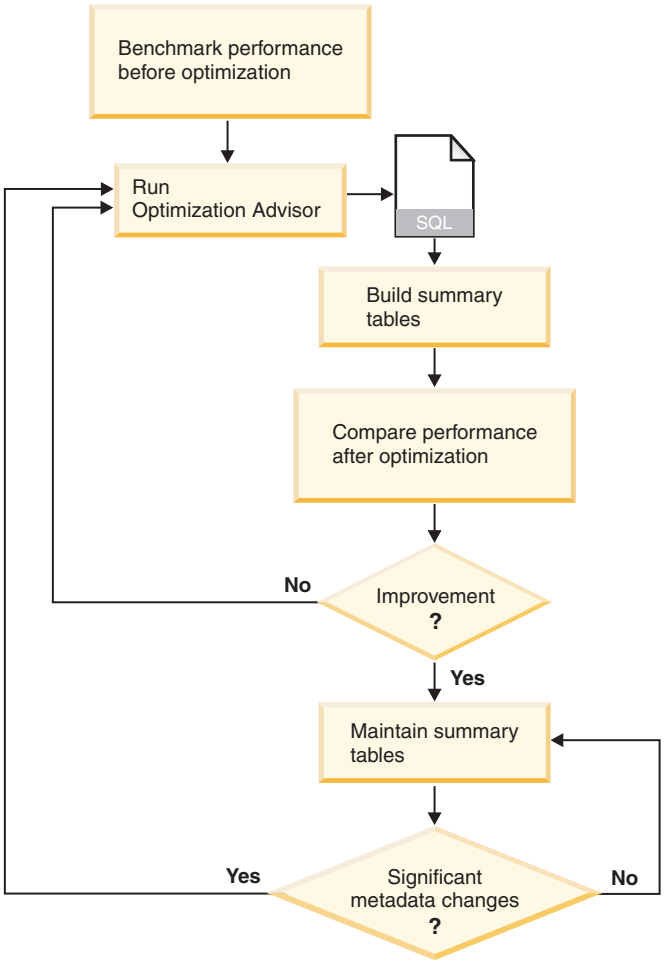

Figure 29. Optimization process. Overview of the main steps in the optimization process

The optimization process includes these general tasks:

- **Measure performance**

  Before you run the DB2 Cube Views Optimization Advisor, you should measure the current performance for a specific set of typical queries. Performance measurement is an optional step that provides a benchmark so that you can later analyze the success of the optimization. You can use the db2batch Benchmark Tool provided with DB2 UDB to create a benchmark. For information about using db2batch, see "Testing query results" on page 112. You run sample queries to complete the performance benchmark, but the Optimization Advisor does not require sample queries because it is metadata based and makes recommendations without knowing the specific queries that will be issued.

- **Run the Optimization Advisor wizard**

  You provide several important parameters to the wizard including: types of queries that you want to optimize for, disk space and time limitations, update method, and table space locations. For information about these parameter choices, see "Parameters for the Optimization Advisor" on page 106. The Optimization Advisor creates its recommendations based on the information that you provide, metadata, DB2 statistics, and any data sampling that you allow. The Optimization Advisor considers the parameters that you specify and generates two SQL files. One SQL file contains the SQL commands to build a set of recommended summary tables. The other SQL file contains the SQL commands to update the recommended summary tables.

- **Create the summary tables**

  You can create the summary tables immediately after completing the wizard, or you can add the operation to your normal database maintenance schedule. Creating the summary tables can require significant time and processing resources. After the summary tables are built, verify that the performance of queries against the optimized cube model is improved. Run the same set of sample queries that you ran before optimizing, and compare the performance results. If you do not see significant performance gains, you might need to run the Optimization Advisor again and allocate more disk space, or time, or both, or you might need to change other settings. For information about how to verify and analyze your performance results, see "Testing query results" on page 112.

- **Maintain the summary tables**

  After your summary tables are created, you need to regularly maintain the tables to ensure that they stay appropriately synchronized with your data. When you run the Optimization Advisor, you choose either a refresh-immediate or refresh-deferred update option.

  Refresh-immediate option

  If you choose the immediate update option, DB2 UDB keeps your base tables and summary tables synchronized and incrementally updates the summary tables when the underlying tables are changed. DB2 UDB supports incremental maintenance for simple aggregations such as SUM and COUNT. For other aggregations, the Optimization Advisor recommends summary tables that use the refresh-deferred option regardless of which refresh option you select.

  Refresh-deferred option

  If you choose the deferred option, you will rebuild your summary tables to update them. You can decide when to perform the summary table update. If you make significant changes throughout your base tables, deferring the update can be more efficient than incremental updates.

  When choosing between these options, you need to make trade-offs between the resources that you can allocate to the maintenance and how precisely the data must be synchronized. For more information about the immediate and deferred update options, see "Summary table maintenance" on page 114.

- **Periodic reevaluation**

  You need to periodically reevaluate the summary tables to ensure that they continue to meet your needs. If you significantly change the metadata by adding or updating a cube model, you might need to run the Optimization Advisor again and build a new set of summary tables:

  – If you add a metadata object such as a new dimension or measure, queries that access data from the new object will not be able to use the existing summary tables. However, queries that do not use the new object will continue to use the summary tables.

  – If you update a metadata object to include data that was not previously optimized for, queries that access the updated object will not be able to use the summary tables.

  – If you delete one or more objects, the effectiveness of the summary tables is not changed, but you are wasting disk space on aggregations that are no longer used.

  In addition to significant metadata changes, you might also need to run the wizard again if the regularly performed query types change and are not the type that you optimized for.

Each time you run the wizard and build new tables, you should complete the whole optimization process again including creating a benchmark and analyzing the performance of the summary tables.

If you drop a cube model you can also drop the associated summary tables if they are not used for any other purpose. DB2 Cube Views does not drop summary tables when the associated cube model is dropped. For information about how to drop a summary table, see "Dropping a summary table" on page 115.

Metadata design considerations for optimization

The way that you design your metadata objects, including levels and hierarchies, measures, cubes, and optimization slices, affects the summary tables that the Optimization Advisor wizard recommends.

Generally, you want to define the basic structure of your facts object, dimensions, and joins according to the structure of your data. You have few choices in the basic structure of these high-level objects within the cube model, so you can rarely improve the recommended summary tables by altering these objects. However, you do have more flexibility in selecting how you define your levels and hierarchies, measures, cubes, and optimization slices.

Levels and hierarchies

If possible, define your levels using the ideal modeling method described in "Levels" on page 26. By using the ideal modeling method, you might reduce the amount of disk space that the recommended summary tables use and the amount of temporary space used to refresh the recommended summary tables.

Measures

If limited disk space is a concern, you might choose to include only critical measures in your cubes and drop any measures that you do not expect to use regularly or that do not have business value, because the more measures that you define in your cubes, the larger your summary tables are.

The types of measures, either distributive or nondistributive, that you include in your cubes can also affect optimization:

- Distributive measures can always be aggregated from one level to the next. For example, SUM(Sales) for Quarter, can be calculated by summing the monthly sales data.
- Nondistributive measures, such as standard deviation, must always be calculated directly from the base data and cannot be aggregated from one level to the next.

In general, the Optimization Advisor and the DB2 optimizer have more flexibility and options when the cube model contains only distributive measures. You can optimize a cube model that contains nondistributive measures, but you might get better optimization results if you omit the nondistributive measures that are not needed from your cubes.

Cubes

Design cubes that match the needs of your business. Cubes are most effectively used in optimization when you build a cube that focuses on an important region of the cube model's data.

If you know of one or more characteristics of the queries that your users frequently issue, you can specify that information in optimization slices for the Optimization Advisor. The Optimization Advisor recommends summary tables that improve queries issued to the specified regions of the cubes. An optimization slice is defined by a set of specific levels or **Any** level designations for each cube dimension, and the type of query expected at the slice such as drill-down, report, MOLAP extract, hybrid extract, or drill through.

In the OLAP Center, you can specify that a cube is generally used for one type of query, such as drill-down. When you specify one query type for a cube, the OLAP Center defines an optimization slice with the type that you specified and the **Any** option for every cube dimension. For more information about optimization slices, see "Optimization slices for cubes" on page 92.

The following list describes the types of queries that you can specify for a cube in the OLAP Center:

Drill-down queries

Drill-down queries usually access a subset of data that is focused at the top of a cube model. Queries can go to any level in the cube model. When users drill deep into one dimension, they typically stay much higher in the other dimensions. Optimizing for drill-down queries will mostly benefit queries that stay in the upper levels of the cube model. Relational OLAP (ROLAP) spreadsheet applications are usually used to perform drill-down queries. For example, a spreadsheet application user might start by accessing the revenue for all regions and all products for the year 2004. Then the user can move deeper into the data by querying for revenue by quarter in all regions and for each country.

Performance is usually very important for these types of queries because they are issued real-time by a user who has to wait for the results to be processed.

Report queries

Report queries are equally likely to access any part of the cube model. Report queries are often issued in batches. Query performance is usually not as critical for report queries as for drill-down queries because a user is less likely to be waiting for an immediate response to each individual query.

MOLAP extract queries

MOLAP extract queries access either the base level of a cube or the optimization slice defined for the cube. The cube is used to load data into a MOLAP data store. The cube optimization slice logically maps to the extract slice that you load the data into the MOLAP application for further processing.

If you specify the MOLAP extract type for a cube in the OLAP Center or if you specify an optimization slice with the **Any** option specified across all of the cube dimensions, the Optimization Advisor creates summary tables that optimize for the data to be extracted at the base level of the cube. Make sure that the base levels of the cube map to the slice that you extract at.

Advanced settings

If you specify Advanced settings for a cube in the OLAP Center, you can specify optimization slices for specific, frequently queried regions of the cube

If you know of one or more characteristics of the queries that are frequently issued, you can specify that information in optimization slices for the Optimization Advisor. The Optimization Advisor considers those slices when recommending summary tables. An optimization slice is defined by a set of specific levels or any level designations for each cube dimension and the type of query expected at the slice such as drill-down, report, MOLAP extract, hybrid extract, or drill through. For information on optimization slices, see "Optimization slices for cubes"

Optimization slices for cubes

An optimization slice is an optional, but powerful aid to guide the Optimization Advisor to provide summary tables that are focused on the most important regions of your cube model.

Cubes can often have many cube dimensions. By specifying one or more optimization slices, you can specify which region of the cube has the most query activity. An optimization slice is defined by a set of one or more levels, and the type of query expected at the slice such as drill-down, report, MOLAP extract, hybrid extract, or drill through. Any type of query can benefit from defining an optimization slice, but report queries are likely to have the most significant benefit.

You must specify one option per cube dimension when you create the optimization slice. Use the following guidelines when specifying an option for each cube dimension:

- Specify a specific level in a cube dimension, such as Month in the Time cube dimension, if you know that the specified level is important or frequently queried.
- Specify **All** in a cube dimension if the highest aggregation of the cube dimension is important or frequently queried.
- Specify **Any** in a cube dimension if no level is significantly more important than any other level in that cube dimension, many levels in that cube dimension are queried, or you do not know how often each level in that cube dimension is queried.

For example, in a cube with 10 cube dimensions, you might have important levels in only two cube dimensions, so you might specify specific levels in those two cube dimensions and **Any** in every other cube dimension.

The following sections describe example optimization slices for each query type, and the summary tables that the Optimization Advisor might recommend:

Drill-down optimization slices

A drill-down optimization slice signifies that users commonly drill down to the defined levels in the cube dimensions. Therefore the Optimization Advisor should include these levels in one or more of the recommended summary tables. Because the type is drill-down, the Optimization Advisor might optimize for both shallow queries and deep queries in some of the cube dimensions.

Recommended: Specify a specific level for a cube dimension only when you know that level is particularly important. In general, specify the **Any** option for most cube dimensions and select a specific level only when that level is involved in intense query activity.

Figure 30 shows an example of a drill-down optimization slice and the slices that the Optimization Advisor might recommend summary tables at. The optimization slice is defined at the Any level in the Product cube dimension and the Market cube dimension, and Month level in the Time cube dimension. This optimization slice signifies that users common drill down to the Month level, but do not have a particular drill-down pattern in the Product or Market cube dimensions.

The possible summary table recommendation includes two aggregation levels. One aggregation level is defined at the Line-State-Month slice and the other aggregation level is defined at the Family-Region-Month slice. Both aggregation levels include aggregations at the Month level of the Time dimension, but provide different levels of aggregation in the other two dimensions. The higher slice provides greater performance improvement for shallow drill-down queries because the queries can be satisfied immediately and DB2 UDB will not need to aggregate up. The lower aggregation level provides performance improvement for deeper drill-down queries.

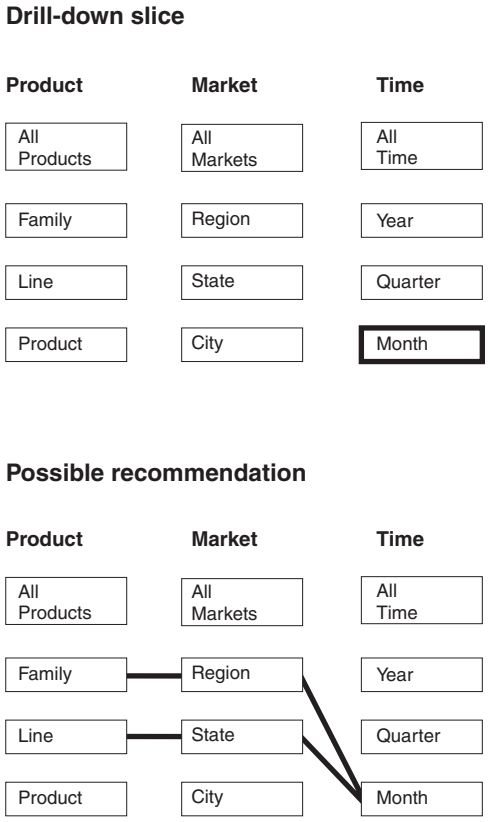

Figure 30. Drill-down. Drill-down optimization slice and possible summary table recommendations

Report optimization slices

A report optimization slice signifies that users commonly create reports at the levels defined in the cube dimensions. Therefore the Optimization Advisor should include these levels in one or more of the recommended summary tables.

Recommended: Specify a specific level for a cube dimension only when you know that level is particularly important. In general, specify the **Any** option for most cube dimensions and select a specific level only when that level is involved in intense query activity.

Figure 31 on page 95 shows an example of a report optimization slice and the slices that the Optimization Advisor might recommended summary tables at.

The optimization slice is defined at the Product level in the Product cube dimension, the Any level in the Market cube dimension, and the Quarter level in the Time cube dimension. This optimization slice signifies that users create reports that include the Product and Quarter levels, and either do not include the Market cube dimension, might include several levels in the Market cube dimension, or you do not know which level in the Market region the users include. For example, users might often create reports that show Sales data for each product family for the last four quarters, but the reports vary in showing the Sales data by Region, State, or City.

The possible summary table recommendation includes two aggregation levels. One aggregation level is at the Product-City-Quarter slice and the other aggregation level is at the Product-Region-Quarter slice. Both aggregation levels include the Product and Quarter levels that are specified in the slice. Based on data sampling and other metadata, the Optimization Advisor decided to create summary tables that cover two aggregation levels, one that includes the City level of the Market cube dimension and another that includes the Region level of the Market cube dimension.

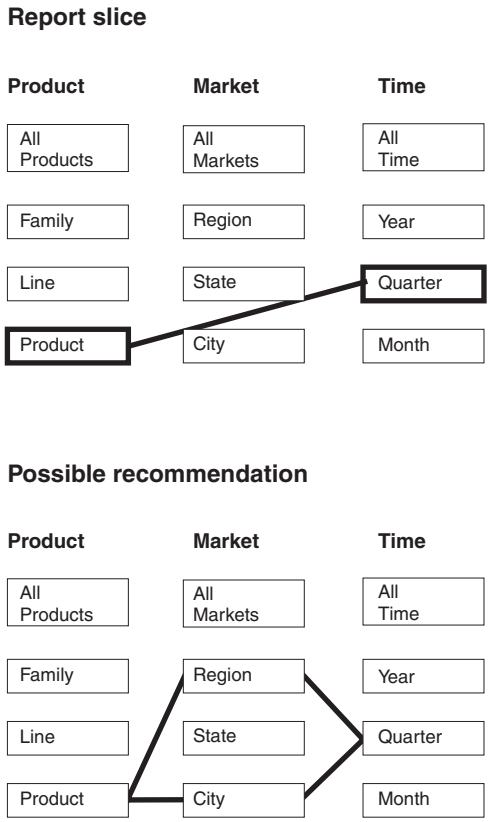

Figure 31. Report. Report optimization slice and possible summary table recommendations

MOLAP extract optimization slices

A MOLAP extract optimization slice signifies that you commonly extract data at the specified slice into a MOLAP cube in a vendor product. Therefore the Optimization Advisor should recommend summary tables that ensure queries issued to the specified slice are fast.

Recommended: Specify a specific level for each cube dimension so that the optimization slice matches the level of data that you extract into your MOLAP cube.

You can define only one MOLAP extract optimization slice per cube. You cannot define a hybrid extract optimization slice in a cube that contains a MOLAP extract optimization slice.

Figure 32 on page 96 shows an example of a MOLAP extract optimization slice and the slice that the Optimization Advisor might recommend a summary table at. The optimization slice is defined at the Product level in the Product cube dimension, the City level in the Market cube dimension, and the Month level in the Time cube dimension. This optimization slice signifies that you will extract the data at the Product-City-Month levels into a MOLAP cube.

The possible recommendation includes one summary table that directly satisfies the MOLAP extract query specified by the Product-City-Month optimization slice.

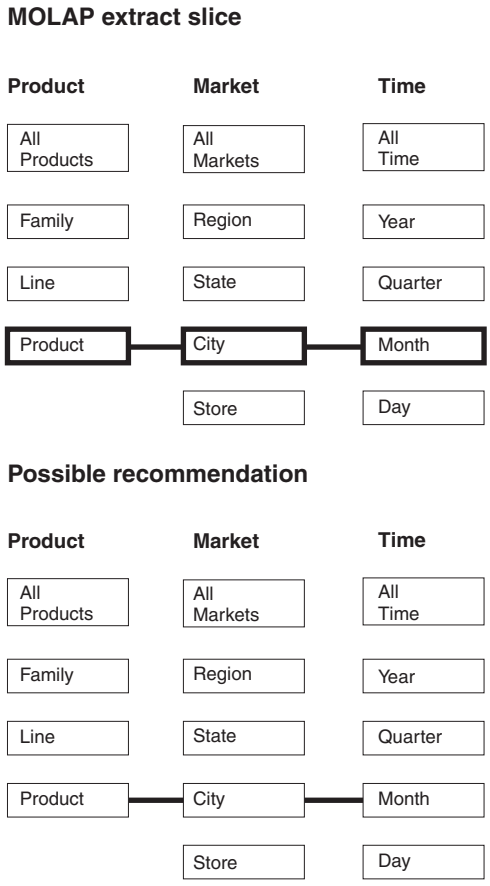

Figure 32. MOLAP extract. MOLAP extract optimization slice and possible summary table recommendations

Hybrid extract optimization slices

A hybrid extract optimization slice signifies that you commonly extract data at the specified slice into a hybrid OLAP (HOLAP) cube in a vendor product. Therefore the Optimization Advisor should include the specified slice in a recommended summary table.

Recommended: Specify a specific level for each cube dimension so that the optimization slice matches the level of data that you extract into your HOLAP cube.

You can define only one hybrid extract optimization slice per cube. You cannot define a MOLAP extract optimization slice in a cube that contains a hybrid extract optimization slice. You can define zero or more drill through optimization slices in the same cube that contains a hybrid extract optimization slice. The Optimization Advisor expects that you might have drill through queries below the specified hybrid extract optimization slice and tries to optimize for drill through queries below the specified slice as well as for hybrid extract queries at the specified slice.

Figure 33 on page 97 shows an example of a hybrid extract optimization slice and the slice that the Optimization Advisor might recommend a summary table at. The optimization slice is defined at the Product level in the Product cube dimension, the City level in the Market cube dimension,

and the Month level in the Time cube dimension. This optimization slice signifies that you will extract the data at the Product-City-Month levels into a HOLAP cube.

The possible summary table recommendation includes two aggregation levels. The aggregation level at the Product-City-Month slice directly satisfies the HOLAP extract query specified by the optimization slice. The aggregation level at the Line-State-Day slice includes the Day level, that is below the hybrid extract slice, to satisfy possible drill through queries in the Time cube dimension. The Optimization Advisor analyzed the other metadata and performed data sampling to develop this recommended summary table.

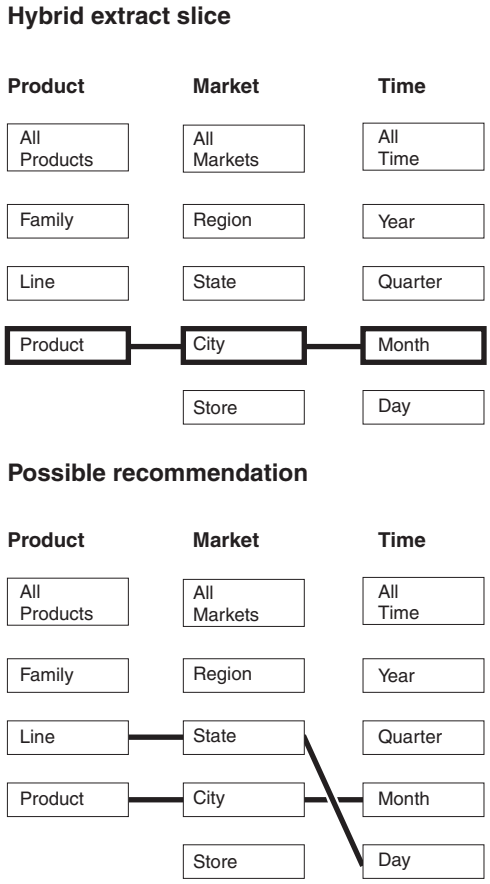

Figure 33. Hybrid extract. Hybrid extract optimization slice and possible summary table recommendations

Drill through optimization slices

A drill through optimization slice must have a corresponding hybrid extract optimization slice defined in the cube. A drill through optimization slice signifies that you commonly drill through to the specified slice from a hybrid OLAP (HOLAP) cube in a vendor product. Therefore the Optimization Advisor should include a slice at or below the specified levels in a recommended summary table.

Recommended: Specify a specific level in a cube dimension for a drill through type of optimization slice, only when you know that level is particularly important. In general for drill through optimization slices, specify the **Any** option for

most cube dimensions and select a specific level only when that level is involved in intense query activity.

Figure 34 on page 99 shows an example of a drill through optimization slice and the corresponding hybrid extract optimization slice, and the slices that the Optimization Advisor might recommend summary tables at. The hybrid extract optimization slice is defined at the Product level in the Product cube dimension, the City level in the Market cube dimension, and the Month level in the Time cube dimension. This optimization slice signifies that you extract the data at the Product-City-Month levels into a HOLAP cube. The drill through optimization slice is defined at the Any level in the Product cube dimension, the Store level in the Market cube dimension, and the Quarter level in the Time cube dimension. This optimization slice signifies that drill through queries from the HOLAP cube usually include the Store and Quarter levels and that drill through queries might or might not reference specific levels from the Product cube dimension.

The possible summary table recommendation includes two aggregation levels. The summary table at the Product-City-Month slice directly satisfies the HOLAP extract query specified by the optimization slice. The aggregation level at the Family-Store-Quarter slice includes the Store level and the Quarter level which are specified by the drill through optimization slice. The Optimization Advisor analyzed the other metadata and performed data sampling to recommend that this summary table also include the Family level in the Product cube dimension.

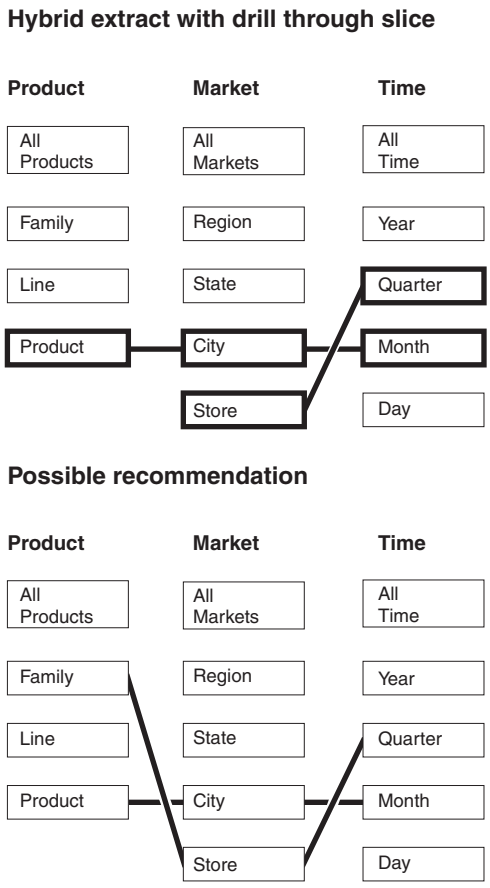

Figure 34. Drill through. Hybrid extract optimization slice and drill through optimization slice and possible summary table recommendations

Analyzing queries for candidate optimization slices

Optimization slices are a powerful tool for improving query performance, but they are effective only if they accurately reflect the SQL queries that are issued.

When analyzing the query patterns of your users, consider the following information:

- The type of queries that your users are likely to issue
- The levels of the hierarchies that are frequently accessed by queries
- The product that your users issue queries from

To determine which regions of the cubes you should create optimization slices for, you can review query history of your users. You might want to also survey your users to determine if their query needs are expected to shift in the future. You are looking for the areas that are the highest priority to optimize for.

Recommendation: Do not specify more than three optimization slices for one cube.

Examples of optimization slices to define for particular situations

Each of the scenarios is based on the Daily sales cube in the CVSAMPLE database that has three cube dimensions including Product, Market, and Time, each with the following cube hierarchies:

Table 35. CVSAMPLE. The cube dimensions and corresponding cube hierarchies for the Daily sales cube in the CVSAMPLE database.

Product cube dimension	Market cube dimension	Time cube dimension
All products	All markets	All time
Family	Region	Year
Line	State	Quarter
Product	City	Month
	Postal code Store	Day

The situations described in the following table are examples of what optimization slices you might define for specific query loads based on the CVSAMPLE database.

Table 36. Example optimization slices

Situation	Scenario	Optimization slices to define
Users use a query product that issues a particular type of query	You know that your users primarily use a product that issues drill-down queries. You do not have any information about what regions of the cubes are queried more frequently.	Because you do not know any specific information about what regions of the cubes are more important to your users, a specific optimization slice will not provide much benefit for you. In this general situation, you can designate the entire cube for drill-down queries when you create the cube in the Cube wizard, or by modifying the cube properties after you create the cube.
Most queries are of one type and focused on one grouping of levels	You know that your users primarily issue report queries and that nearly all of the queries reference the State level of the Market dimension.	You know specific information about a particularly significant region of the cube, so an optimization slice is very beneficial. You can define an Any-State-Any optimization slice that is a report type. This slice references Any level in the Product cube dimension, the State level in the Market cube dimension, and Any level in the Time cube dimension.

Table 36. Example optimization slices (continued)

Situation	Scenario	Optimization slices to define
Most queries are of one type and focused on a few groupings of levels	You know that your users primarily issue report queries. About half of your queries reference the State level of the Market cube dimension and the other half of your queries are randomly distributed across the other levels of the cube dimension.	You can define the following two optimization slices to accurately represent this query distribution: • Define a report optimization slice with the Any-State-Any levels to represent the queries that reference the State level. This slice references Any level in the Product cube dimension, the State level in the Market cube dimension, and Any level in the Time cube dimension. • Define a report optimization slice with the Any-Any-Any levels to represent the other queries that can reference any levels in any of the cube dimensions. By specifying this second slice, you clearly indicate that there are a significant number of queries that access other levels of the Market dimension.
Most queries are of one type and focused on many groupings of levels	You know that your users primarily issue report queries. The query activity is not random, but is distributed across about 15 regions of the cube.	An optimization slice is meant to specify areas of high query activity, and in this situation the queries are too distributed to specify a particular set of optimization slices. Instead of specifying optimization slices, designate the entire cube for report queries when you create the cube in the Cube wizard, or by modifying the cube properties after you create the cube.

Constraint definitions for optimization

Constraints provide valuable information to the Optimization Advisor and to the DB2 optimizer. You must define informational or enforced constraints for foreign keys and primary keys in your star schema or snowflake schema.

You must define constraints on your base tables before you can use the Optimization Advisor. The constraints need to support the base rules, cube model completeness rules, and optimization rules that are described in "Metadata object rules" on page 37, so that the cube model is valid for optimization. The rules primarily define how to join together the metadata objects of your cube model.

You can use informational constraints for the foreign key constraints that you need to define. Informational constraints are a new type of constraint offered in DB2 Universal Database, Version 8. Informational constraints provide a way to improve query performance without increasing maintenance costs. These constraints can be used by the DB2 SQL compiler but are not enforced by the database manager. This type of constraint allows DB2 UDB to know about the relationships in the data without requiring the relationship to be enforced. For primary key constraints, you must use the database-enforced constraints provided with DB2 UDB.

Each join needs a corresponding constraint defined. For example, columns that are involved in facts-to-dimension joins and dimension-to-dimension joins that are used in a snowflake schema, need constraints.

To optimize a cube model based on the snowflake schema shown in Figure 35 on page 103, you need to define constraints on each of the facts-to-dimension joins. The three facts-to-dimension joins are:

- Between Store.StoreID and Sales.StoreID
- Between Time.TimeID and Sales.TimeID
- Between Product.ProductID and Sales.ProductID

Several rules apply to each of these joins. You can use informational constraints only for foreign key constraints.

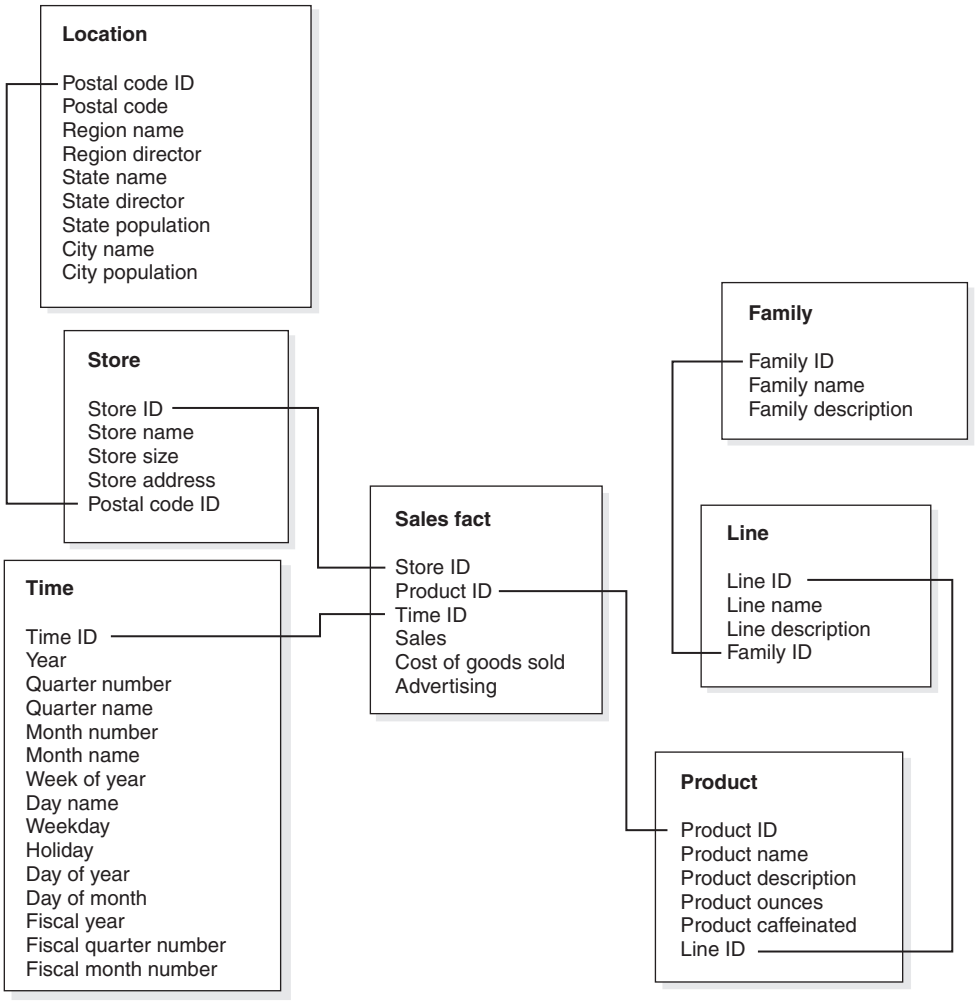

Figure 35. Snowflake schema. Relational tables used the snowflake schema from the CVSAMPLE database.

For the join between the Store and Sales tables, you must define the following constraints:

- StoreID is the primary key in the Store table.
- Store.StoreID and Sales.StoreID are both nonnullable columns.
- Sales.StoreID is a foreign key that reference Store.StoreID. Foreign key constraints can be defined as informational constraints.
- If Sales.StoreID is not the primary key for the Sales table, the join cardinality is 1:Many (Store.StoreID : Sales.StoreID). If Sales.StoreID is the primary key for the Sales table, the join cardinality is 1:1.
- The join type is INNER JOIN.

For the join between the Time and Sales tables, you must define the following constraints:

- TimeID is the primary key in the Time table.
- Time.TimeID and Sales.TimeID are both nonnullable columns.
- Sales.TimeID is a foreign key that reference Time.TimeID. Foreign key constraints can be defined as informational constraints.

- If Sales.TimeID is not the primary key for the Sales table, the join cardinality is 1:Many (Time.TimeID : Sales.TimeID). If the Sales.TimeID is the primary key for the Sales table, then the join cardinality is 1:1.
- The join type is INNER JOIN.

For the join between the Product and Sales tables, you must define the following constraints:

- ProductID is the primary key in the Product table.
- Product.ProductID and Sales.ProductID are both nonnullable columns.
- Sales.ProductID is a foreign key that reference Product.ProductID. Foreign key constraints can be defined as informational constraints.
- If Sales.ProductID is not the primary key for the Sales table, the join cardinality is 1:Many (Product.ProductID : Sales.ProductID). If Sales.ProductID is the primary key for the Sales table, the join cardinality is 1:1.
- The join type is INNER JOIN.

In a snowflake schema, each dimension has a primary dimension table, to which one or more additional dimensions can join. The primary dimension table is the only table that can join to the fact table. Each of the outrigger tables that join directly to the primary table must have a join cardinality of Many:1 (where Many is on the side of the primary table) or 1:1. The primary dimension table usually has the most detailed level of information of all of the dimension tables because of these join cardinality rules. If a set of dimension tables only uses 1:1 join cardinalities, then all of the tables have the same level of detail.

This cube model is based on a snowflake schema, so you must define additional constraints on the joins between the dimension tables. The three dimension-to-dimension joins are:

- Between the Store table and the Location table
- Between the Product table and the Line table
- Between the Line table and the Family table

Several rules apply to each of these joins. You can use informational constraints only for foreign key constraints.

For the join between the Store and Location tables, you must define the following constraints:

- PostalcodeID is the primary key in the Location table.
- Location.PostalcodeID and Store.PostalCodeID are both nonnullable columns.
- Store.PostalCodeID is a foreign key referencing Location.PostalCodeID. Foreign key constraints can be defined as informational constraints.
- The join cardinality is 1:Many (Location.PostalCodeID : Store.PostalCodeID) because Store.PostalCodeID is not the primary key or unique key for the Store table.
- The join type is INNER JOIN.

For the join between the Product and Line tables, you must define the following constraints:

- LineID is the primary key of the Line table.
- Line.LineID and Product.LineID are both nonnullable columns.
- Product.LineID is a foreign key referencing Line.LineID. Foreign key constraints can be defined as informational constraints.

- The join cardinality is 1:Many (Line.LineID : Product.LineID), because Product.LineID is not the primary key or unique key for the Product table.
- The join type is INNER JOIN.

For the join between the Line and Family tables, you must define the following constraints:

- FamilyID is the primary key of the Family table.
- Family.FamilyID and Line.FamilyID are both nonnullable columns.
- Line.FamilyID is a foreign key referencing Family.FamilyID . Foreign key constraints can be defined as informational constraints.
- The join cardinality is 1:Many (Family.FamilyID : Line.FamilyID), because Line.FamilyID is not the primary key or unique key for the Line table.
- The join type is INNER JOIN.

Figure 36 shows a valid set of dimension tables in a snowflake schema dimension. The primary dimension table is the Customer table, with three additional outrigger tables including City and CustomerGroup joined directly to Customer, and CityInfo joined to City. The join cardinalities are semantically valid because there can be many customers in a city or a customer group, and one set of city information exists per city. This is a valid dimension for optimization because it conforms to the optimization validation rules. The dimension has only one primary table, and the City and CustomerGroup tables joined directly to the primary table are joined with a Many:1 cardinality. The CityInfo table is joined with a 1:1 cardinality which is also valid. The Customer table has the most detailed level of information out of the four dimension tables.

Valid dimension for optimization

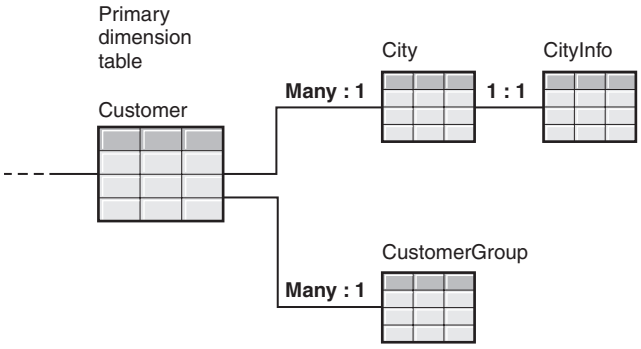

Figure 36. Valid dimension. A set of dimension tables used in one dimension that can be optimized

Figure 37 on page 106 shows an invalid set of dimension tables in a snowflake schema dimension. Because of the cardinality relationships that are defined, it is not possible for any of these tables to be the primary dimension table in a cube model that will be optimized. Although the cardinalities are semantically valid, if any of these tables joined with the fact table as the primary dimension table, the data in the fact table would be multiplied, which causes what is known as a fan trap.

For example, if Customer is the primary dimension table, the 1:Many join cardinality between Region and SalesRep makes the dimension invalid for optimization. If each region has five sales representatives, then when the SalesRep and Region tables are joined, there are five entries for each region. When these

tables are joined with the City and Customer tables, and ultimately with the fact table, an additional five rows are added for each existing row in the City, Customer, and fact table. Repeating the same fact row five times causes the measures to be miscalculated. Each of the other tables in the dimension has similar problems. The City table cannot be the primary dimension table because of the 1:Many joins between City and Customer and between Region and SalesRep. The Region table cannot join with the fact table because every join in the dimension is a 1:Many join to the Region table. The SalesRep table cannot be the primary dimension table either because of the 1:Many joins between the Region and City tables and the City and Customer tables.

Invalid dimension for optimization

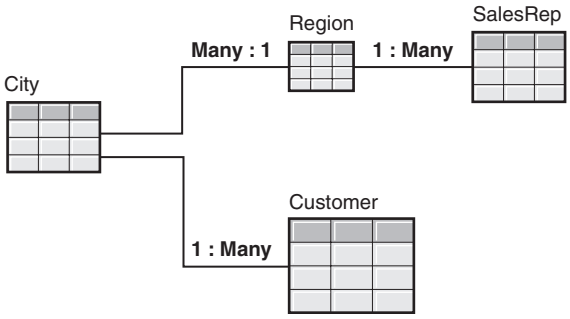

Figure 37. Invalid dimension. A set of dimension tables used in one dimension that cannot be optimized

Parameters for the Optimization Advisor

The information that you provide to the Optimization Advisor wizard for each parameter affects the summary tables that the wizard recommends and the performance improvements that you gain. Be sure to supply accurate information and to make careful decisions between cost and performance requirements.

Expected query activity

You specify the type of queries that you want to optimize each cube in the cube model for. The query types describe when and how DB2 relational data is typically accessed. This information helps the Optimization Advisor understand which portions of the cube model are queried most frequently. You can also specify optimization slices for a cube if you know that a few particular slices are queried most often.

You define the query types for each cube when you create the cube. You can use the Optimization Advisor to review what you specified for each cube, and make any changes as needed.

Disk space limitations

You specify an approximate amount of disk space that can be used for summary tables. The Optimization Advisor cannot know the exact size of the summary tables until they are built, so it recommends summary tables that are as close to the specified amount of disk space as possible. The summary tables that are built might use more or less space than you specify.

The amount of disk space that you specify is directly related to the optimization results. Increasing the disk space can increase both the number of queries with improved performance and the degree of improvement. You should consider the following factors when choosing the amount of disk space:

- The query performance levels that you want
- The number of cube models that you are optimizing for
- How critical each cube model is
- How frequently each cube model is used
- The availability and cost of the disk space

Typically, you can see significant improvement by allowing a moderate amount of disk space, such as 1% to 10% of the space currently used by the relational tables that are referenced by the cube model. Table 37 shows the relationship between the amount of disk space used for summary tables and the expected query performance improvement. When deciding how much space to provide, consider each cube model in the context of all of your metadata and base tables.

Table 37. Disk space. Percentages of disk space used and the corresponding expected performance improvements

Percentage of base tables disk space used for summary tables	Expected improvement for relevant queries
Less than 1%	Low
5%	Medium
50%	High
Unlimited	Highest

Time limitations

The time that you specify is the maximum amount of time that the Optimization Advisor can use to determine the recommendations. The more time that you allow for the Optimization Advisor to run, the better the results. The following table provides some approximate guidelines for the amount of time you should provide to the Optimization Advisor. Performance results will vary, and you might need to allow more time than is specified in Table 38.

Table 38. Time limitations. Guidelines for how much time to provide to the Optimization Advisor

Database optimization scenario	Approximate time limit
Not performing data sampling	5 to 30 minutes
Performing data sampling on a small database that is less than 10 gigabytes in size	1 hour or less
Performing data sampling on a large database that is more than 10 gigabytes in size	Several hours

Data sampling

Data sampling is a way for the Optimization Advisor to examine the data in your cube model. This provides the Optimization Advisor with more information so that it can create the most effective set of recommendations. Recommendations created with data sampling will match the specified disk space more accurately. Without

data sampling, the Optimization Advisor will analyze only the metadata and DB2 statistics to determine the recommendations.

Optimizing a cube model

By optimizing for queries performed on a cube model, you can improve the performance of products that issue OLAP-style SQL queries.

Prerequisites: You must have DB2 constraints specified for the base tables used in the cube model. Constraints must be specified between each fact table and dimension table and between each dimension table in a snowflake schema. The constraints must be specified on nonnullable columns. For more information about defining constraints, see "Constraint definitions for optimization" on page 101.

When you optimize a cube model, the Optimization Advisor wizard creates SQL that can build a set of recommended summary tables for a cube model. Summary tables aggregate commonly accessed data to speed up query performance.

To optimize a cube model:

1. Open the Optimization Advisor wizard by right-clicking a cube model in the OLAP Center object tree and clicking **Optimization Advisor**.
2. On the Query Types page, review the types of queries that you want to optimize each cube for. You can modify the type of query or specify optimization slices for the cube. The cube query type is used to improve the optimization results. For information about optimization slices, see "Optimization slices for cubes" on page 92.
3. On the Summary Tables page, specify if you want immediate or deferred update summary tables. For information about the update options, see "Summary table maintenance" on page 114. Specify which table space to store the summary tables and summary table indexes in.
4. On the Limitations page, specify how much disk space you want to allow for the summary tables and indexes that will be built. Specify if you want to allow data sampling. Also specify the maximum amount of time you want to allow for the Optimization Advisor to determine recommendations. The more space, information, and time that you specify, the more significantly your performance results will improve. For information about specifying the parameters for the Optimization Advisor wizard, see "Parameters for the Optimization Advisor" on page 106.

   This is the last page of input parameters in the Optimization Advisor wizard. Click **Next** to open a progress window so that you can monitor (and if necessary stop) the progress of the Optimization Advisor in determining the recommendations for creating and refreshing the summary tables. You can modify the time limitation for the Optimization Advisor. If you click **Stop**, the Optimization Advisor returns the summary table recommendations that it determined in the given time.
5. On the SQL Scripts page, type a unique file name in the **SQL script to create the summary tables** field and a unique file name in the **SQL script to refresh the summary tables** field.
6. Click **Finish** to save the recommended SQL scripts into the file names that you specified.
7. Run the SQL scripts. If you are creating large summary tables, building the summary tables might require a substantial amount of time to complete. You can use the DB2 Command Center or Command Window to run the SQL scripts. To run the SQL scripts from the DB2 Command Window:

a. Change to the directory where you saved the SQL scripts.
b. Connect to the database of the cube model that you optimized. For example, enter: `db2 connect to CVSAMPLE`.
c. Enter the following command:

   ```
   db2 -tvf filename
   ```

   where *filename* is the name of the Create summary table SQL script.

Example of an SQL script to create the summary tables

The Optimization Advisor wizard provides an SQL script to create the recommended summary tables. The SQL script contains the necessary SQL commands to build one or more summary tables.

Figure 38 on page 110 shows part of a sample SQL script that creates one summary table, and how the metadata objects map to the SQL. In the sample SQL script, the summary table is called DB2INFO.MQT0000000021T01, where 21 is the cube model ID and T01 is the summary table ID. The cube model ID can be up to 10 digits long. The summary table ID identifies the summary table within the cube model. The summary table ID allows for up to 99 summary tables in one cube model. Do not change the summary table name that the Optimization Advisor wizard defines. If you change the table name, DB2 Cube Views cannot identify the summary tables that it creates for the cube model.

```
DROP TABLE DB2INFO.MQT0000000021T01;

UPDATE COMMAND OPTIONS USING c OFF;

CREATE SUMMARY TABLE DB2INFO.MQT0000000021T01 AS

(SELECT
SUM(T1."SALES") AS "SALES",
SUM(T1."COGS") AS "COGS",
SUM(T1."ADVERTISING") AS "ADVERTISING",
SUM(T1."COGS" + T1."ADVERTISING") AS "Total expense",
SUM(T1."SALES" - (T1."COGS" + T1."ADVERTISING")) AS "Profit",

T5."REGION_NAME" AS "REGION_NAME",
T5."STATE_NAME" AS "STATE_NAME",
T6."LINEID" AS "LINEID (LINE)",
T4."YEAR" AS "YEAR",
T4."QUARTER_NUMBER" AS "QUARTER_NUMBER",
'Qtr ' CONCAT (cast (T4."QUARTER_NUMBER" AS CHAR(1))) AS "QUARTER_NAME",
T4."MONTH_NUMBER" AS "MONTH_NUMBER"

FROM "CVSAMPLE"."SALESFACT" AS T1,
"CVSAMPLE"."STORE" AS T2,
"CVSAMPLE"."PRODUCT" AS T3,
"CVSAMPLE"."TIME" AS T4,
"CVSAMPLE"."LOCATION" AS T5,
"CVSAMPLE"."LINE" AS T6

WHERE
T1."STOREID"=T2."STOREID" AND T1."PRODUCTID"=T3."PRODUCTID"
AND T1."TIMEID"=T4."TIMEID" AND T2."POSTALCODEID"=T5."POSTALCODEID"
AND T3."LINEID"=T6."LINEID"

GROUP BY
T5."REGION_NAME",
T5."STATE_NAME",
T6."LINEID",
T4."YEAR",
T4."QUARTER_NUMBER",
'Qtr ' CONCAT (cast (T4."QUARTER_NUMBER" AS CHAR(1))),
T4."MONTH_NUMBER")

DATA INITIALLY DEFERRED
REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM
NOT LOGGED INITIALLY;

COMMENT ON TABLE DB2INFO.MQT0000000021T01 IS 'AST created for cube model CVSAMPLE.Sales Model';

COMMIT;

ALTER TABLE DB2INFO.MQT0000000021T01 ACTIVATE NOT LOGGED INITIALLY;

REFRESH TABLE DB2INFO.MQT0000000021T01;

CREATE INDEX DB2INFO.IDX0000000021T0101 ON DB2INFO.MQT0000000021T01("STATE_NAME");
CREATE INDEX DB2INFO.IDX0000000021T0102 ON DB2INFO.MQT0000000021T01("MONTH_NUMBER");
CREATE INDEX DB2INFO.IDX0000000021T01C ON DB2INFO.MQT0000000021T01("LINEID (LINE)") CLUSTER;

COMMIT;

REORG TABLE DB2INFO.MQT0000000021T01;

RUNSTATS ON TABLE DB2INFO.MQT0000000021T01 AND INDEXES ALL;

COMMIT;
```

Maps to measures

Maps to attributes

Fact and dimension tables

Joins fact and dimension tables. If applicable, joins dimension to dimension tables in a snowflake schema

Maps to one slice of cube model

Creates indexes

Figure 38. Sample SQL script

If more than one summary table is recommended for your cube model, your SQL script to create the summary tables will include a set of these statements for each summary table.

The following sections explain the statements in the sample create summary tables SQL script:

DROP TABLE statement

Each summary table that will be created is first dropped to ensure that a table with that name does not already exist. In Figure 38, the DB2INFO.MQT0000000021T01 table is dropped with the statement: `DROP TABLE DB2INFO.MQT0000000021T01;`.

CREATE TABLE statement

The script creates the summary table using a CREATE TABLE statement. This statement is the largest part of the script and includes the SELECT statement with the SELECT, FROM, WHERE, and GROUP BY clauses, and the update method definition. The summary table is created with the appropriate columns but no data. The SQL script that refreshes the summary tables aggregates the data from the fact and dimension tables and populates the summary table.

The table name is defined in the first line of the CREATE TABLE statement: `CREATE SUMMARY TABLE DB2INFO.MQT0000000021T01`.

The SELECT clause shown in Figure 38 on page 110 has five lines that begin with SUM. Each of these lines maps to one of the cube model's measures. For example, `SUM(T1."SALES"-(T1."COGS"+T1."ADVERTISING')) AS "Profit"` maps to the Profit calculated measure with the aggregation function SUM. The cube model that the summary table is being created for has the following measures: Sales, COGS, Advertising, Total expense, Profit. The next seven lines that select a column without performing any calculations, map to attributes. For example, `T5."REGION_NAME" AS "REGION_NAME"` maps to the Region name attribute. The summary table includes the following attributes from the cube model: Region name, State name, Line ID, Year, Quarter number, Quarter name, and Month name..

The tables in the FROM clause are the fact and dimension tables used in the cube model. This example uses the SalesFact, Store, Product, Time, Location, and Line tables.

The WHERE clause defines the joins between the fact and dimension tables, and each join maps to a join object in the cube model. The cube model being optimized is based on a snowflake schema, so the dimension-to-dimension joins are also included in the WHERE clause.

The GROUP BY clause maps to slices defined for the cube model. Figure 38 on page 110 shows one grouping set that maps to a particular slice. The groupings can include the following types of metadata to define the slice:

- Level key attributes from the hierarchy at the slice level
- Level key attributes that are above the slice level
- Related attributes that are not functionally dependent on a level key attribute

This section of the SQL script might contain GROUPING SETS so that the summary table can contain multiple levels. If a cube model contains nondistributive measures, this section of the SQL script might contain a ROLLUP.

In the example shown, the cube model being optimized has the following hierarchies: Market [Region, State, City, Postal code, Store], Product [Family, Line, Product], Time [Year, Quarter, Month, Day], and Fiscal Time [Fiscal year, Fiscal quarter, Fiscal month]. If a level from a hierarchy is not included in the grouping set, then the slice is at the highest level, such as All Time, All Regions, or All Products. The slice in the GROUP BY clause is the State–Line–Month slice, and includes Region name, Year, Quarter number, and Quarter name attributes. Region name is above the State level, and Year, Quarter number and Quarter name are attributes that are above the Month level. Each of the level attributes in the slice are attributes that the SELECT clause maps to.

The last part of the CREATE TABLE statement is the update method definition. In Figure 38 on page 110, the last three lines of the CREATE TABLE statement set the summary table as refresh deferred:

```
DATA INITIALLY DEFERRED
REFRESH DEFERRED
ENABLE QUERY OPTIMIZAITON
MAINTAINED BY SYSTEM
NOT LOGGED INITIALLY;
```

If you define a summary table as refresh immediate, the statements would be:

```
DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM
NOT LOGGED INITIALLY;
```

CREATE INDEX statements

The Optimization Advisor wizard recommends one or more indexes for your summary table, they will be created after the summary table is created. In Figure 38 on page 110, both clustered and nonclustered indexes are created. After the indexes are created, the REORG statement is used to reorganize the table according to the clustering index. In some cases, this can improve read performance on the table.

RUNSTATS statement

After all the recommended aspects of the summary table are created, the RUNSTATS statement updates the DB2 optimizer statistics that the DB2 optimizer uses to consider the summary tables and indexes for query rerouting.

Testing query results

You can use the db2batch Benchmark Tool in DB2 Universal Database to benchmark your query performance results before and after you create the summary tables with the Optimization Advisor.

To test the performance of your queries:

1. Create an input file with the queries that you want to test separated by semicolons.
2. Enter the following command on a command line:

   ```
   db2batch -d dbname -f file_name -cli
   ```

   where *dbname* specifies the database to run the queries against, *file_name* specifies the input file with your SQL queries, and `-cli` specifies to run in CLI mode. The db2batch tool summarizes performance results and provides both arithmetic and geometric means. For syntax and options, enter `db2batch -h` on a command line. See the DB2 Information Center for more information on the db2batch Benchmark Tool and creating benchmark tests.

If you are satisfied with the performance results after creating the recommended summary tables, you do not need to do any additional performance analysis.

If your queries do not improve as much as you expected, you can run the Optimization Advisor wizard again and allow more disk space and time and enable data sampling if you did not enable it before. Allowing more disk space will most likely have the largest effect on performance. The more space that you provide for the summary tables, the more improvement you will see. If you allow the wizard to perform data sampling, the wizard can make better recommendations. Likewise, the more time that you allow for the wizard to create the recommendations, the better the recommendations are likely to be.

If you are not satisfied with the performance results because your queries do not improve at all or only very little, or if your queries perform satisfactorily for a period of time and then decrease in performance, see “Troubleshooting summary tables” on page 113.

Troubleshooting summary tables

If the performance of your queries does not improve after you create summary tables, you can use the DB2EXPLAIN facility to troubleshoot the query routing.

Before you use DB2EXPLAIN to verify that DB2 UDB is using the summary tables, you should:

- Verify that the statistics are up to date on the base tables and the summary tables.
- Identify which queries are performing unacceptably if you do not already know. You can use the DB2 SQL Snapshot Monitor to capture slow queries.

To determine why your queries do not perform as you expect them to:

1. Create the explain tables. To set up the explain tables for your database, connect to the database and run the following command from your `\SQLLIB\misc` directory:

   ```
   db2 -tvf explain.ddl
   ```

2. Run the explain facility. When explain mode is turned on, the SQL queries do not run, and only information requests for the explain command are processed. Run the following series of SQL commands to turn explain mode on, set the refresh age so that DB2 UDB considers summary tables if they are refresh-deferred, run the query, turn explain mode off, and query the explain table to see if the query was rerouted

   ```
   set current explain mode explain

   set current refresh age any

   SELECT SUM(SALES) FROM MDSAMPLE.SALESFACT

   set current explain mode no

   SELECT EXPLAIN_TIME, EXPLAIN_LEVEL AS "LEV",
       QUERYNO, STATEMENT_TEXT
   FROM EXPLAIN_STATEMENT
   WHERE STATEMENT_TEXT LIKE '%SALESFACT%'
   ORDER BY EXPLAIN_TIME
   ```

3. View the explain information and check that your rewritten query is rerouted to a summary table. For example, you might see a report like the following sample:

   ```
   2002-06-30-23.22.12.325002 O   11 SELECT SUM(SALES)
     FROM MDSAMPLE.SALESFACT
   2002-06-30-23.22.12.325002 P   11 SELECT Q3.$C0
     FROM (SELECT SUM(Q2.$C0) FROM (SELECT Q1.SALESFACT_SALES
     FROM DB2INFO.MQT0000000021T01 AS Q1) AS Q2) AS Q3
   ```

   There are two lines for one execution of the query. The line marked with an `O` is the original query that is sent to DB2 UDB. The line marked with a `P` is the query as rewritten by the DB2 optimizer. You can see in the rewritten query from this example that the DB2 optimizer selected data from the `DB2INFO.MQT0000000021T01` summary table.

   If the query is rerouted to the summary table but does not perform as you expected, you might need to run the Optimization Advisor wizard again with different options.

   If the query is not rerouted to a summary table, determine the reason and take the appropriate action. The reasons why a query might not be rerouted to a summary table include:

   The summary table does not exist

First, make sure that the summary table exists. If it does not exist, run the Optimization Advisor wizard to generate the Create summary tables SQL script. Then run the script to create the summary tables.

A refresh-deferred summary table expired

If your summary table exists and you set it up to be refresh-deferred, you might need to update the refresh age. You can set the table's refresh age to be as large as possible and session independent by setting `(DFT_REFRESH_AGE) = 99999999999999 (ANY)`.

The query accesses data that is not included in the summary table

If your query is accessing data that is not in your summary table, the DB2 optimizer will not reroute the query. If you added a new measure after you created your summary tables, that new measure does not exist in your summary tables. If you try to query the new measure, the DB2 optimizer cannot reroute the query to the summary table because the summary table does not contain all of the data to satisfy the query.

Additionally, if you try to query data that is below the cube model slice that the summary table is built for, you cannot use the summary table. For example, if the query requests data aggregated at the City level but the summary table includes data that is aggregated at the State level (which is above the City level), the query cannot use the summary table.

The query contains constructs that cannot be rerouted

The DB2 optimizer cannot reroute queries that use some complex query constructs. Some complex constructions that inhibit the DB2 optimizer from rerouting the queries are recursion and physical property functions like:

- NODENUMBER
- Outer joins
- Unions
- XMLAGG
- Window aggregation functions, which are aggregation functions specified with the OVER clause

Summary table maintenance

When the data in your base tables changes, you need to update your summary tables. You can update your summary tables in two different ways: refresh immediate or refresh deferred.

You choose to create refresh-immediate or refresh-deferred summary tables when you run the Optimization Advisor wizard. The choice that you make affects the update setting for the tables and the refresh summary tables SQL script. For both options, you need to run the refresh summary tables script as part of your normal database maintenance schedule. Running the refresh script can require significant time and processing resources. Make sure you allocate enough time in your maintenance batch window to complete the updates.

Refresh-immediate

Refresh-immediate summary tables are kept closely synchronized with your base tables. DB2 UDB tracks the changes to the base tables so that it can incrementally update the summary tables by changing only the portion

of the summary tables that corresponds to the changed portion of the base tables. If it is important to you that the summary table data be kept in unison with your base tables, use the refresh-immediate option. Refresh-immediate might be a good choice if, for example, your base tables are updated with weekly sales data, and users complete weekly reports that reflect the updated sales data.

If you commonly have many changes scattered throughout your base tables, refresh-immediate is probably not the best choice because it can require significant overhead for DB2 UDB to track the changes and individually perform the update statements to aggregate the changes again.

If you update your base tables using regular SQL statements, such an INSERT, UPDATE, and DELETE, DB2 UDB automatically synchronizes the affected summary tables after you change your base tables. However, if you update your base tables using the DB2 LOAD or IMPORT commands, you need to manually trigger the synchronization by running the refresh script after you complete the update.

Immediate update cannot be used in all situations, and the Optimization Advisor wizard might recommend the deferred option if necessary.

Refresh-deferred

Refresh deferred summary tables are usually updated less frequently than refresh immediate because you need to manually synchronize the summary tables with the base tables. The summary tables are based on a snapshot of the data at the time that they are created. Each update re-creates the summary table based on the current data, but has no knowledge of how the data changed since the summary table was last created.

Refresh deferred is a good choice when you are making significant changes throughout the corresponding base tables or if you are updating data more quickly than you need to access it. For example, if your sales data is updated weekly but you need to create reports on a quarterly basis only, you can use the refresh-deferred option and rebuild your summary tables each quarter before running your report.

Dropping a summary table

DB2 Cube Views does not drop the associated summary tables when you drop a cube model. If you do not use the summary tables for any other purpose, you can drop the tables to free disk space.

Summary tables are a type of table, and can be dropped using the normal DB2 procedures using the Control Center or the command line. Any associated indexes are also dropped with the summary table.

The summary tables are defined in the DB2INFO schema. The summary table name includes the cube model ID. For example, a summary table might be named DB2INFO.MQT0000000021T01, where 21 is the cube model ID and T01 uniquely identifies the summary table within the cube model. The cube model ID can be up to 10 digits long.
To drop a summary table from a command line, enter `DROP TABLE table_name`.

Chapter 6. DB2 Cube Views and federated data sources

This section describes the following topics:

Overview of federated systems
You can use IBM DB2 Information Integrator as your enterprise-level solution to information integration. DB2 Information Integrator is a collection of technologies that combines data management systems and federated systems and several other technologies into a common platform.

Overview of optimizing remote data sources with DB2 Cube Views
You can use DB2 Cube Views to optimize a federated star schema or snowflake schema to achieve significant improvements in your query performance.

Enabling a federated system for DB2 Cube Views
To enable the federated system for DB2 Cube Views, you must define the remote data source, define nicknames on the federated server, and define informational constraints on the nicknames.

Troubleshooting query rerouting for federated data sources
If your queries issued to a remote data source do not improve as you expected after you optimize, ensure that the federated system is set up correctly for DB2 Cube Views.

Overview of federated systems

You can use IBM DB2 Information Integrator as your enterprise-level solution to information integration. DB2 Information Integrator is a collection of technologies that combines data management systems and federated systems and several other technologies into a common platform.

A DB2 federated system is a special type of distributed database management system (DBMS). You can use the federated systems aspect of the information integration technology to access diverse types of data spread across various data sources. A federated system consists of the following components:

- A DB2 instance that operates as a federated server
- A database that acts as the federated database
- One or more data sources
- Clients (users and applications) that access the database and data sources

With a federated system, you can send distributed requests to multiple data sources within a single SQL statement. For example, you can join data that is located in a DB2 Universal Database table, an Oracle table, and an XML tagged file in a single SQL statement.

Federated servers

The DB2 server in a federated system is referred to as the federated server. Any number of DB2 instances can be configured to function as federated servers. You can use existing DB2 instances as your federated servers, or you can create new ones specifically for the federated system.

The DB2 instance that manages the federated system is called a server because it responds to requests from end users and client applications. The

federated server often sends parts of the requests it receives to the data sources for processing. A pushdown operation is an operation that is processed remotely. The DB2 instance that manages the federated system is referred to as the federated server, even though it acts as a client when it pushes down requests to the data sources.

The federated server interacts with data sources through wrappers. The federated server uses routines stored in a library called a wrapper module to implement a wrapper. These routines allow the federated server to perform operations such as connecting to a data source and retrieving data from it iteratively. Typically, the DB2 federated instance owner uses the CREATE WRAPPER statement to register a wrapper in the federated database.

Federated databases

To end users and client applications, data sources appear as a single collective DB2 UDB database. Users and applications interface with the federated database managed by the federated server. The federated database contains a system catalog. The federated database system catalog contains entries that identify data sources and their characteristics. The federated server consults the information stored in the federated database system catalog and the data source wrapper to determine the best plan for processing SQL statements.

Data sources

In a federated system, a data source can be a relational DBMS instance (such as DB2, Informix, Oracle, or Sybase) or a nonrelational data source (such as BLAST search algorithm or an XML tagged file).

The method, or protocol, used to access a data source depends on the type of data source. For example, DRDA is used to access DB2 family data sources such as DB2 for z/OS and OS/390.

Clients

Clients can include users and applications that access the federated database and data sources that you set up. DB2 Cube Views is an example of an application that can act as a client accessing a federated database and data source.

Overview of optimizing remote data sources with DB2 Cube Views

You can use DB2 Cube Views to optimize a federated star schema or snowflake schema to achieve significant improvements in your query performance.

The benefits include:

- Having an integrated DB2 platform to access multiple IBM and vendor products from
- Extending the functional richness of DB2 UDB for Linux, UNIX, and Windows to DB2 UDB for z/OS and DB2 UDB for iSeries
- Improving the performance of queries to your federated databases and data sources

In your federated system, the fact and dimension tables might be on a remote server or across several remote servers. You must locally represent your star schema or snowflake schema for DB2 Cube Views. You can locally represent your star schema or snowflake schema with nicknames that refer to the remote tables, replicated copies of the remote tables, or a combination of nicknames and

replicated tables. You must locally represent all of the fact tables and dimension tables in your star schema or snowflake schema in some way on your local server.

After your star schema or snowflake schema exists on your local server, you can use DB2 Cube Views to build a cube model based on that star schema or snowflake schema, and optimize that cube model using the Optimization Advisor. If you locally represent some of your tables with nicknames, specifying that the Optimization Advisor wizard can use data sampling provides much better summary table recommendations, but requires more time to create the recommendations. The Optimization Advisor might not be able to use data sampling in every situation, but it is recommended that you allow data sampling when possible.

You can create the recommended summary tables on the local server, and a query that is directed to tables in the remote star or snowflake schema can be answered in one of the following three ways:

Query routes to a local summary table

If the query can be answered by the data in the summary table, the DB2 optimizer will route the query directly to the local summary table, and will not need to interact with the remote tables at all. This routing scenario provides the most significant performance improvement.

Often, a query cannot be answered by only the summary table, but can be answered by joining the summary table with one or more dimension tables. In this case, you might consider collocating some or all of your dimension tables (but not your fact table) on the federated server so that the DB2 optimizer can complete the joins on local tables.

Query is pushed down to the remote data source

If the query cannot be answered by the data in the summary table and any collocated dimension tables, the DB2 optimizer will attempt to push the query down to the remote server. The remote server completes the query, and returns the result set to the local server.

If the result set is small, you can still achieve significant performance improvements. This method is possible if all of the tables in the remote star schema or snowflake schema exist on only one remote server.

Query is run locally on data pulled up from remote data sources

If the query cannot be answered by the data in the summary table and any collocated dimension tables and cannot be pushed down to the remote data source, the tables that are needed to satisfy the query are pulled up and copied from the remote server to the local server, and the query is completed locally. For example, if the star schema is located on a remote zSeries server and the query uses a function that is available in DB2 UDB but not in DB2 for z/OS, the query cannot be completed on the remote server.

This routing scenario might not provide performance improvement if the fact and dimension tables needed to answer the query are large.

Enabling a federated system for DB2 Cube Views

To enable the federated system for DB2 Cube Views, you must define the remote data source, define nicknames on the federated server, and define informational constraints on the nicknames.

Prerequisites: Make sure that the DB2 Information Integrator support is enabled. To enable the federated system support from a command line, type: `db2 update dbm cfg using federated yes`. Stop and start DB2 UDB after enabling the federated system support.

To enable a remote data source for DB2 Cube Views:

1. Define remote data sources.
2. Define nicknames for remote tables.
3. Define informational constraints on nicknames.

Defining remote data sources

To define and configure a federated server for DB2 Cube Views, you must provide the federated server with information about the remote data sources and objects that you want to access.

To define a remote data source:

1. Catalog the remote database and the corresponding node in the federated server database directory so that the federated server knows which remote data source to connect to. You can use the Add Database wizard in the Configuration Assistant to catalog the remote database and the corresponding node.
2. Connect to the local database on the federated server. If you do not already have an existing local database, create a local database and then connect to that database.
3. Register the wrapper by issuing the CREATE WRAPPER command. For example, if your remote data source is from the DB2 family, such as z/OS, issue the following command:

   ```
   CREATE WRAPPER drda LIBRARY 'libdb2drda.a'
   ```

   You must know what library is required by your data source.
4. Register the server definitions for each server used by the remote data sources that you want to access. For example, create a drda wrapper for DB2 family servers. Issue the following command to register a remote z/OS server:

   ```
   CREATE SERVER server_name
     TYPE DB2/ZOS
     VERSION 8.1
     WRAPPER DRDA
     AUTHORIZATION "userid" PASSWORD "password"
     OPTIONS (DBNAME 'database_name')
   ```

   Tip: You can run this command as described from an SQL script file. To issue this command directly from a DB2 command window, type the command surrounded in double quotation marks, and use escaped double quotation marks (\") around the user ID and password that you supply:

   ```
   DB2 "CREATE SERVER server_name
     TYPE DB2/ZOS
     VERSION 8.1
     WRAPPER drda
     AUTHORIZATION \"userid\" PASSWORD \"password\"
     OPTIONS (DBNAME 'database_name')"
   ```
5. For each server that you defined, set the query workload to always be pushed down from the federated server to the remote data source. The data on the remote data source is likely to be large and you do not want DB2 UDB on the federated server to try to copy the data from the remote data source to the federated server. By setting the server to maximal pushdown, DB2 UDB will

always try to first push the query down to the remote data source so that only the query result set is copied to the federated server. This setting is needed if you want to achieve performance improvements from DB2 Cube Views optimization. For example:

```
CREATE SERVER OPTION DB2_MAXIMAL_PUSHDOWN
  FOR SERVER server_name
  SETTING 'Y'
```

6. Create the user mapping so that the wrapper can automatically connect to the server. For example:

```
CREATE USER MAPPING FOR USER
  SERVER server_name
  OPTIONS(REMOTE_AUTHID 'user_name', REMOTE_PASSWORD 'password')
```

Defining nicknames for remote tables for DB2 Cube Views

Define nicknames for each remote table that is part of the star schema or snowflake schema that you are creating on your federated server. DB2 Cube Views requires the nicknames to work with the remote tables.

To define a nickname for a table on a remote data source:
Use the CREATE NICKNAME statement to define a nickname for each remote table that you want to access. For example:

```
CREATE NICKNAME local_schema_name.local_nickname
  FOR remote_server_name.remote_schema_name.remote_table_name
```

Tip: Use the same schema name for your local nicknames as are defined on the remote server. For example:

```
CREATE NICKNAME CVSAMPLE.FAMILY FOR remote_server_name.CVSAMPLE.FAMILY
```

Defining informational constraints on nicknames for DB2 Cube Views

Define informational constraints to document relationships between your data that can improve performance. You should define informational constraints for the foreign keys between tables in the star schema or the snowflake schema on the federated server.

The DB2 Cube Views Optimization Advisor wizard requires informational constraints to recommend performance-improving summary tables. The DB2 Optimizer also uses the constraints to process queries more efficiently and to appropriately route queries to existing summary tables.

DB2 Information Integrator automatically defines primary key constraints on the federated server that match existing primary key constraints on the remote data source. You must create informational constraints for foreign keys that you use to build the star schema or snowflake schema on the federated server.

To define a foreign key:
Use the ALTER NICKNAME statement to add informational constraints to your nicknames. For example:

```
ALTER NICKNAME local_schema_name.local_nickname
  ADD FOREIGN KEY(column_name)
  REFERENCES local_schema_name.local_nickname (column_name)
  ON DELETE RESTRICT
  NOT ENFORCED
  ENABLE QUERY OPTIMIZATION
```

You have completed enabling a remote data source for DB2 Cube Views.

Next, use DB2 Cube Views to build a complete cube model for the star schema or snowflake schema on your federated server. After building a complete cube model, you can use the Optimization Advisor wizard to optimize the cube model.

Troubleshooting query performance for remote data sources

If your queries issued to a remote data source do not improve as you expected after you optimize, ensure that the federated system is set up correctly for DB2 Cube Views.

You must complete the following steps before you can expect queries issued to remote data sources to improve:

1. Enable your federated system for DB2 Cube Views.
2. Create a complete cube model that satisfies the base rules, cube model completeness rules, and optimization rules described in "Metadata object rules" on page 37.
3. Optimize a cube model.
4. If query performance does not improve, check the steps described in "Troubleshooting summary tables" on page 113.

If, after completing the steps above, the performance of your queries still do not improve, consider the following issues:

- Make sure that all applicable constraints are defined.
- Make sure that the DB2_MAXIMAL_PUSHDOWN setting is set to yes as described in "Defining remote data sources" on page 120.
- Consider collocating the dimension tables that are involved in the queries on your federated server. Collocating the dimensions, so that a replicated copy of the dimension table exists on the federated server, might improve performance.

Chapter 7. DB2 Cube Views API

DB2 Cube Views API overview

DB2 Cube Views offers an application programming interface (API) that provides programmatic access to metadata stored in DB2 Cube Views. Using the API, applications can use DB2 Cube Views' metadata objects to interact with metadata without needing to interact with relational tables and joins.

The DB2 Cube Views API provides access to the metadata that is stored in the system catalog tables of a DB2 database. Applications that use the API can create and modify metadata objects that model multidimensional and OLAP structures in a data warehouse.

Figure 39 shows how data and metadata is exchanged through the API.

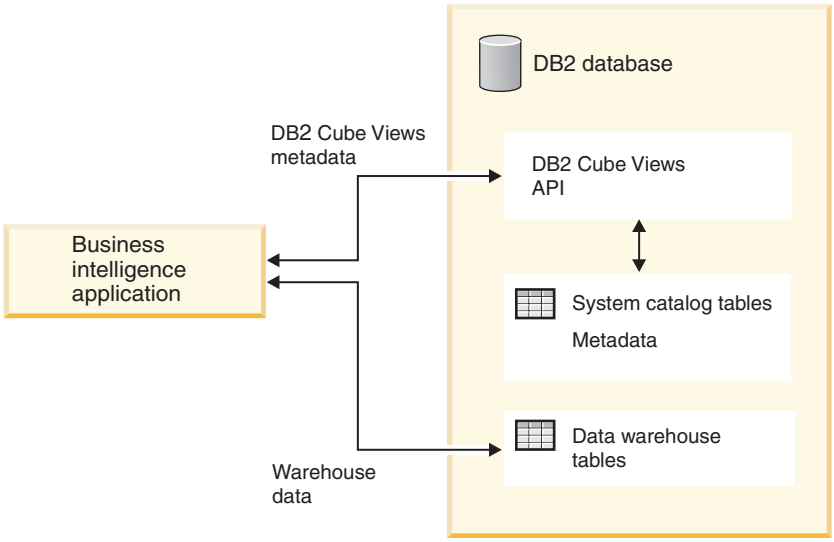

Figure 39. Data exchange though the DB2 Cube Views API

The API is a single stored procedure that is registered to a DB2 database. This stored procedure accepts input and output parameters that you use to express complex metadata and metadata operations. The API parameter format is defined by XML schema.

The API uses the following technologies to exchange metadata between DB2 Cube Views and business intelligence applications:

- SQL with ODBC
- DB2 CLI and JDBC
- XML

DB2 Cube Views API: DB2 stored procedure and XML parsing

The DB2 Cube Views API exchanges metadata between a business intelligence application and a DB2 database through the DB2 Cube Views stored procedure and XML parsing.

For information about programming with DB2 stored procedures, see the *DB2 Application Development Guide*. Before you program with the DB2 Cube Views API, you should understand the following concepts:

Transaction

DB2 UDB supports transactional, multi-user metadata access. (See the *DB2 Application Development Guide* for more information). All database actions that are performed with the DB2 Cube Views API belong to the calling application's database transaction. Therefore, an application can execute COMMIT or ROLLBACK after the API calls the md_message stored procedure to manage the units of database work.

Memory management

Parameters are exchanged between applications and the md_message stored procedure in the form of CLOB structures. Applications that call the md_message stored procedure must preallocate CLOB parameter structures that are the same size as those used to catalog the stored procedure. The API supports the DB2 UDB maximum size for a CLOB, which is 2 GB. The default CLOB size is 1 MB.

System configuration

To support the exchange of large parameters, you might need to change the following DB2 UDB settings:

- The database client application that calls the md_message stored procedure might need to be linked by using larger heap and stack sizes.
- The DB2 query heap size for the database might need to be increased by using the query_heap_sz setting.

XML parsing

Applications that use the API must parse the output parameter that is returned by the md_message stored procedure. A variety of XML parsers are available to developers who want to use the API.

Error handling

Error information is generated in three forms by the API:

- SQLCODE and SQLSTATE information that are returned by the stored procedure to the calling application
- XML structures that are delivered to calling applications by using the **response** API parameter
- Error and run-time log files that are located on the database server that is running the API

If an error occurs due to XML validation, parsing, or tagging, the **response** parameter will be returned to the calling application with an <error> tag in place

of an operation tag. This <error> XML tag will contain a <status> tag with a return code and message that describe the problem encountered by the API.

If an error occurs in the API that is unrelated to XML processing but it is related to the execution of a metadata operation, the contents of the **response** parameter are returned.

The following example shows the type of information in an <error> tag. In this example, descriptions of the parameter structures use only a limited number of XML tags. Most parameters have more XML tags than are shown here, and parameter content will be validated with the XML schema.

```
<olap:response xmlns:olap="http://www.ibm.com/olap"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="8.1.2.1.0">
<error>
<status id="3100" text="The system failed to parse XML for
   &quot;INPUT PARAMETER&quot; (line:&quot;3&quot;,
   char:&quot;26&quot;, message:&quot;Unknown element
   'dropa'&quot;). " type="error"/>
</error>
</olap:response>
```

When the DB2 Cube Views stored procedure is called, regardless of whether the stored procedure was actually executed, DB2 UDB returns an SQLCODE and an SQLSTATE to the calling application. If the DB2 Cube Views stored procedure was able to execute, the stored procedure returns a status message as part of the XML data that is sent to the calling application.

DB2 Cube Views stored procedure

The stored procedure is called md_message and it processes parameters that are expressed in the DB2 Cube Views parameter format.

The procedure extracts operation and metadata information from the input parameters, and then it performs the requested metadata operations. The procedure generates output parameters that contain the execution status (success or failure) of requested operations and metadata information, depending on the operation.

The stored procedure runs as the user that is specified in the .fenced file on UNIX systems. Fenced users must have WRITE access to the log files that are specified in the db2md_config.xml file. Otherwise, the stored procedure cannot log anything.

The DB2 Cube Views stored procedure is implemented as a DB2 UDB stored procedure. It can be used by any application that uses any of the programming interfaces for DB2 UDB. The name of the stored procedure is case insensitive, but the name and contents of the stored procedure's parameters are case sensitive. The syntax of md_message and a prototype are:

```
Syntax:     call md_message (request, metadata, response)
Prototype:  md_message (request  IN   CLOB(1M),
                        metadata INOUT  CLOB(1M),
                        response  OUT   CLOB(1M))
```

The **request**, **metadata**, and **response** parameters are character large object (CLOB) type, which is a DB2 UDB data type. An application populates the **request** parameter with a description of the operation to be performed, and it can optionally populate the **metadata** parameter with the metadata that the operation will act on. After it implements the input parameters, md_message returns the

status of the operation in the **response** parameter and returns the requested metadata in the **metadata** parameter. The **metadata** parameter is used for both input and output of metadata. DB2 UDB transfers the parameter structures between business intelligence applications and the md_message stored procedure on the database server.

The size of the CLOB arguments can vary. The default size is 1 MB. The CLOB argument sizes are specified by the sqllib/misc/db2mdapi.sql script. You can recatalog the stored procedure with any size for the CLOB parameters up to 2 GB. When you increase the size of the parameter, more memory is used by the stored procedure at run time because output parameter buffers are preallocated to the cataloged size when the stored procedure is started. If the size is too small, data in the input and output parameters will be truncated.

To change the default CLOB size, re-register the stored procedure API with increased size limits.

1. Edit the sqllib/misc/db2mdapi.sql file and increase the size limits of the CREATE PROCEDURE statement.

```
CREATE PROCEDURE
  DB2INFO.MD_MESSAGE(IN    request  CLOB(1M),
                     INOUT metadata CLOB(10M),
                     OUT   response CLOB(10M))
```

2. Re-run the db2mdapi.sql file to re-register the stored procedure API. Ignore any errors as the script attempts to create tables that already exist.

For information about calling the stored procedure API from C++, see sample C++ source code in the sqllib/samples/olap/client/db2mdapiclient.cpp file.

The following example shows how to call the DB2 Cube Views stored procedure from an embedded SQL application:

```
// Standard declarations
// ...

// Include the Communication Area to access error details
EXEC SQL INCLUDE SQLCA;

// SQL declarations of host variables that will be used for calling the
// DB2 Cube Views stored procedure
EXEC SQL BEGIN DECLARE SECTION;

// Allocate CLOB for the request parameter
SQL TYPE is CLOB(1M)       request;

// Allocate CLOB for the metadata parameter
SQL TYPE is CLOB(1M)       metadata;

// Allocate CLOB for the response parameter
SQL TYPE is CLOB(1M)       response;

EXEC SQL END DECLARE SECTION;

// Connect to database and other application initializations
// ...

// Populate the request parameter structure with the operation
strcpy(request.data, "<request><describe> ... </describe></request>");

// string length with end-of-string
```

```
request.length = strlen(request.data) + 1;

// Populate the metadata parameter structure with the metadata
strcpy(metadata.data, "");
// string length with EOS
metadata.length = strlen(metadata.data) + 1;

// Call DB2 Cube Views stored procedure
EXEC SQL CALL "DB2INFO.MD_MESSAGE"(:request,:metadata,:response);

// Check that the stored procedure has returned without errors
if (sqlca.sqlcode)
{
// error checking using sqlaintp()
}

// Process response parameter structure to determine success of operation
// ...

// Process metadata parameter structure to extract requested metadata
// ...

// Disconnect from database and other application terminations
// ...
```

DB2 Cube Views API parameters

Parameters for DB2 Cube Views API metadata operations

The API for DB2 Cube Views offers three types of metadata functions: retrieval, modification, and administration. Each type of function includes one or more operations, and each operation has a set of parameters.

The parameter format defines the standard by which metadata operations and objects are represented and exchanged between business intelligence applications and DB2 Cube Views. The parameter format uses XML to represent DB2 Cube Views metadata operations and objects. The XML schema defines the parameter format.

Input and output parameters

Each operation in the stored procedure md_message has two input and two output parameters.

The input parameters are the **request** parameter and the **metadata** parameter. The output parameters are the **response** parameter and the **metadata** parameter.

Input	Parameter content
Request	Contains the operation that is requested of the stored procedure. The request can include options that affect the behavior and scope of the request.
Metadata	Contains metadata objects that will be used with the operation described in the **request** parameter. Some request operations, such as Describe, don't require input metadata.
Output	
Response	Contains all of the results of the operation that was performed by the stored procedure, except metadata objects.

Input	Parameter content
Metadata	Contains metadata objects that were requested by the operation that is described in the **request** input parameter. Some operations, such as Create, don't return output metadata.

Figure 40 shows how the input parameters from the requesting application pass through the DB2 Cube Views API to the output parameters. The API creates a representation of the metadata objects that can be read by DB2 UDB.

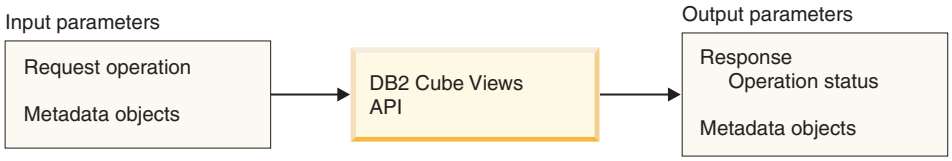

Figure 40. API parameters

DB2 Cube Views metadata operations

Retrieval operation: Describe

DB2 Cube Views includes one retrieval operation: Describe.

The Describe operation supports both XML version 8.1.2.1.0 and 8.2.0.1.0. All other operations require XML version 8.2.0.1.0. For more information about versions of the XML schema files, see "DB2 Cube Views metadata tables and XML schema files" on page 153.

Describe

This operation retrieves metadata object information. This operation returns information for one or more metadata objects of the specified *objectType* (for example, single dimension object, a set of dimension objects, a set of objects including all object types). Metadata objects are returned in the **metadata** parameter. The format that is used to represent retrieved metadata objects is described in "Metadata object format" on page 155. In recursive mode, this operation also returns information for all other metadata objects as it goes through the object-association paths beginning from the set of objects that satisfy the *objectType* specification.

Table 39 on page 129 lists the components in a Describe **request** parameter. Table 40 on page 129 lists the components of the Describe **response** parameter.

Table 39. Describe request parameters

Name	Type	Data type	Valid values	Description

objectType	argument	XML string	`all`
			`cubeModel`
			`cube`
			`cubeFacts`
			`cubeDimension`
			`cubeHierachy`
			`cubeLevel`
			`dimension`
			`facts`
			`hierarchy`
			`attributeRelationship`
			`level`
			`join`
			`measure`
			`attribute`
				Types of DB2 Cube Views metadata objects that can be retrieved. You can specify one of the valid values for this parameter.

restriction [optional]	argument	XML string	See "Operation parameters" on page 144.	Limits the scope of a metadata retrieval. This is analogous to predicates in SQL.
recurse	argument	XML string	`yes` `no`	Enables or disables recursive retrieval of metadata objects.

Table 40. Describe response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure	See section "Message structure" on page 147.	Message that indicates the status of the Describe operation.
object	retrieval results	XML	See "Operation operands" on page 147.	Requested metadata data objects. This value is an empty string if an error occurred during the operation.

Advise operation: Advise

This operation recommends summary tables that should be built to improve query performance for the specified cube model. This operation has arguments that restrict how long the advisor can run and how much disk space it can use for the summary tables.

Table 41. Advise request parameters

Name	Type	Data type	Valid values	Description
cubeModelRef	argument	XML element		Cube model to optimize.
tablespaceName [optional]	argument	XML string		Table space for the summary tables. If no table space is provided then the generated SQL will not specify a table space. In that case DB2 UDB will put the tables in a default table space.
indexspaceName [optional]	argument	XML string		Table space for the summary table indexes. If no table space is provided, the generated SQL will not specify a table space. In that case, DB2 UDB will put the indexes in a default table space.

Table 41. Advise request parameters (continued)

Name	Type	Data type	Valid values	Description
diskspaceLimit [optional]	argument	XML non negative integer		Disk space (in bytes) that is available for the summary tables and their indexes. Specifying 0 means that the advisor decides how much disk space to use. The advisor produces better recommendations when you provide it more disk space. Specifying 0 means unlimited. The default is 0.
timeLimit [optional]	argument	XML non negative integer		The amount of time (in seconds) that should be used to recommend queries. The advisor produces better recommendations when you provide it more time. Specifying 0 means unlimited and the advisor stops when it decides that spending more time will not produce better recommendations. The default is 0.
sampling [optional]	argument	XML string	yes or no	Specifies if data sampling of the cube model base tables should be done. Using sampling generally improves the recommendations but will increase the time for the advisor to run. If sampling is not allowed, the advisor makes recommendations based on database statistics only. With data sampling the advisor exploits both database statistics and sampling. If you have a small replica of the original data, the DB2 UDB statistics can be changed to make the tables appear as large as the original tables. If sampling is not specified then the advisor will make the same recommendations that it made on the original tables without sampling.

Table 41. Advise request parameters (continued)

Name	Type	Data type	Valid values	Description
refresh [optional]	argument	XML string	deferred or immediate	Specifies whether DB2 UDB should try to refresh the summary tables immediately when the base tables change to ensure that they are synchronized. Refresh deferred means that you must specify when the summary tables are refreshed. For refresh immediate, DB2 will update the summary tables at the same time that the base tables are updated. If the base tables are updated using data loads, you must manually specify the refresh. The refresh immediate summary tables are incrementally updated, while refresh deferred summary tables are completely rebuilt. There are many restrictions when using refresh immediate, and the advisor might choose to use deferred even if you specify immediate.

The following example shows an Advise operation:

```
<olap:request xmlns:olap="http://www.ibm.com/olap" ... >
<advise
 tablespaceName="TS_MQTTABLE"
  indexspaceName="TS_MQTINDEX"
  diskspaceLimit="75000"
  timeLimit="300"
  sampling="yes"
  refresh="immediate">
<cubeModelRef name="SalesModel" schema="MDSAMPLE"/>
</advise>
</olap:request>
```

Table 42. Advise response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure		Message that indicates the status of the Advise operation.

Table 42. Advise response parameters (continued)

Name	Type	Data type	Valid values	Description
info	message list	List of message structures		Warning messages that indicate any nonfatal conditions that were encountered and possible changes in the results of the Advise operation from what you specified in the Advise request. For example, an Advise operation might override your specification for REFRESH IMMEDIATE MQTs because of the presence of nondistributive measures in the recommended MQTs. Informational messages might show conditions that caused some parts of the model not to be optimized or show information about what optimizations were done including why the aggregations and indexes were selected.
recommendation		XML element		An estimate of the total disk space that is required for all recommended summary tables and indexes.
sql		XML element		SQL to create and populate a summary table and create indexes for it. You are responsible for running the SQL.
refreshSql		XML element		SQL to refresh the summary tables to synchronize them with base tables that were updated. You are responsible for running the SQL. For refresh immediate summary tables, DB2 UDB sometimes updates the summary tables automatically when the base tables are updated. This action depends on how the base tables are changed.

The following example shows an Advise response operation:

```
<olap:response xmlns:olap="http://www.ibm.com/olap" ... >
<advise>
<status id="0" text="Operation completed successfully.
No errors were encountered." type="informational"/>
<info>
<message id="7401" text="The DB2INFO.MQT0000000041T01 summary table
is recommended.
It is estimated to have 100 rows, 55KB table size and 5KB index size."/>
<message id="7401" text="The DB2INFO.MQT0000000041T02 summary table
is recommended.
It is estimated to have 8 rows, 4KB table size and 1KB index size."/>
<message id="7406" text="The PART dimension does not have any
hierarchies that can be optimized by the Optimization Advisor.
The recommendations will not optimize for any attributes
from this dimension."/>
</info>
```

```
<recommendation diskspace="65108"/>
<sql>
<![CDATA[
-- ********************************************************************
-- * Script to create/refresh summary tables.
-- *
-- * Cube model schema: MDSAMPLE
-- * Cube model name:   SalesModel
-- * Diskspace limit:   75000
-- * Time limit:        300
-- * Sampling:          Yes
-- * Refresh type:      Refresh immediate
-- * Tablespace name:   TS_MQTTABLE
-- * Indexspace name:   TS_MQTINDEX
-- ********************************************************************

DROP TABLE DB2INFO.MQT0000000041T01;

DROP TABLE DB2INFO.MQT0000000041T02;

UPDATE COMMAND OPTIONS USING c OFF;

CREATE SUMMARY TABLE DB2INFO.MQT0000000041T01 AS
(SELECT
SUM(T2."COGS") AS "COGS",
SUM(T2."MARKETING"+T2."PAYROLL") AS "EXPENSE",
SUM(T2."MARKETING") AS "MARKETING",
SUM(T2."PAYROLL") AS "PAYROLL",
SUM(T2."SALES"-(T2."COGS"+(T2."MARKETING"+T2."PAYROLL"))) AS "PROFIT",
SUM(T2."SALES") AS "SALES",
COUNT(*) AS "COUNT",
T5."REGION" AS "REGION",
T5."DIRECTOR" AS "DIRECTOR",
T6."FAMILY" AS "FAMILY",
T6."FAMILYNAME" AS "FAMILYNAME",
T3."SKU" AS "SKU",
T3."CAFFEINATED" AS "CAFFEINATED",
T3."OUNCES" AS "OUNCES",
T3."PKGTYPE" AS "PKGTYPE",
T3."SKUNAME" AS "SKUNAME",
T4."YEAR" AS "YEAR"

FROM
"MDSAMPLE"."MARKET" AS T1,
"MDSAMPLE"."SALESFACT" AS T2,
"MDSAMPLE"."PRODUCT" AS T3,
"MDSAMPLE"."TIME" AS T4,
"MDSAMPLE"."REGION" AS T5,
"MDSAMPLE"."FAMILY" AS T6

WHERE
T1."STATEID"=T2."STATEID" AND
T3."PRODUCTID"=T2."PRODUCTID" AND
T4."TIMEID"=T2."TIMEID" AND
T1."REGIONID"=T5."REGIONID" AND
T3."FAMILYID"=T6."FAMILYID"

GROUP BY
T5."REGION",
T5."DIRECTOR",
T6."FAMILY",
T6."FAMILYNAME",
T3."SKU",
T3."CAFFEINATED",
T3."OUNCES",
T3."PKGTYPE",
T3."SKUNAME",
```

```
T4."YEAR")

DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
IN "TS_MQTTABLE"
INDEX IN "TS_MQTINDEX"
NOT LOGGED INITIALLY;

COMMENT ON TABLE DB2INFO.MQT0000000041T01 IS 'AST created for
cube model MDSAMPLE.SalesModel';

REFRESH TABLE DB2INFO.MQT0000000041T01;

CREATE INDEX DB2INFO.IDX0000000041T0101 ON DB2INFO.MQT0000000041T01("FAMILY",
"SKU");
CREATE INDEX DB2INFO.IDX0000000041T0102 ON DB2INFO.MQT0000000041T01("REGION");

RUNSTATS ON TABLE DB2INFO.MQT0000000041T01 AND INDEXES ALL;

CREATE SUMMARY TABLE DB2INFO.MQT0000000041T02 AS
(SELECT
SUM(T2."COGS") AS "COGS",
SUM(T2."MARKETING"+T2."PAYROLL") AS "EXPENSE",
SUM(T2."MARKETING") AS "MARKETING",
SUM(T2."PAYROLL") AS "PAYROLL",
SUM(T2."SALES"-(T2."COGS"+(T2."MARKETING"+T2."PAYROLL"))) AS "PROFIT",
SUM(T2."SALES") AS "SALES",
COUNT(*) AS "COUNT",
T1."YEAR" AS "YEAR",
T1."QUARTER" AS "QUARTER"

FROM
"MDSAMPLE"."TIME" AS T1,
"MDSAMPLE"."SALESFACT" AS T2

WHERE
T1."TIMEID"=T2."TIMEID"

GROUP BY
T1."YEAR",
T1."QUARTER")

DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
IN "TS_MQTTABLE"
INDEX IN "TS_MQTINDEX"
NOT LOGGED INITIALLY;

COMMENT ON TABLE DB2INFO.MQT0000000041T02 IS 'AST created for
cube model MDSAMPLE.SalesModel';

REFRESH TABLE DB2INFO.MQT0000000041T02;

CREATE INDEX DB2INFO.IDX0000000041T02C ON DB2INFO.MQT0000000041T02("QUARTER")
CLUSTER;

REORG TABLE DB2INFO.MQT0000000041T02;

RUNSTATS ON TABLE DB2INFO.MQT0000000041T02 AND INDEXES ALL;

COMMIT;

]]>
</sql>
<refreshSql>
<![CDATA[
-- ******************************************************************************
```

```
-- * Script to create/refresh summary tables.
-- *
-- * Cube model schema: MDSAMPLE
-- * Cube model name:   SalesModel
-- * Diskspace limit:   75000
-- * Time limit:        300
-- * Sampling:          Yes
-- * Refresh type:      Refresh immediate
-- * Tablespace name:   TS_MQTTABLE
-- * Indexspace name:   TS_MQTINDEX
-- ********************************************************************

UPDATE COMMAND OPTIONS USING c OFF;

REFRESH TABLE DB2INFO.MQT0000000041T01;
REFRESH TABLE DB2INFO.MQT0000000041T02;
COMMIT;

]]>
</refreshSql>
</advise>
</olap:response>
```

Modification operations: Alter, Create, Drop, Import, and Rename

The DB2 Cube Views API includes five modification operations:

- Alter
- Create
- Drop
- Import
- Rename

When the stored procedure implements the modification operation, it also ensures that objects are complete and referentially valid.

Create

The Create operation creates metadata objects. It accepts one or more metadata object operands and creates these objects in DB2 Cube Views in the order that they are passed to the operation. Sequences of objects passed to this operation can include objects of different types.

Objects that are passed to this operation can optionally reference other objects. If references exist between objects, they must be reflected in the ordering of the objects. For example, if the object `MyObject` references the object `YourObject`, then `YourObject` must be passed to the operation before `MyObject`. For more information about how metadata objects can reference each other, see "Metadata object format" on page 155.

The Create operation validates each object. Errors are returned if the object that is created already exists, or if an object referenced by the object that is created does not already exist. If an input object specifies a schema that does not exist, the operation creates the schema if you have sufficient authority in the database.

Table 43. Create request parameters

Name	Type	Data type	Valid values	Description
object	operand	XML element	See "Operation operands" on page 147.	Objects that can be created.

Table 44. Create response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure	See section "Message structure" on page 147.	Message that indicates the status of the Create operation.

Alter

The Alter operation updates metadata object information. It accepts one or more metadata object operands and updates their object counterparts in the metadata catalog tables. Objects are updated in the order that they are passed to the operation. Sequences of objects that are passed to this operation can include objects of different types.

Objects that are passed to this operation can optionally reference other objects. If references exist between objects, they must be reflected in the ordering of the objects. For more information about how metadata objects can reference each other, see "Metadata object format" on page 155.

This operation cannot update the schema or the name of an object. Object names can be changed with the Rename operation.

The Alter operation validates each object. Errors are returned if the object that is updated does not exist.

Table 45. Alter request parameters

Name	Type	Data type	Valid values	Description
object	operand	XML element	See "Operation operands" on page 147.	Objects that can be updated.

Table 46. Alter response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure	See section "Message structure" on page 147.	Message that indicates the status of the Alter operation.

Rename

The Rename operation renames a single DB2 Cube Views metadata object that is identified by its current schema and name. Only the name of an object can be changed. The schema of an object cannot be changed. The Rename operation can rename objects even if they are currently referenced by other metadata objects.

Table 47. Rename request parameters

Name	Type	Data type	Valid values	Description
objectType	argument	XML string	cubeModel cube cubeFacts cubeDimension cubeHierachy cubeLevel dimension facts hierarchy attributeRelationship level join measure attribute	Type of DB2 Cube Views metadata objects that are renamed. You can specify one of the valid values for this parameter.
currentRef	operand	DB2 Cube Views metadata object reference	See "Operation operands" on page 147.	Current schema and name of the metadata object being renamed.
newRef	operand	DB2 Cube Views metadata object reference	See "Operation operands" on page 147.	New schema and name of the metadata object that is renamed.

Table 48. Rename response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure	See section "Message structure" on page 147.	Message that indicates the status of the Rename operation.

Drop

The Drop operation deletes metadata objects from DB2 Cube Views. This operation deletes one or more metadata objects depending on the *objectType* and *restriction* components that are specified. If the object that is dropped is currently referenced by another metadata object, an error is returned.

Table 49. Drop request parameters

Name	Type	Data type	Valid values	Description
objectType	argument	XML string	See "Operation parameters" on page 144.	Types of metadata object being deleted. You can specify one of the valid values for this parameter.
restriction [optional]	argument	XML string	See "Operation parameters" on page 144.	Limits the scope of a metadata deletion. This is analogous to predicates in SQL.

Table 50. Drop response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure	See section "Message structure" on page 147.	Message that indicates the status of the Drop operation.

Import

The Import operation creates metadata objects or reports the existence of metadata objects in the metadata catalog. This operation behaves similarly to the Create operation, except for the manner with which it deals with the presence of preexisting metadata objects.

You can define optional modes for the Import operation. These different modes determine what action to take when you try to import objects with names identical to objects already in the catalog.

Depending on the mode that you run, errors are returned if the object that is created already exists, or if an object that is referenced by the object that is created does not already exist. If an input object specifies a schema that does not exist, the Import operation creates the schema if you have sufficient authority for the database.

The Import operation validates each object.

See "Operation parameters" on page 144 for a detailed description of the various operation modes.

Table 51. Import request parameters

Name	Type	Data type	Valid values	Description

mode	argument	XML string	`create new - ignore collisions`
			`create new - replace collisions`
			`create new - abort on collision`
			`report new - report collisions`
				Defines the actions for new and existing objects that are imported.
				
				See the description for *mode* in "Operation parameters" on page 144.

object	operand	XML element	See "Operation operands" on page 147.	Objects being imported.

Table 52. Import response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure	See section "Message structure" on page 147.	Message that indicates the status of the Import operation.

Table 52. Import response parameters (continued)

Name	Type	Data type	Valid values	Description
newList	reference list	XML element	See the description for *mode* in "Operation parameters" on page 144.	List of name-schema pairs that reference the new objects.
collisionList	reference list	XML element	See the description for *mode* in "Operation parameters" on page 144.	List of name-schema pairs that reference objects that collide with other objects.

How the API handles functional dependencies for modification operations

The DB2 Cube Views metadata API manages the functional dependencies for a level object according to specific rules. Table 53 shows the action that the API takes for each level modification request.

Table 53. API action for functional dependencies

Request	Action
Create a level	The API creates a corresponding DB2 functional dependency if possible. If a functional dependency cannot be created, the API creates the level without a functional dependency and returns a warning message.
Drop a level	The API will drop the level and the associated DB2 Functional dependency. If the API encounters an error trying to drop the level or the functional dependency, the API returns an error message and the level object is not dropped.
Alter a level (without an Functional dependency)	If you do not create a functional dependency when you alter a level, no action is taken. If you do create a functional dependency when you alter a level, the API creates a corresponding DB2 functional dependency if possible. If a functional dependency cannot be created, the API alters the level without a functional dependency and returns a warning message.
Alter a level (with an Functional dependency)	If the level has a functional dependency and you want to drop the functional dependency when you alter the level, the API will alter the level and drop the associated DB2 functional dependency. If the API encounters an error trying to alter the level, the API returns an error message and the level object is not altered. If you want to alter the level and maintain the functional dependency, the API drops and re-creates the functional dependency. If a functional dependency cannot be dropped, the API returns an error message, but continues to alter the level and re-create the new functional dependency with a different name.
Import a level (Create new: ignore collisions mode)	The API creates a corresponding DB2 functional dependency if possible. If a functional dependency cannot be created, the API creates the level without a functional dependency and returns a warning message.
Import a level (Create new: replace collisions mode)	Same as Alter.

Administration operations: Validate and Translate

DB2 Cube Views includes two administration operations: Validate and Translate. You use the Translate operation only when you migrate from DB2 Cube Views, Version 8.1 to DB2 Cube Views, Version 8.2. The translate operation MAPS `Version 8.1 metadata XML to version 8.2 metadata XML.

Validate

The Validate operation checks the validity of one or more metadata objects. Validity is defined by whether an object conforms to the DB2 Cube Views object rules. For the objects that are validated by this operation, you specify the *objectType* argument and the *restriction* parameters. Use the *mode* parameter to specify the extent of the validation.

The Validate operation checks the following issues:

- Completeness of metadata object information
- Referential integrity between metadata objects
- Existence of referenced relational tables, views, aliases, and nickname columns.
- Correctness of SQL Expression stored in metadata objects, such as attributes and measures

The Validate operation stops when it finds an invalid metadata object. A message that describes the validation violation is returned by this operation when it finds a violation. The other operations (Create, Alter, and Import) also implicitly validate metadata objects. The translate operation does not validate the metadata objects.

Table 54. Validate request parameters

Name	Type	Data type	Valid values	Description
objectType	argument	XML string	See "Operation parameters" on page 144.	Types of DB2 Cube Views metadata object being validated. You can specify one of the valid values for this parameter.
restriction (optional)	argument [optional]	XML string	See "Operation parameters" on page 144.	Limits the scope of a metadata validation. This is analogous to predicates in SQL.

mode	argument	XML string	`base`
			`cubeModel`
			`  completeness`
			`optimization`
				Defines the extent of the validation actions to be performed.

Table 55. Validate response parameters

Name	Type	Data type	Valid values	Description
status	status message	DB2 Cube Views message structure	See section "Message structure" on page 147.	Message that indicates the status of the Validate operation.
info	message list	List of message structures	See section "Message structure" on page 147.	List of messages that describe the warnings and errors generated by the Validate operation.

Translate

The Translate operation checks that the incoming metadata XML is syntactically correct by validating the XML schema, but it does not validate column or other

references. You can use the Translate operation to convert arbitrary DB2 Cube Views metadata. The Translate operation requires a complete XML document. References to objects that are not in the XML document might create errors.

The Translate operation maps objects as shown in the following table. Each Version 8.1 object maps to an identical object in Version 8.2, unless otherwise noted.

Table 56. How Version 8.1 objects are mapped to Version 8.2 objects

Version 8.1 objects	Version 8.2 objects
Attribute	• Attributes are extended to have a new property *nullability.* • Existing attributes will have nullability of *unknown*.
Join	No changes.
Attribute relationship	• No longer referenced by hierarchies and cube hierarchies. • All attribute relationships become orphans. (No other objects refer to these attribute relationships.) When you export a cube model or cube from OLAP Center, you no longer see attribute relationships in the exported XML file. From OLAP Center, you must export all metadata to export attribute relationships.
Hierarchies + attributes + attribute relationships	• Exactly one level object is created for each attribute that is referenced by any Version 8.1 hierarchy. • The level object will use the same name (schema, name, and so on) as its Version 8.1 source attribute. • The level's default attribute will be the source attribute. • The level's key consists of the source attribute plus ancestor attributes above the source attribute based on one of the Version 8.1 hierarchies. • If the source attribute was referenced in more than one Version 8.1 hierarchy, the hierarchy is chosen with the fewest ancestors (in cases of a tie, one is arbitrarily chosen) and the ancestor attributes are added from top down to the source as the key attributes in the Version 8.2 level. • All Version 8.1 attribute relationships referenced by hierarchies that have left attribute equal to the source attribute will be used to create the related attributes in the level. For each matching (on the left) attribute relationship, the right side attribute is added to the level as a related attribute. When the related attribute list is created, duplicates are removed.
Cube hierarchies + attributes + attribute relationships	• A unique cube level object is created for each attribute referenced by each Version 8.1 cube hierarchy. • The cube level is named based on the attribute name and the cube name. If the cube is named Sales with an Attribute named Region, the cube level will be named Region (Sales). • Each cube level refers to its corresponding level object. For example, Region (Sales) points to the Region level. • The cube level's related attributes are based on the attribute relationship objects in the given cube hierarchy. For each attribute relationship that has the left attribute equal to the source attribute (in the parent Level), the right side attribute will be added to the cube level as a related attribute.
Hierarchy	Each Version 8.1 hierarchy maps to a Version 8.2 hierarchy with an ordered set of levels (no attributes and no attribute relationships).
Cube hierarchy	Each Version 8.1 cube hierarchy maps to a Version 8.2 cube hierarchy with an ordered set of cube levels (no attributes and no attribute relationships).
Dimension	Associated levels are added.
Cube dimension	No changes.

Table 56. How Version 8.1 objects are mapped to Version 8.2 objects (continued)

Version 8.1 objects	Version 8.2 objects
Measure	Like attributes, measures are extended to have a new property *nullability*. Existing measures will have *nullability* of unknown.
Facts	No changes.
Cube facts	No changes.
Cube model	No changes.
Cube	No changes during translation, but in Version 8.2, a cube can include usage information.

Sample input and output parameters in metadata operations

The following samples show how you can structure parameters in the three types of metadata operations. In these examples, portions of the XML structures are excluded, but they are represented with an ellipsis (...).

Retrieval operation

The following samples show how a retrieval operation called describe is structured. See "Retrieval operation: Describe" on page 128 for more information about the describe operation. In the following example, the **metadata** parameter is empty on input, but populated on output.

Request and metadata

```
<olap:request xmlns:olap="http://www.ibm.com/olap" ... >
   <describe objectType="cube" recurse="no">
      <restriction>
         <predicate property="schema" operator="=" value ="myschema"/>
      </restriction>
   </describe>
</olap:request>

<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... />
```

Response and metadata

```
<olap:response xmlns:olap="http://www.ibm.com/olap" ... >
   <describe>
      <status id="0" text="Operation completed
       successfully."type="informational"/>
   </describe>
</olap:response>

<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... >
   <cube name="cube1" schema="myschema" ... > ... </cube>
      ...
   <cube name="cubeN" schema="myschema" ... > ... </cube>"
</olap:metadata>
```

Modification operations

The following samples show how a modification operation called create is structured. See "Modification operations: Alter, Create, Drop, Import, and Rename" on page 135 for more information about the create operation and other modification operations. The **metadata** parameter is populated on input but empty on output.

Request and metadata

```
<olap:request xmlns:olap="http://www.ibm.com/olap" ... >
   <create/>
</olap:request>

<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... >
   <attribute name="LocationID" ... > ... </attribute>
   <attribute name="Country" ... > ... </attribute>
   <attribute name="State" ... > ... </attribute>
   <attribute name="City" ... > ... </attribute>
   <dimension name="Location" ... type="regular">
      <attributeRef name="LocationID" ... </attributeRef>
      <attributeRef name="Country" ... </attributeRef>
      <attributeRef name="State" ... </attributeRef>
      <attributeRef name="City" ... </attributeRef>
        ...
   </dimension>
</olap:metadata>
```

Response and metadata

```
<olap:response xmlns:olap="http://www.ibm.com/olap" ... >
   <create>
      <status id="0" text="Operation completed
       successfully."type="informational"/>
   </create>
</olap:response>

b<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... >
```

Administration operation

The following examples show how an administration operation called validate is structured. See "Administration operations: Validate and Translate" on page 140 for more information about the validate operation.

Request and metadata

```
<olap:request xmlns:olap="http://www.ibm.com/olap" ... >
   <validate objectType="cube" mode="base">
      <restriction>
         <predicate property="schema" operator="=" value ="myschema"/>
      </restriction>
   </describe>
</olap:request>

<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... />
```

Response and metadata

```
<olap:response xmlns:olap="http://www.ibm.com/olap" ... >
   <validate>
   <status id="1" text="...Additional information
     returned."type="informational"/>
   <info><message id="6299" text="At least one
     database view was found during validation."
     type="warning"/></info>
   </validate>
</olap:response>

<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... >
```

Additional operation XML sample files that can be used with the db2mdapiclient utility are in the SQLLIB\samples\olap\xml\inputdirectory.

Operation parameters

Various parameters are available for each metadata operation. These parameters tailor the behavior of an operation to its particular application.

DB2 Cube Views provides five parameters for the metadata operations:

- objectType
- recurse
- restriction
- mode (for Import operation)
- mode (for Validate operation)

objectType parameter

This parameter specifies the type of metadata objects that are involved in the requested operation. The following object types correspond directly to the DB2 Cube Views metadata object model.

- all
- cubeModel
- cube
- cubeDimension
- cubeFacts
- cubeHierarchy
- cubeLevel
- dimension
- facts
- hierarchy
- attributeRelationship
- level
- join
- measure
- attribute

recurse parameter

This parameter controls whether an operation performs recursively. In a non-recursive mode, an operation performs its actions on only the metadata objects that directly match the objectType parameter and **restriction** parameter specifications. An operation starts in the non-recursive mode set of metadata objects and additionally performs its actions on all other metadata objects as it goes through the object-association paths starting from the non-recursive mode set of objects. The **recurse** parameter includes the *yes* option and the *no* option.

For example, a non-recursive operation might return a list of dimensions; whereas a recursive operation might return a list of dimensions, in addition to all other objects (of different types) that are referenced by those dimensions and objects that are referenced by those dimension objects.

restriction parameter

This parameter specifies that a metadata operation will be restricted or limited in scope. This parameter is similar to predicates in an SQL query. Restrictions are expressed in XML by using the <restriction> and <predicate> tags that are defined by the DB2 Cube Views XML schema.

Restrictions can be based on the object properties that are common to all of the metadata objects and on the relationships between metadata objects.

Property-based predicates contain the following attributes:

property
: Associated with a predicate tag and must specify either a *name* attribute or a *schema* attribute.

operator
: Associated with a predicate tag and must specify the equal sign (=).

value
: Associated with a predicate tag and is the string representation of the value to be compared to the property that is specified by the *property* attribute.

See "Sequence of the operation steps" on page 148 for a description about how the **restriction** parameter relates to the overall sequencing of operation steps.

This example restricts the scope of an operation to the objects in the ABC schema:

```
<restriction>
       <predicate property="schema" operator="=" value="ABC">
</restriction>
```

mode (for Import) parameter

This parameter sets the mode for the Import operation. The following table describes the available modes.

A collision occurs when an object that is passed to the Import operation as input already exists in the metadata catalog.

Table 57. Import modes

Mode	Description	Returned reference lists
Create new: ignore collisions	• Input objects that do not collide are created. • Input objects that collide are not created. • Preexisting objects are not altered. • Errors are not generated by collisions.	**newList** Contains the name-schema pairs for the objects that were successfully created. **collisionList** Contains the name-schema pairs for the objects in a collision that were ignored and were not created.

Table 57. Import modes (continued)

Mode	Description	Returned reference lists
Create new: replace collisions	• Input objects that do not collide are created. • Input objects that collide replace preexisting objects. • Preexisting objects are replaced by the input objects. • Errors are not generated by collisions.	**newList** Contains the name-schema pairs for the objects that were successfully created. **collisionList** Contains the name-schema pairs for the objects in a collision that were replaced.
Create new: abort on collision	• Input objects are created only if no collisions exist for the entire operation. • In the case of a collision, no objects are created as part of the operation. • Preexisting objects are not altered. • Errors are generated by collisions.	**newList** Contains the name-schema pairs for the objects that were successfully created or the noncolliding objects that were not created. **collisionList** Contains the name-schema pairs for the objects in a collision that were not created.
Report new: report collisions	• No objects are created. • Reports on the collision status of the input objects. • Preexisting objects are not altered. • Errors are not generated by collisions.	**newList** Contains the name-schema pairs for the objects not in a collision were and not created. **collisionList** Contains the name-schema pairs for the objects in a collision that were not created.

Tags of the newList and collisionList reference lists must adhere to a predefined ordering. The following list shows the ordering of reference types for the IMPORT operation:

1. <attributeRef>
2. <joinRef>
3. <attributeRelationshipRef>
4. <levelRef>
5. <cubeLevelRef>
6. <hierarchyRef>
7. <cubeHierarchyRef>
8. <dimensionRef>
9. <cubeDimensionRef>
10. <measureRef>
11. <factsRef>
12. <cubeFactsRef>
13. <cubeModelRef>
14. <cubeRef>

mode (for Validate) parameter

This parameter sets the mode for the Validate operation. The following table describes the available modes for the Validate operation. The rules in the following table refer to categories of the DB2 Cube Views object rules.

Table 58. Validate modes

Mode	Description
base	Checks conformance to the base rules
cubeModel completeness	• Checks conformance to the cube model completeness rules • Checks conformance to the base rules
optimization	• Checks conformance to the optimization rules • Checks conformance to the cube model completeness Rules • Checks conformance to the base rules

Operation operands

When an operation requires metadata objects or their references to accompany the request, these objects or references are called the ″operands″ of the operation.

The operands that are passed to metadata operations by using either the **request** or **metadata** parameters are:

object This operand contains the metadata objects that are acted on. The format used to represent metadata objects is described in “Metadata object format” on page 155.

currentRef
This operand is for the Rename operation, and it contains metadata object's schema and name.

newRef
Similar to the *currentRef* operand, this operand is used for the Rename operation, and it contains the metadata object's schema and name.

Message structure

The DB2 Cube Views API returns informational, warning, and error messages that have a particular structure.

The following table describes the components of a DB2 Cube Views message.

Table 59. Message component

Component	Description
id	A unique integer identifier for the message.
type	A message can be one of three types: • informational • warning • error
text	The character string that contains the text of the message.

Table 59. Message component (continued)

Component	Description
tokens	The values that are substituted into the text string for the message. A message can include any number of tokens. The following XML elements can appear as tokens in a message: • attributeRef • joinRef • attributeRelationshipRef • levelRef • cubeLevelRef • hierarchyRef • cubeHierarchyRef • dimensionRef • cubeDimensionRef • measureRef • factsRef • cubeFactsRef • cubeModelRef • cubeRef • column • text

Here is an example of a message with no tokens:

```
<status id="0" text="Operation completed successfully."type="informational"/>
```

Here is an example of a message with tokens:

```
<status id="6331" text="The left attribute for
the &quot;MDOBJ_ID_ATTRIBUTERELATIONSHIP.MDSAMPLE.State_PopGroup&quot;
attribute relationship is not a part of the
&quot;MDOBJ_HIERARCHY.MDSAMPLE.RegionState&quot; hierarchy."
type="error">
<tokens>
<attributeRelationshipRef name="State_PopGroup" schema="MDSAMPLE"/>
<text value="MDOBJ_HIERARCHY.MDSAMPLE.RegionState"/>
</tokens>
</status><
```

Sequence of the operation steps

Only three of the operation arguments determine the scope of an operation.

The three arguments are listed here in the order that they are applied to an operation:

1. objectType
2. restriction
3. recurse

The following example shows how some objects might be returned that apparently do not match the restriction that you intended, but were returned as part of the recurse phase of the operation.

Example: Recursively describe the cubes that belong to the schema `myschema`:

Operation arguments:

```
objectType = "cube"
restriction =  <restriction>
          <predicate property="schema" operator="=" value="myschema"/>
                 </restriction>
recurse = "yes"
```

The Describe operation begins by limiting its scope to cube objects. Of these cube objects, only the ones that belong to the `myschema` schema are selected. For each of these selected cube objects, the objects that they refer to are selected, and the objects are different types and potentially different schemas. All of the selected objects are then returned as part of a response to the cube request.

Logging and tracing

Run-time tracing for the DB2 Cube Views API

The API supports three priorities of tracing. Using the configuration file, an administrator can set the level of tracing to be logged. Run-time tracing is turned off by default. The default trace file name is `db2mdtrace.log`.

Typically, you do not need to run tracing. Tracing might be required if an error occurs in the API and IBM Software Support asks you to provide a trace file.

The following table describes the different tracing levels.

Table 60. Tracing levels

Level	Description	Examples
None	• Tracing is off	Not applicable
High	• Tracks only the external and internal API entry and exit points • Tracks the flow between components • Can include function arguments	• Begins and ends parsing • Begins and ends Create, Describe, Drop, and so on
Medium	• Tracks the flow of control between complex functions in the external and internal API • Tracks the flow between components • Includes high-level trace points	• Shows complex function calls made by the Create operation
Low	• Tracks simple or atomic functions in the internal API • Includes high-level and medium-level trace points • Use this level for most trace points	• Shows calls to the get or set methods for the metadata objects

When tracing is turned on with the level set to a value other than `None`, errors that occur in the API might be recorded in both the error log and the trace log, depending on the level and severity setting for these logs.

Log files for the DB2 Cube Views API

The API log files are generated at a DB2 instance level. The name of the error log file is `db2mderror.log`, and the name of the trace log file is `db2mdtrace.log`.

For a given DB2 instance the runs the DB2 Cube Views API, the log files for the API will be generated in the DB2 diagnostic data directory, also know as DB2DIAG. This DB2 diagnostic directory is generally in the following directories:

On Windows

DB2 instance path directory, such as `c:\sqllib\<myinst>`

On AIX

DB2 instance path/db2dump, such as ~my inst `/sqllib/db2dump`

You can change the default DB2DIAG path by using the `DB2DIAG db2 dbm cfg` setting.

The db2idrop utility cleans up the log files associated with a DB2 instance. If the default for DB2DIAG is not used, then the db2idrop utility cannot clean up the log files for the DB2 Cube Views API. The log files that cannot be cleaned up by the db2idrop utility must be manually cleaned up. Errors that occur when the DB2 Cube Views API loads the configuration file are logged in the `db2mdapi.log` file. The `db2mdapi.log file` is in the DB2DIAG path similar to the other API logs.

Error logging

The API distinguishes between three severities of errors. The default severity setting is medium, and the default error log file name is mderror.log. When an error occurs while reading the configuration file, this error is logged in a file named db2mdapi.log.

The following table describes the error severity levels.

Table 61. Error severity levels

Severity	Description	Examples
None	• Ignore all errors and warnings	Not applicable
High (Most errors have this severity)	• Records only critical, unrecoverable errors • Dumps a callstack to the log	• Internal coding error
Medium	• Records user-recoverable errors • Logs high-severity errors also • Dumps a callstack to the log	• End-user mistakes, such as attempts to create a duplicate object • Metadata validation errors • Out of memory. You can increase memory or reduce usage.
Low	• Records warning situations • Logs high and medium severity errors also • Low severity errors do not dump a callstack	• Warning of internal error • Informational messages

When the API is configured to high or medium error logging, and a high or medium error occurs, the API generates a callstack beginning at the point where the error occurs in the API. This callstack is similar to a medium-level trace, but the data is sent to the error log instead of the trace log.

Logging and tracing scenarios

The following logging and tracing scenarios demonstrate how errors can be captured.

Scenario 1 (Trace level: medium; Error severity: high): When a high-severity error occurs, it appears in both the error and the trace logs.

```
<log>
   <trace level="medium" logFile="db2mdtrace.log" bufferSize="0" />
   <error level="high" logFile="db2mderror.log" bufferSize="0" />
  </log>
```

Scenario 2 (Trace level: medium; Error severity: low): When a low-severity error occurs, it appears in only the error log because the trace log allows only entries of level medium or high.

```
<log>
   <trace level="medium" logFile="db2mdtrace.log" bufferSize="0" />
   <error level="low" logFile="db2mderror.log" bufferSize="0" />
  </log>
```

Errors related to missing environment variables or to failures accessing log files are returned via the SQLSTATE of the stored procedure call to the database client application. When an error occurs processing the configuration file, this error is logged in the db2mdapi.log file. If an error occurs opening any of the user-specified log files, no error is captured.

Code page support

DB2 Cube Views uses two code pages: the DB2 client code page (application code page) and the DB2 database code page. See the DB2 *Administration Guide: Planning*, ″Supported territory codes and code pages″ for information about how to determine the DB2 client code page. The DB2 Cube Views API stored procedure runs in the DB2 database code page. The DB2 database code page is set when the database is created. The DB2 client code page and DB2 database code page can be different. CLI will convert the stored procedure character large object (CLOB) parameters from client code page to database code page for the stored procedure.

The following illustration shows how the client communicates with the server through a call-level interface (CLI). The CLI converts client code pages to the database code page.

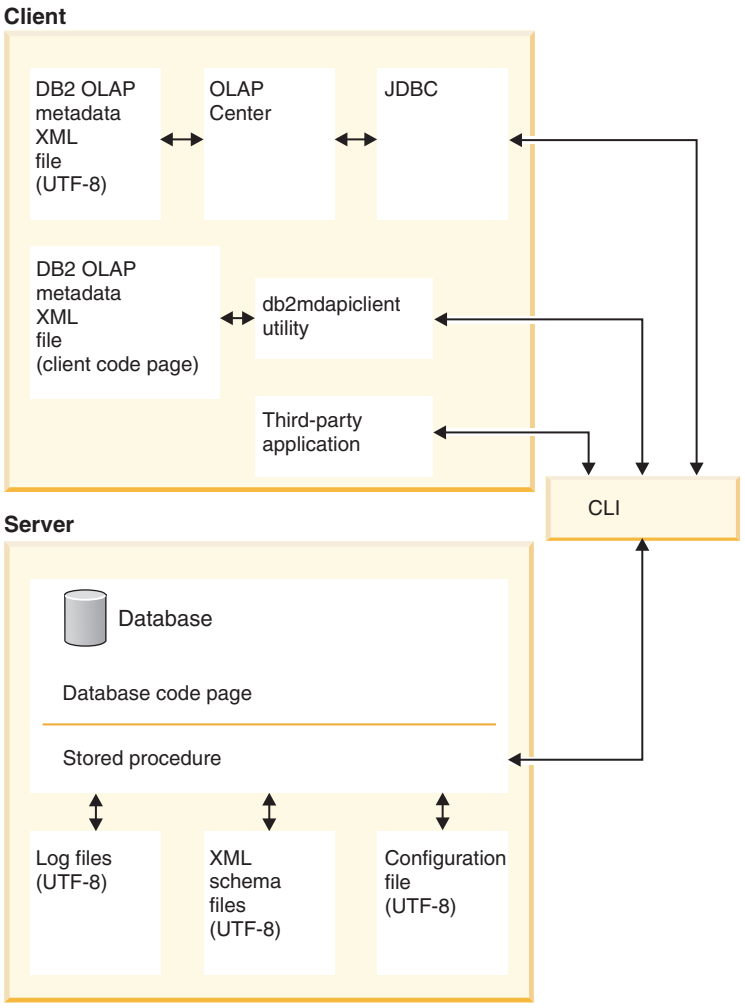

Figure 41. How data flows from different clients that use code pages or UTF-8 format through the DB2 CLI then to the database server

CLI manages the conversion between application code page and database code page. Data that is sent from the DB2 Cube Views client to the API is considered input. Data that is sent from the API to the DB2 Cube Views client is considered output. Input and output data is encoded in the DB2 client code page.

The components of DB2 Cube Views have the following code page specifications. The OLAP Center:

- Accepts and generates only DB2 Cube Views XML files that are encoded in UTF-8
- Returns an error if an input DB2 Cube Views XML file specifies an encoding other than UTF-8
- Interprets the lack of an encoding specification in a file as meaning that the file is encoded in UTF-8
- With the Export function creates DB2 Cube Views XML files with an explicit encoding specification of UTF-8

The db2mdapiclient:

- Interprets input DB2 Cube Views XML files as being encoded in UTF-8 unless the -l option is specified, and therefore ignores explicit encoding specifications listed within the files.

- Generates DB2 Cube Views XML files encoded in UTF-8 unless the -l option is specifie, and does not include explicit encoding specifications within these files.

The stored procedure API:

- Interprets CLOB parameters as being encoded in the DB2 client code page
- Ignores explicit encoding specifications in input DB2 Cube Views XML files
- Generates DB2 Cube Views XML files with no explicit encoding specifications
- Processes input and output XML files using the DB2 database code page
- Generates API log files that are encoded by using UTF-8 including any embedded DB2 messages. API log files are not in XML.
- Does not create log files explicitly stating an encoding of UTF-8
- Encodes in UTF-8 the XML schema files that are used by the API
- Encodes in UTF-8 the XML API configuration file

For third-party applications, other applications that directly call the DB2 Cube Views API will have to pass and accept as parameters XML files that are encoded in the DB2 client code page.

DB2 Cube Views metadata tables and XML schema files

Certain functions in the API are not supported if the version number is not correct.

Versions of metadata tables

The API only works if it connects to a DB2 database with a current set of DB2 Cube Views metadata tables. The current version for DB2 Cube Views Version 8.2 is 8.2.0.1.0. The version number for DB2 Cube Views Version 8.1 is 8.1.2.0. The DB2 Cube Views metadata catalog table version number is stored in SYSINFOVERSION table.

See “Overview of the db2mdapiclient utility” on page 157 for more information about metadata table versions.

The DB2 Cube Views XML schema files, or XSD files, are used for the DB2 Cube Views API. The XML schema files are used by the API DB2INFO.MD_MESSAGE stored procedure.

Versions of XML schema files

All of the XML documents that are passed to and from the metadata API must have a version number. This version number allows the stored procedure to know what XML schema the client uses. The XSD schema files specify which version numbers are acceptable for particular operations.

The current version for DB2 Cube Views Version 8.2 is 8.2.0.1.0.

The following schema files contain information about the stored procedure md_message().

db2md_parameter.xsd file

This file contains information about request and response operations. This file works with version 8.1.2.1.0 and version 8.2.0.1.0.

db2md_metadata.xsd file

This file includes the db2md_types.xsd file and specifies only metadata elements. This file does not contain version information.

db2md_types.xsd file

This file contains information about all metadata objects. This file works with version 8.1.2.1.0 and version 8.2.0.1.0.

The API can support the previous version of schema files, version 8.1.2.1.0, but supports only the Describe and Translate operations. The API returns an error if the requested version 8.1.2.1.0 operation is anything other than Describe or Translate.

DB2 Cube Views configuration file

The API can be configured at the level of a DB2 instance. You can change the parameters of a configuration file that is called `db2md_config.xml`.

Every installation of DB2 Cube Views has a default configuration file in the db2_installation_path/cfg directory. For example, on Windows, the default configuration file might be in the c:\sqllib\cfg directory, and on AIX, the default configuration file might be in the /usr/opt/db2_08_01/cfg directory.

Every DB2 instance that runs DB2 Cube Views has a physical copy of the db2md_config.xml file in the db2_instance_path directory. For example, on Windows, the physical copy might be in the c:\sqllib\my_inst directory, and on AIX, the physical copy might be in the ~my_inst/sqllib directory.

The db2icrt utility copies the default configuration file to the db2_instance_path directory, and it creates a new instance. For DB2 instances that were created before DB2 Cube Views was installed, you can manually copy the configuration file into the instance directory if the installation program did not successfully copy the file. If the API cannot find the configuration file in the instance directory, the API will try to copy the default configuration file to the instance directory.

A configuration file, db2md_config.xml, is used to set error logging and run-time tracing. By modifying the contents of the configuration file, an administrator can specify the level of tracing, the severity of errors to log, and the buffer size (in bytes) to use when logging.

The content structure of the db2md_config.xml configuration file is defined by the db2md_config.xsd XML schema file. The following example shows the contents of the configuration file.

```
<olap:config xmlns:olap="http://www.ibm.com/olap"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xsi:schemaLocation="http://www.ibm.com/olap db2md_config.xsd">
      <log>
           <trace level="none" logFile="db2mdtrace.log" bufferSize="0"/>
           <error level="medium" logFile="db2mderror.log" bufferSize="0"/>
      </log>
</olap:config>
```

Metadata object format

The DB2 Cube Views XML schema defines the base XML elements that map directly to the objects in the DB2 Cube Views metadata object model. Complex metadata structures are then represented as sequences of these base elements. Associations between objects within complex metadata structures are captured through name references between base elements.

An example of a name reference is the way in which a cube element can contain a reference to a dimension element. The following example shows the type of data provided for a cube object as defined by the XML schema follows. In this example, only text descriptions are shown; in application, XML representations of information are used.

```
cube
->cube model reference
->cube dimension references
->cube facts reference
->view
```

In the case of a cube object, the references to the other types of objects are all contained within the base element representing the cube. With non-recursive retrieval operations, sequences of cube objects (and only cube objects) are presented. With recursive retrieval operations, information on cube objects is presented, in addition to information on any other object (of different type) referenced by the identified cubes.

The ordering of objects is defined by the DB2 Cube Views XML schema. Within the scope of a single operation, objects of the same type (such as., cube objects) are grouped together. Within these groups, the order of elements is influenced by the references between objects of the same type. Referenced objects must appear before referencing objects. The ordering between these groups is as follows:

1. attribute
2. join
3. attributeRelationship
4. level
5. cubeLevel
6. hierarchy
7. cubeHierarchy
8. dimension
9. cubeDimension
10. measure
11. facts
12. cubeFacts
13. cubeModel
14. cube

The order of the object-type groups is independent of the associations between objects. The fact that attributes and joins play different roles when associated with different object types does not affect their order within the Association format.

For an example XML file that shows the type and order of metadata information, see the XML metadata file included with the CVSAMPLE.

Chapter 8. Sample files

Overview of the db2mdapiclient utility

The db2mdapiclient utility is provided as sample source code for coding an application for DB2 Cube Views Multidimensional Services. You can use the utility to import, export, and optimize metadata objects.

Note: The utility shipped with DB2 Cube Views Version 8.2 has been enhanced and does not match exactly the sample source code in the sqllib\samples\olap\client\db2mdapiclient.cpp file.

You can use the db2mdapiclient utility to perform any of the operations that are supported by the DB2 Cube Views stored procedure, MD_MESSAGE(), which are described in the following table:

Table 62. Operations. Multidimensional Services operations that the db2mdapiclient utility can perform

Task	Operation
Export metadata objects to a file	DESCRIBE
Import metadata objects from a file	CREATE or IMPORT
Change existing metadata objects	ALTER or RENAME
Delete existing metadata objects	DROP
Verify that the existing metadata objects are valid	VALIDATE
Optimize a cube model	ADVISE
Migrate XML metadata from Version 8.1 to Version 8.2	TRANSLATE

The db2mdapiclient utility uses files to hold the XML that is passed to and received from the MD_MESSAGE() stored procedure.

For importing, the db2mdapiclient utility typically uses an XML file that is produced by a DB2 Cube Views bridge or that was exported from the OLAP Center. For exporting, the db2mdapiclient utility produces an XML file that a DB2 Cube Views bridge utility can use to add metadata to a database or OLAP tool. The character encoding used for the input and output XML files is important. For more information on character encoding, see “Code page support” on page 151.

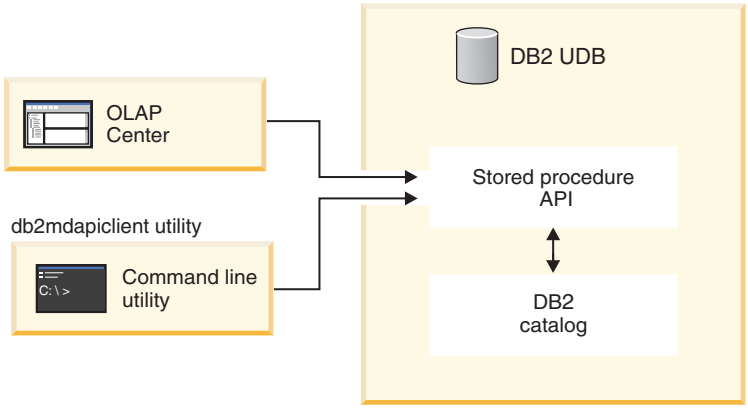

Figure 42. Metadata transfer. The db2mdapiclient utility and the OLAP Center transfer metadata through Multidimensional Services

The db2mdapiclient command: manipulating metadata objects

You can use the db2mdapiclient utility from the command line.

To see a list of parameters for the db2mdapiclient command, you can enter `db2mdapiclient` at a command line. The syntax for the **db2mdapiclient** command is:

```
db2mdapiclient -d dbname [-u user] [-p password] -i request.xml -o response.xml
    [-m inputmetadata.xml] [-n outputmetadata.xml] [-a parameter_buffer_size]
    [-b metadata_buffer_size] [-v] [-l] [-h]
```

`-d dbname` specifies the name of the database.

`-u user` specifies the user ID to connect to the specified database.

`-p password` specifies the password to connect to the specified database.

`-i request.xml` specifies the required input file that contains the operation to perform.

`-o response.xml` specifies the required output file that contains the response XML from the MD_MESSAGE() stored procedure. The third argument in the MD_MESSAGE() stored procedure returns this response XML.

`-m inputmetadata.xml` is the input file that contains the DB2 Cube Views metadata object XML. This option is required for Create or Import operations.

`-n outputmetadata.xml` is the optional output file that contains the response metadata object XML, if applicable, from the second argument of the MD_MESSAGE() stored procedure.

`-a parameter_buffer_size` specifies the buffer size for the parameters. The default value is 1048576 bytes.

`-b metadata_buffer_size` specifies the buffer size for the metadata object information. The default value is 1048576 bytes.

`-v` specifies that you want extra information printed while the command is processing.

`-l` disables UTF-8 support and specifies that the input and output files are in the local code page.

`-h` displays usage information.

The following diagram shows how the MD_MESSAGE() stored procedure is associated with the two input and two output files:

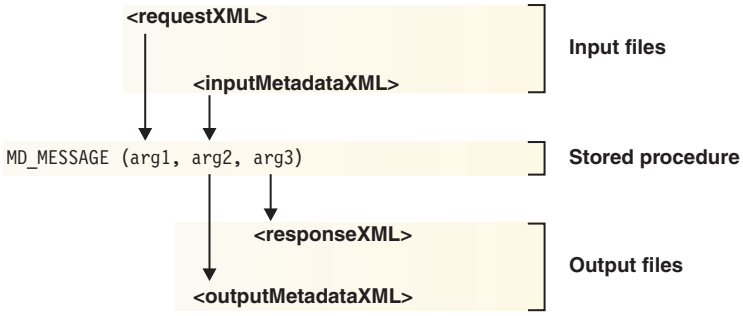

Figure 43. How the stored procedure handles the two input and output files from the db2mdapiclient utility

For example, to import DB2 Cube Views metadata objects for the CVSAMPLE database for Windows, change to the ..\SQLLIB\samples\olap\xml\input directory and enter the following command:

```
db2mdapiclient -d CVSAMPLE -u db2admin -p mypasswrd -i create.xml
  -o myresponse.xml -m ..\..\CVSAMPLE\CVSampleMetadata.xml.xml
```

For a description of the sample files that are provided, see "API sample files" on page 160. For more information on the Multidimensional Services operations, see the "DB2 Cube Views API overview" on page 123.

Sample database files

All of the following files, that are related to the CVSAMPLE database, are located in the \SQLLIB\samples\olap\cvsample\ directory.

CVSampleMetadata.xml
: An XML file that contains the CVSAMPLE metadata. Use this file to import the CVSAMPLE metadata with the OLAP Center and the db2mdapiclient utility.

CVSampleTables.sql
: An SQL script that you use to populate the CVSAMPLE tables.

FAMILY.txt, LINE.txt, LOCATION.txt, PRODUCT.txt, SALESFACT.txt, STORE.txt, TIME.txt
: A set of text files that contain the CVSAMPLE table data.

CVSampleExplain.sql
: An SQL script that you can use to determine if DB2 is rerouting a query to a summary table.

The \SQLLIB\samples\olap\xml\input directory also contains files that relate to the CVSAMPLE database.

Create.xml
: An XML file with the CREATE operation. Use this file to load the sample with the db2mdapiclient utility.

API sample files

Sample API files for the CVSAMPLE database are provided with DB2 Cube Views. You can use the sample files to perform sample scenarios with the db2mdapiclient utility. The db2mdapiclient utility is a thin wrapper to the Multidimensional Services stored procedure interface. The utility is provided as sample source code to show how to code an application against the API. The source code is located in \SQLLIB\samples\olap\client\db2mdapiclient.cpp.

You pass the contents of the sample files listed for each scenario as parameters to the MD_MESSAGE() stored procedure. Sometimes, the metadata parameter to the stored procedure is ignored on input, or not returned on output, which is designated in the following scenarios as <empty>. When you do not need input metadata do not specify the `-m` option in the db2mdapicleint utility command. When you do not need output metadata do not specify the `-n` option in the db2mdapicleint utility command.

For more information about using the MD_MESSAGE() stored procedure with the db2mdapiclient utility, see "Overview of the db2mdapiclient utility" on page 157. For more information about using the MD_MESSAGE() stored procedure by itself, see "DB2 Cube Views stored procedure" on page 125.

All of the sample files are located in the `\SQLLIB\samples\olap\xml\` directory. You can use the sample files to perform the following sample scenarios:

DROP

Use these files to drop all of the metadata objects in the metadata catalog. This sample assumes that the metadata catalog is not empty. If the metadata catalogs are empty, you receive a warning message that no objects are found for the operation.

Files that contain input parameters

- **Request:** `input\Drop.xml`
- **Metadata:** `<empty>`

  Do not specify the `-m` option in the db2mdapiclient utility.

Files that contain output parameters

- **Response:** `output\Drop_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

CREATE

Use these files to create metadata objects in the metadata catalog. This sample assumes that the metadata catalog is empty.

Files that contain input parameters

- **Request:** `input\Create.xml`
- **Metadata:** `input\CVSampleTestMetadata.xml`

Files that contain output parameters

- **Response:** `output\Create_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

DESCRIBE

Use these files to describe all of the metadata objects in the metadata catalog.

Files that contain input parameters

- **Request:** `input\Describe.xml`
- **Metadata:** `<empty>`

  Do not specify the `-m` option in the db2mdapiclient utility.

Files that contain output parameters

- **Response:** `output\Describe_response.xml`
- **Metadata:** `<metadata XML file>`

DESCRIBE (Restricted)

Use these files to recursively describe the CVSample.Daily Sales cube. This sample assumes that you have previously imported the metadata CVSampleMetadata.xml for CVSAMPLE.

Files that contain input parameters

- **Request:** `input\CVSampleDescribe_restricted.xml`
- **Metadata:** `<empty>`

  Do not specify the `-m` option in the db2mdapiclient utility.

Files that contain output parameters

- **Response:** `output\DescribeRestricted_response.xml`
- **Metadata:** `<metadata XML file>`

ALTER

Use these files to alter metadata objects in the metadata catalog. This sample assumes that you have previously imported the metadata CVSampleMetadata.xml for CVSAMPLE.

Files that contain input parameters

- **Request:** `input\Alter.xml`
- **Metadata:** `input\CVSampleAlter.xml`

Files that contain output parameters

- **Response:** `output\Alter_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

RENAME

Use these files to rename the CVSAMPLE.Sales Model cube model. The cube model is renamed to CVSAMPLE.Sales Model (2004). This sample assumes that you have previously imported the metadata CVSampleMetadata.xml for CVSAMPLE.

Files that contain input parameters

- **Request:** `input\CVSampleRename.xml`
- **Metadata:** `<empty>`

  Do not specify the `-m` option in the db2mdapiclient utility.

Files that contain output parameters

- **Response:** `output\Rename_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

VALIDATE

Use these files to validate all of the metadata objects in the metadata catalog using an optimization validation mode.

Files that contain input parameters

- **Request:** `input\Validate.xml`
- **Metadata:** `<empty>`

  Do not specify the `-m` option in the db2mdapiclient utility.

Files that contain output parameters

- **Response:** `output\Validate_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

VALIDATE (Restricted)

Use these files to validate the CVSAMPLE.Daily Sales cube using an optimization validation mode. This sample assumes that you have previously imported the metadata CVSampleMetadata.xml for CVSAMPLE.

Files that contain input parameters

- **Request:** `input\CVSampleValidate_restricted.xml`
- **Metadata:** `<empty>`

  Do not specify the `-m` option in the db2mdapiclient utility.

Files that contain output parameters

- **Response:** `output\ValidateRestricted_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

IMPORT with the *create new - ignore collisions* mode

Use these files to import metadata objects into the metadata catalog using the *create new - ignore collisions* import mode. This sample assumes that the metadata catalog is empty.

Files that contain input parameters

- **Request:** `input\Import_mode1.xml`
- **Metadata:** `CVSampleTestMetadata.xml`

Files that contain output parameters

- **Response:** `output\Import_mode1_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

IMPORT with the *create new - replace collisions* mode

Use these files to import metadata objects into the metadata catalog using the *create new - replace collisions* import mode. This sample assumes that you have already completed the IMPORT with the *create new - ignore collisions* mode scenario.

Files that contain input parameters

- **Request:** `input\Import_mode2.xml`
- **Metadata:** `CVSampleTestMetadata.xml`

Files that contain output parameters

- **Response:** `output\Import_mode2_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

IMPORT with the *create new - abort on collision* mode

Use these files to import metadata objects into the metadata catalog using the *create new - abort on collision* import mode. This sample assumes that you have already completed the IMPORT with the *create new - replace collisions* mode scenario.

Files that contain input parameters

- **Request:** `input\Import_mode3.xml`
- **Metadata:** `CVSampleTestMetadata.xml`

Files that contain output parameters

- **Response:** `output\Import_mode3_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

IMPORT with the *create new - report collisions* mode

Use these files to import metadata objects into the metadata catalog using the *create new - report collisions* import mode. This sample assumes that you have already completed the IMPORT with the *create new - abort on collision* mode scenario.

Files that contain input parameters

- **Request:** `input\Import_mode4.xml`
- **Metadata:** `CVSampleTestMetadata.xml`

Files that contain output parameters

- **Response:** `output\Import_mode4_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

TRANSLATE

Use these files to translate Cube Views Version 8.1 metadata to Cube Views Version 8.2 metadata.

Files that contain input parameters

- **Request:** `input\Translate.xml`
- **Metadata:** `CVSampleTranslateMetadata.xml`

Files that contain output parameters

- **Response:** `output\Translate_response.xml`
- **Metadata:** `<metadata XML file>`

ADVISE

Use these files to run the Optimization Advisor for the cube model CVSAMPLE.Sales Model. This sample assumes that you have previously imported the metadata CVSampleMetadata.xml for CVSAMPLE, and have not run the previous Rename scenario.

Files that contain input parameters

- **Request:** `input\CVSampleAdvise.xml`
- **Metadata:** `<empty>`

Do not specify the `-m` option in the db2mdapiclient utility.

Files that contain output parameters

- **Response:** `output\Advise_response.xml`
- **Metadata:** `<empty>`

  Do not specify the `-n` option in the db2mdapiclient utility.

Appendix. Messages

The following messages are from the server, API, and OLAP Center of DB2 Cube Views.

Socket error: Opening and closing a database connection multiple times can cause a socket error. In rare circumstances, a socket error can occur when you run DB2 Cube Views with DB2 Universal Database Enterprise Server Edition, Version 8.1.2 in a partitioned environment on Windows 2000 Advanced Server. The error might occur if you repeat the following steps more than 10 000 times quickly within a single Windows session:

1. Open a connnection to a DB2 database.
2. Call the DB2 Cube Views stored procedure to perform a metadata operation.
3. Close the database connection.

The workaround is to restart the Windows workstation to reactivate the socket.

SQLSTATE, API, and other server messages

API SQL states

01HQ1: See output XML and server logs.

Explanation

The call to the stored procedure completed, but errors were detected during the execution of one of the requested metadata operations.

User Response

Check the contents of the output parameters of the stored procedure for information. You can also check the entries in the server logs for more information.

38Q00: See server logs for more information.

Explanation

The call to the stored procedure failed. The requested metadata operation or operations were not executed. No information was returned from the stored procedure through the output parameters.

User Response

Check the entries in the server logs for more information.

38Q01: Installation path is unknown.

Explanation

The call to the stored procedure failed because the DB2 installation directory cannot be determined by the stored procedure process. The requested metadata

operation or operations were not executed. No information was returned from the stored procedure through the output parameters.

User Response

If you use a Windows operating system, ensure that the DB2PATH environment variable is set to the correct value either by default or by user action. Restart the database manager and then reissue the call to the stored procedure. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

38Q02: Cannot open the server log file.

Explanation

The call to the stored procedure failed because at least one of the log files that is used by the stored procedure could not be opened for writing by the stored procedure process. The requested metadata operation or operations were not executed. No information was returned from the stored procedure through the output parameters.

User Response

Ensure that the log files specified in the stored procedure configuration file (for example, olap_config.xml) can be created or opened for reading and writing on the appropriate file system. If the log files do not already exist, the stored procedure attempts to create these files. On AIX, ensure that the log files can be read and written to by the fenced database user ID.

38Q03: Metadata input parameter missing.

Explanation

The call to the stored procedure failed because the requested metadata operation requires that metadata be passed as input to the stored procedure, but no metadata was provided through the input metadata parameter. No information was returned from the stored procedure through the output parameters.

User Response

Provide the metadata necessary using the input metadata stored procedure parameter for the requested metadata operation the next time you make a call to the stored procedure.

38Q04: [*error_type*] ERROR: Response output buffer too small.

Explanation

The call to the stored procedure failed because the output parameter buffer for the operation response is too small to accommodate the CLOB structure being returned. No information was returned from the stored procedure via the output parameters.

User Response

Recatalog the stored procedure by using a larger size for the output response parameter.

Common

Success codes

0: Operation completed successfully. No errors were encountered.:
Explanation

The requested metadata operation completed successfully. No errors were encountered during execution of the operation.

User Response

This information is for your information only. No action is required.

1: Operation completed. Additional information was returned.:
Explanation

The requested metadata operation completed. The operation has returned additional information that may describe warning or error situations.

User Response

Check the INFO element for the additional information returned.

2: Operation completed. No changes were made to the metadata.:
Explanation

The requested metadata operation completed. The operation resulted in no changes being made to the metadata in the database catalog.

User Response

Reissue the metadata operation request by using a different mode if you want changes to be made to the metadata in the database catalog.

Common external error/warning codes

100: Failed to allocate memory for *operation***. Make sure that memory is available.:**
Explanation

During execution of the requested metadata operation, the stored procedure failed to allocate required memory segments.

User Response

Increase the memory that is available to the fenced stored procedure process.

101: An internal error occurred while processing the *object name* **object.:**
Explanation

During execution of the requested metadata operation, an unexpected internal error was encountered.

User Response

Contact IBM Customer Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

102: The output buffer of size *buffer_size* **is too small. Change the buffer size to at least** *size***.:**
Explanation

The output parameter buffer available to the stored procedure is too small to accommodate the CLOB structure generated by the stored procedure.

User Response

If possible recatalog the stored procedure using larger sizes for the OUT and INOUT parameters. Otherwise, you must restrict your query so that less information is returned by the stored procedure.

103: A valid license for this product does not exist.:
Explanation

No metadata operations can be performed because a valid product license does not exist for this installation of the product.

User Response

Install a valid product license on the system, or contact IBM Software Support or IBM Software Sales to purchase a new product license.

104: An internal error occurred. The following tokens were returned: *token0***,** *token1***,** *token2***,** *token3***.:**
Explanation

During execution of the requested metadata operation, an unexpected internal error was encountered.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation that was attempted. If possible, also provide the stored procedure log files from the database server.

599: The operation was not executed.:
Explanation

An error was encountered prior to executing this operation. As a result, this operation was not executed.

User Response

Check the results of previous metadata operations that were executed during the same stored procedure call. You can also check the entries in the server logs for more information. After you correct the problems that caused the earlier operation to fail, call the stored procedure again and request the same metadata operations.

Common internal error/warning codes

600: The input *parameter_name* **parameter is invalid with this message:** *message***. Check the parameter and try again.:**
Explanation

One of the parameters that was passed as input to a method internal to the stored procedure is invalid.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

601: The input *parameter_name* **parameter is NULL.:**
Explanation

One of the parameters that was passed as input to a method internal to the stored procedure has an invalid value of NULL.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

602: The *parameter_name* **parameter with value** *value* **is not in the valid range of** *range_value1***,** *range_value2***.:**
Explanation

One of the parameters that was passed as input to a method internal to the stored procedure has a value outside of the valid range.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

603: The Unicode String *string* **is either incorrect or invalid. There might be a memory problem.:**
Explanation

A Unicode string in the stored procedure is either incorrect or invalid. This might indicate a memory problem on the system or in the stored procedure. This might also be the result of the wrong version of the ICU libraries being loaded by the stored procedure.

User Response

Ensure that there is adequate memory available to accommodate the volume of data being processed by the stored procedure. Ensure that the version of the ICU libraries intended for use with the current version of the stored procedure are being loaded. You might need to check the runtime library search path set in the environment to determine correct setup.

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

604: Failed to convert the contents of the string *string* **from Unicode to the active code page of** *code_page***.:**
Explanation

Conversion of a Unicode string object to a string using another encoding failed. This might indicate a memory problem on the system or in the stored procedure. This might also indicate a code page conversion problem on the system, or the wrong version of the ICU libraries was loaded by the stored procedure.

User Response

Ensure that the necessary ICU codepage conversion files are installed on the database server system. Ensure that there is adequate memory available to accommodate the volume of data being processed by the stored procedure. Ensure that the version of the ICU libraries intended for use with the current version of the stored procedure are being loaded. You may need to check the runtime library search path set in the environment to determine correct setup.

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

605: The allocated memory at *memory_buffer* **needs to be freed.:**
Explanation

A method internal to the stored procedure has returned a memory buffer that needs to be freed by another internal method.

User Response

A method internal to the stored procedure should free the returned memory buffer. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

606: Conversion of XMLCh to UChar for *UChar* **failed.:**
Explanation

Conversion between an XMLCh character and a UChar character failed.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

607: The input buffer size of *size* **is too small. Change the buffer size to at least** *new_size***.:**
Explanation

A memory buffer internal to the stored procedure is too small to accommodate the text for a required message.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

608: The type of *stored_procedure_name* **is not valid in the current context.:**
Explanation

An unexpected type was encountered during the stored procedure processing.

User Response

For further assistance, contact IBM Software Support for further assistance with the status ID and text for the metadata operation that you attempted. If possible, provide the stored procedure log files from the database server.

609: The *data_type* **data type is not valid in the current context.:**
Explanation

An unexpected data type was encountered during the stored procedure processing.

User Response

For further assistance, contact IBM Software Support for further assistance with the status ID and text for the metadata operation that you attempted. If possible, provide the stored procedure log files from the database server

MDOBJECT.LIB errors

1000: Failed to clone object *object_name*.**:**
Explanation

An error occurred while cloning a class object internal to the stored procedure.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1001: The deployment for *hierarchy_name* **hierarchy cannot be set to recursive because the hierarchy has more than two levels.:**
Explanation

The hierarchy has too many levels to be recursive deployment.

User Response

Modify the hierarchy to have two or fewer levels before changing the deployment to recursive.

1002: The called function *function_name* **is not supported.:**
Explanation

A virtual method internal to the stored procedure was not implemented for one of the stored procedure's classes.

User Response

Contact IBM Customer Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1003: The container is unexpectedly empty.:
Explanation

A container structure internal to the stored procedure is unexpectedly empty.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1004: The *object_name* **object cannot be found in the container.:**
Explanation

An object that was searched for in one of the stored procedure's internal container structures is unexpectedly missing.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1005: A duplicate of the *element_name* **element already exists in the container.:**
Explanation

An object that should not already exist in one of the stored procedure's internal container structures already exists.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1006: An exception occurred during a list operation.:
Explanation

An unexpected exception occurred while executing an operation on one of the stored procedure's internal list structures.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1007: An internal error occurred in the container with error code *error and number* **and msg** *message***.:**

Explanation

An error occurred while executing an operation on one of the stored procedure's internal container structures.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1008: The copy operation did not copy all properties completely. The copy operation failed for the *property_name* **property with value** *value***.:**
Explanation

An error occurred while executing a copy operation on one of the stored procedure's internal objects. One of the internal object's properties failed to be copied.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1009: The object type of *type1* **is not valid.** *type2* **expected.:**
Explanation

An unexpected object type was encountered during stored procedure processing.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1010: The *parameter_name* **parameter does not have a complete ID.:**
Explanation

One of the parameters that was passed as input to a method internal to the stored procedure is a metadata object ID that is incomplete.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1011: The *object_name* **object does not have a complete ID.:**
Explanation

A metadata object ID is unexpectedly incomplete in the stored procedure.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1012: The *parameter_name* **parameter is the same as the object.:**
Explanation

One of the parameters passed as input to an object method internal to the stored procedure is an object that is unexpectedly equal to the object owning the method.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1013: An unexpected NULL pointer was encountered.:
Explanation

An unexpected NULL pointer was encountered during stored procedure processing.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1014: The container cursor reached the end of the container.:
Explanation

A cursor on one of the container structures internal to the stored procedure has unexpectedly reached the end of the container.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1015: The *object_name* **object is invalid. Reason: ID=***ID***, Message=***message***.:**
Explanation

A metadata object internal to the stored procedure is invalid.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

1016: The *object_name* **object has a list that is of type** *list_type* **and that is unexpectedly empty.:**

Explanation

A container structure that is internal to the stored procedure is unexpectedly empty.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

MDOPERATION.LIB errors

2001: The generated query *query* **does not contain the required column objectType.:**
Explanation

An SQL query generated by the stored procedure is missing a required column.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

MDPARSER.LIB errors

3001: An XML exception was encountered by the parser during *operation* **with message** *message***.:**
Explanation

An unexpected exception was encountered in the stored procedure while parsing the XML passed to the stored procedure.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid for the XML schema that is published for this product. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

3002: An unexpected parser exception was encountered in *operation***.:**
Explanation

An unexpected exception was encountered in the stored procedure while parsing the XML passed to the stored procedure.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid for the XML schema that is published for this product. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

3003: SAXParseException encountered by parser during *operation* **with message** *message***.:**
Explanation

An unexpected exception was encountered in the stored procedure while parsing the XML passed to the stored procedure.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid for the XML schema that is published for this product. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

3004: The system failed to get parser error message for *operation***.:**
Explanation

An unexpected error occurred in the stored procedure while parsing the XML passed to the stored procedure. An error message from the XML parser could not be retrieved.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid for the XML schema that is published for this product. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

3100: The system failed to parse XML for *parameter type* **(line:** *line***, char:***character***, message:** *message***).:**
Explanation

The stored procedure could not parse the input XML. The input XML might not be well formed or it might be invalid for the XML schema that was published for this product.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid for the XML schema that is published for this product.

3101: An unknown metadata object was encountered. *parser_message***.:**
Explanation

An unknown type of metadata object exists in the XML passed to the stored procedure. This input XML cannot be processed by the stored procedure.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid for the XML schema that is published for this product.

3102: An unknown XML attribute was encountered. *attribute_name***,** *attribute_value***.:**

Explanation

An unknown type of XML attribute exists in the XML passed to the stored procedure. This input XML cannot be processed by the stored procedure.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid against the XML schema that is published for this product.

3103: An invalid enumeration value was encountered by the handler for the attribute with name *name* **and value** *value***.:**
Explanation

An invalid enumeration value exists in the XML passed to the stored procedure. This input XML cannot be processed by the stored procedure.

User Response

Ensure that the XML passed to the stored procedure is well formed and that it is valid against the XML schema that is published for this product.

3500: Data is required for the attribute or element name *name***.:**
Explanation

The stored procedure failed to set the value for the indicated XML attribute or element in the XML that is to be returned by the stored procedure.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

3501: Data is required for the attributes *attribute_name1* **and** *attribute_name2***.:**
Explanation

The stored procedure failed to set the value for the indicated XML attribute or element in the XML that is to be returned by the stored procedure.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

3502: An invalid enumeration value was encountered by the formatter for the attribute with name *name* **and value** *value***.:**
Explanation

An invalid enumeration value was encountered in the stored procedure while formatting the XML that was to be returned by the stored procedure.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

MDDATABASE.LIB errors

4000: The database connection failed. Database name *database_name*, **User name** *user_name*.:
Explanation

The stored procedure failed to establish its own connection to the database.

User Response

Ensure that the user ID that the stored procedure uses has the appropriate privileges to connect to the database.

4001: The database connection was not issued because a connection already exists.:
Explanation

The stored procedure unexpectedly encountered a duplicate internal connection to the database. The requested metadata operation could not be performed.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4002: The database operation failed.:
Explanation

An error occurred while executing an SQL statement issued by the stored procedure to the database.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4003: Execution of the CLI call *call_name* **failed.:**
Explanation

An error occurred while executing the indicated CLI call.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4004: The returned data is truncated.:

Explanation

Diagnostic information that was returned during the failed database operation was truncated.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4005: A warning was received from the database. SQLSTATE=*code***, Message=***message***.:**
Explanation

Warning information was returned by a CLI call that was issued by the stored procedure.

User Response

Check the database manager log files on the client and on the server.

4008: An unknown DB2 data type was encountered.:
Explanation

An unknown data type was encountered by the stored procedure while executing a database request.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4009: No valid savepoint name was generated.:
Explanation

The stored procedure was unable to generate a valid database transaction savepoint name. The stored procedure uses its database application ID to form the savepoint name.

User Response

Reissue the call to the stored procedure. Reissuing the call might generate a new database application ID for the stored procedure, and it might therefore allow for the generation of a valid savepoint name. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4010: The attempt to set a DB2 savepoint failed.:

Explanation

The stored procedure was unable to set a database transaction savepoint. A savepoint with the same name as the one used by this instance of the stored procedure may already exist in the current transaction.

User Response

If possible, release the savepoints for the current transaction before reissuing the call to the stored procedure. You can also reissue the call to the stored procedure by using a new transaction.

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4011: A savepoint was not set prior to this point of execution.:
Explanation

A transaction savepoint is unexpectedly missing at a point in the stored procedure. The missing savepoint was possibly not set by the stored procedure, or the savepoint might was released through database actions that were done outside of the stored procedure.

User Response

Reissue the call to the stored procedure. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4012: There was an invalid savepoint string storage.:
Explanation

The database transaction savepoint name was not stored correctly in a data structure internal to the stored procedure possibly because not enough memory is available to the stored procedure process.

User Response

Reissue the call to the stored procedure. If the problem persists, then increase the memory available to the fenced stored procedure process. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4013: The savepoint failed to clear.:
Explanation

The stored procedure was unable to clear a database transaction savepoint. The stored procedure possibly did not set the missing savepoint, or the savepoint was possibly released through database actions that were done outside of the stored procedure.

User Response

Reissue the call to the stored procedure. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4014: Attempting to determine the DB2 AUTOCOMMIT setting failed.:
Explanation

An attempt by the stored procedure to determine whether the DB2 AUTOCOMMIT feature is enabled or disabled failed.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4015: Attempting to set DB2 AUTOCOMMIT OFF failed.:
Explanation

An attempt by the stored procedure to disable the DB2 AUTOCOMMIT feature failed.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4016: No data was returned from the CLI call SQLFetch().:
Explanation

No data was returned to the stored procedure by the CLI function SQLFetch(). This might be acceptable, but the stored procedure should not have allowed this error to propagate through the stored procedure unchanged.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4017: *Object_name* **object was not properly constructed.:**
Explanation

An database object internal to the stored procedure was not initialized properly.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4018: The database disconnect failed.:
Explanation

The stored procedure failed to disconnect from the database.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4019: DB2 SQL error - SQLCODE *sqlcode*, **SQLSTATE** *sqlstate*, **SQLMESG** *sqlmesg*.**:**
Explanation

An error occurred while executing an SQL statement that was issued by the stored procedure to the database.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4020: DB2 SQL error - No details are available.:
Explanation

Diagnostic information is not available for an error that occurred while executing an SQL statement that was issued by the stored procedure to the database.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4021: DB2 SQL error - No details are available.:
Explanation

An error occurred while trying to gather diagnostic information for another error that occurred while executing an SQL statement that was issued by the stored procedure to the database.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4022: The allocation of DB2 *handle_name* **handle failed.:**

Explanation

An error occurred while trying to allocate a DB2 handle within the stored procedure.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4023: Freeing the DB2 *handle_name* **handle failed.:**
Explanation

An error occurred while trying to free a DB2 handle within the stored procedure.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4028: The transaction was not stopped.:
Explanation

An error occurred while trying to end the stored procedure's transaction.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4029: Duplicate rows found sharing the same name and schema in a main object table.:
Explanation

Duplicate rows sharing the same name and schema were unexpectedly found in one of the metadata catalog tables. This sharing is indicative of an internal error in the stored procedure.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4030: The DBINFO structure was not initialized. Make sure that the stored procedure was created in the database by using the DBINFO option.:
Explanation

A DBINFO structure was not received by the stored procedure from the database client.

User Response

Ensure that the stored procedure is cataloged in the appropriate database by using the DBINFO option.

4031: Setting the schema as DB2INFO failed.:
Explanation

The stored procedure failed to set DB2INFO as the current schema.

User Response

Check the database manager log files on the client and on the server. Reissue the call to the stored procedure.

4032: The operation failed due to a collision between an object in the main object table and the object being inserted.:
Explanation

An SQL INSERT statement failed in the stored procedure because it will result in a duplicate metadata object entry in one of the metadata catalog tables.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4033: The operand of the column function includes a column function.:
Explanation

A column function nested in another column function was detected in one of the SQL statements issued by the stored procedure. Column functions cannot be nested in SQL statements.

User Response

Modify the SQL expression template for the input attribute or measure object so that nested column functions are no longer present in the SQL statements generated by the stored procedure.

4034: The DB2 ISOLATION LEVEL setting was not determined.:
Explanation

An attempt by the stored procedure to determine the database transaction isolation level failed. The isolation level could not be determined.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4035: Setting the DB2 ISOLATION LEVEL to READ STABILITY failed.:

Explanation

An attempt by the stored procedure to set the database transaction isolation level failed. The stored procedures requires an isolation level of READ STABILITY.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4036: The version of DB2 Universal Database that is currently installed could not be determined.:
Explanation

An attempt by the stored procedure to determine the version level of the database manager failed.

User Response

Check the database manager log files on the client and on the server. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

4037: DB2 Universal Database, Version *version_number*, **FixPak** *number* **is currently installed but is not compatible with this version of DB2 Cube Views. Upgrade your version of DB2 Universal Database.:**
Explanation

The version DB2 Universal Database that is currently installed is not compatible with this version of DB2 Cube Views. Upgrade your version of DB2 Universal Database so that it is at or above the DB2 Cube Views version level.

User Response

Ensure that compatible versions of DB2 Universal Database and DB2 Cube Views are installed on the same server. Refer to the installation and setup documentation for more information.

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation that you attempt. If possible, also provide the stored procedure log files from the database server.

4038: an SQL statement could not be processed because it is too long or too complex.:
Explanation

A statement was issued by the stored procedure that could not be processed because it exceeds a system limit for either length or complexity, or because too many constraints or triggers are involved.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation that you attempted. If possible, also provide the stored procedure log files from the database server.

4039: The required SYSINFOVERSION metadata table does not exist in the DB2 catalog tables. Migrate your metadata.:
Explanation

The required SYSINFOVERSION metadata table does not exist in the DB2 catalog tables. Migrate your metadata.

User Response

Update the DB2 catalog tables by migrating your metadata. See the *DB2 Cube Views Guide and Reference* book for information on migrating. For additional assistance, contact IBM Software Support with the status ID and text of the metadata operation that you attempted. If possible, provide the stored procedure log files from the database server.

4040: The metadata table version does not match the current version of the DB2 Cube Views API. The version of your metadata tables is *version_number1***and the current API version is** *version_number2***. You need to migrate your metadata tables to the current API version.:**
Explanation

The VERSION value in the SYSINFOVERSION table does not match the version of the DB2 Cube Views API that you are trying to perform an operation with. You need to migrate the metadata tables and ensure that the correct VERSION value is in the SYSINFOVERSION table.

User Response

Migrate the metadata tables in the DB2 catalog. See the *DB2 Cube Views Guide and Reference* for information on migrating. For further assistance, contact IBM Software Support with the status ID and text for the metadata operation that you attempted. If possible, provide the stored procedure log files from the database server.

4041: The version of your metadata tables is not specified. You need to migrate your metadata tables to the current API version.:
Explanation

The VERSION value in the SYSINFOVERSION table does not exist. You need to run the db2mdmigrate.sql script to migrate the metadata tables in the DB2 catalog.

User Response

Migrate the existing metadata tables. See the *DB2 Cube Views Guide and Reference* for information on migrating. For further assistance, contact IBM Software Support with the status ID and text for the metadata operation that you attempted. If possible, provide the stored procedure log files from the database server.

4042: Multiple or duplicate versions of your metadata tables are specified. Check your metadata tables to ensure that only one, correct version is specified.:

Explanation

More than one version or duplicate versions of your metadata tables are specified in the SYSINFOVERSION catalog table. You should have only one, correct version specified in the SYSINFOVERSION table.

User Response

Contact IBM Software Support for further assistance. If possible, provide the stored procedure log files from the database server.

MDUTILITY.LIB errors

5000: The utility failed to parse string *string***.:**
Explanation

A method internal to the stored procedure encountered an error while parsing an internal string value.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

5001: The utility number format did not initialize successfully. Make sure that enough memory is available.:
Explanation

The ICU number formatter was not initialized properly within the stored procedure. This might be the result of inadequate memory resources available to the stored procedure process. This might also be the result of the wrong version of the ICU libraries being loaded by the stored procedure.

User Response

Increase the memory available to the fenced stored procedure process, and reissue the call to the stored procedure. Ensure that the version of the ICU libraries intended for use with the current version of the stored procedure are being loaded. You might need to check the run-time library search path set in the environment to determine correct setup.

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

5002: The utility resource bundle did not initialize successfully. Error code=*code***. Make sure the bundle exists and it is in the path** *path_name***.:**
Explanation

An ICU resource bundle was not initialized properly within the stored procedure. The improper initialization might be the result of the following problems: adequate memory resources are not available to the stored procedure process; the wrong version of the ICU libraries were loaded by the stored procedure; or the wrong resource bundle was loaded for the stored procedure.

User Response

Increase the memory available to the fenced stored procedure process, and reissue the call to the stored procedure. Ensure that the version of the ICU libraries intended for use with the current version of the stored procedure are being loaded. You might need to check the run-time library search path set in the environment to determine correct setup. Ensure that the correct version of the stored procedure's resource bundle was installed on the database server system.

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

5003: The data path from the environment variable *variable_name* **was not found. Check that the environment variable is set properly.:**
Explanation

A DB2 environment variable that is used by the stored procedure is not set.

User Response

Ensure that DB2 was installed correctly on the system. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

5004: The target stream is closed.:
Explanation

A data stream that is used internally by the stored procedure is unexpectedly closed. There might not be enough filehandles available on the database system.

User Response

Ensure that enough filehandles are available from the operating system. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

5005: The target is writing the characters by using the default encoding.:
Explanation

The default encoding documented for the stored procedure is being used by the stored procedure to write data to files on the database server file system.

User Response

Applications that read the files that are written to by the stored procedure must be able to interpret data encoded in the default encoding of the stored procedure.

5006: The input log string of *string* **is not written. The level of the string is** *string_level* **and the level of log is** *log_level***.:**
Explanation

The current logging level does not allow the indicated message to be written to one of the log files set for the stored procedure.

User Response

Modify the logging level if the indicated message needs to be written to one of the log files of the stored procedure.

5007: The message text for the error code *code* **was not found.:**
Explanation

The text for the indicated error code was not found in the resource bundle file of the stored procedure. The wrong version of the resource bundle file might be in use.

User Response

Ensure that the correct version of the resource bundle file of the stored procedure was installed on the database server system. Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

5008: There was a failure while accessing *operation* **for the global static MsgBase object.:**
Explanation

An error occurred in the stored procedure while trying to access an internal message object.

User Response

Contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

External API errors

6000-6199

6000: OLAPMSG() failed with error code *code***.:**
Explanation

The stored procedure failed during execution.

User Response

Based on the return code either fix the problem and reissue the call to the stored procedure, or contact IBM Software Support for further assistance. If contacting IBM Software Support, provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

6001: The instantiated SQL template(s) for the *object_name* **object is invalid with a value of** *value***. Reason ID=***ID***, Message** *message***.:**
Explanation

The instantiated SQL template is the SQL statement fragment that can be formed by combining the SQL expression templates for all of the attributes and measures

involved in a composite attribute or composite measure. A problem was found with the instantiated SQL template for the specified object.

User Response

See the reason ID and message specified. Reissue the call to the stored procedure after you make any changes that was suggested by the reason message.

6002: The *object1* object references *object2* object, but the *object2* object does not exist in the database.:
Explanation

Database objects can reference other objects only if those other objects exist in the database.

User Response

Create the object to be referenced in the database, and then reissue the metadata operation request. Or remove the reference to the missing object, and then reissue the metadata operation request.

6003: The *log_name* log within the specified path could not be opened. Make sure that the specified path exists and that the file has write access.:
Explanation

At least one of the log files that was used by the stored procedure could not be opened.

User Response

Ensure that the path specified in the stored procedure configuration file exists. Ensure that user ID running the stored procedure on the database server has the authority to create, read, and write the required log files.

6005: The input metadata parameter is unexpectedly empty for this operation. The missing metadata parameter is required for this operation.:
Explanation

The requested metadata operation requires that metadata be provided as input. The stored procedure parameter for exchanging metadata is unexpectedly empty.

User Response

Reissue the metadata operation request with the required metadata.

6006: No objects were found matching search criteria: *search_criteria*.:
Explanation

The metadata operation found no metadata objects matching the search criteria specified. No changes were made to the contents of the metadata catalog.

User Response

Reissue the metadata operation with new search criteria if you want to change the contents of the metadata catalog.

6007: Collision(s) between object(s) in the catalog and object(s) being imported were encountered. No changes were made to the metadata.:
Explanation

Collisions were detected between the objects being imported and the objects that already exist in the metadata catalog. Due to the import mode that you specified, no changes were made to the objects in the metadata catalog.

User Response

Reissue the metadata operation by using a different import mode if you want to change the contents of the metadata catalog.

6008: A duplicate *object* **exists within** *metadata_input* **with identity** *ID***.:**
Explanation

Duplicate metadata objects were detected in the metadata input for this metadata operation. Duplicate objects are not allowed as input for metadata operations.

User Response

Remove the duplicate metadata object from the input metadata, and reissue the metadata operation.

6009: An object sharing the same identity as the input *object_name* **object already exists in the metadata catalog.:**
Explanation

The metadata operation could not be performed because a metadata object with the same identity already exists in the metadata catalog.

User Response

Drop the object from the metadata catalog that shares the same identity as the object being created before reissuing the failed metadata operation. Alternatively, you can alter the existing object to match the properties of the new object being created. Otherwise, you must exclude the new object that is causing this error from the metadata operation being performed.

6010: The reference to the *object_name* **object already exists for the input** *object_name* **object.:**
Explanation

A reference between the specified objects is already defined in the metadata catalog. Duplicate references are not allowed.

User Response

Remove one of the duplicate references from the metadata operation request.

6011: The *object_name* **object's schema cannot be changed using the rename operation.:**
Explanation

The rename operation cannot be used to change the schema of a metadata object.

User Response

Ensure that the schema that is specified for the object being renamed remains constant, or use the alter operation.

6013: The version *version1* **of the XML schema used by the client is not supported by the API on the server. The API on the server supports version** *version2* **of the XML schema.:**
Explanation

The version of the XML schema that is used by the client and embedded in the input parameter strings is not supported by the version of the stored procedure on the server.

User Response

Ensure that the client application and the stored procedure use the same version of the XML schema published with this product.

6014: The SQL template(s) for the *object_name* **object cannot be formulated. Reason ID** *ID***, Message** *message***.:**
Explanation

The stored procedure formulates the SQL templates for attributes and measures by combining the SQL expression templates for all of the attributes and measures involved in a composite attribute or composite measure. A problem was encountered in the stored procedure during the formulation of an SQL template.

User Response

See the reason ID and message specified. Reissue the call to the stored procedure after you make any changes suggested by the reason message.

6015: The database user ID does not have the authority to create a database schema in the active database.:
Explanation

The user ID that owns the stored procedure process on the database server does not have the authority to create a database schema in the active database. A database schema is created for each unique metadata object schema.

User Response

Check the database manager log files on the client and on the server. Grant the authority to create a schema in the active database to the user ID that owns the stored procedure process. Reissue the call to the stored procedure.

6016: The database user ID does not have the authority to perform a required action in the active database. The following error message was returned from the database server: *message***.:**
Explanation

The user ID that owns the stored procedure process on the database server does not have the authority to perform a required action in the active database.

User Response

Check the database manager log files on the client and on the server. Grant the required authority to the user ID that owns the stored procedure process. Reissue the call to the stored procedure.

6017: The *object_name* **object does not exist in the metadata catalog.:**
Explanation

The requested operation requires that the indicated object exist in the metadata catalog.

User Response

Create the indicated object in the metadata catalog before reissuing the metadata operation request.

6018: A required table does not exist in the database. The following error message was returned from the database server: *message***.:**
Explanation

A table required by the requested operation does not exist in the database.

User Response

If the missing table is a user table, then create the table and reissue the metadata operation request. If the missing table is a metadata catalog table or a database system table, then contact IBM Software Support for further assistance. Provide the status ID and text for the metadata operation attempted. If possible, also provide the stored procedure log files from the database server.

6019: The API operation uses Version *version_number* **of the XML and the metadata uses version** *version_number* **of the XML. The API operation and metadata XML versions must match.:**
Explanation

The version of the API operation XML must match the version of the metadata XML.

User Response

Ensure that the API operation XML and the metadata XML are using the same version number.

6020: The *operation_name* **API operation does not support version** *version_number* **of the metadata XML.:**
Explanation

The API operation XML must be version 8.2.0.1.0 and the metadata XML must be 8.1.2.1.0 for the TRANSLATE operation.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on metadata rules, metadata validation, and query optimization.

6021: You can use the *version_number* **XML for only the DESCRIBE operation.: Explanation**

You can use the specified XML version for only the DESCRIBE operation.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on metadata rules, metadata validation, and query optimization.

Validation errors

6200: The *object_name* **object is not complete. Make sure that the required properties are set.:**
Explanation

The specified input object does not have all of its required properties set.

User Response

Set the required properties for the given object and issue the metadata operation request again.

6201: The *hierarchy_name* **hierarchy is invalid because it is of type recursive, but is does not have exactly two attributes.:**
Explanation

A recursive hierarchy must reference exactly two attributes. The identified hierarchy violates this rule.

User Response

Modify the identified hierarchy to reference exactly two attributes. See the *Setup and User's Guide* for more information about metadata rules.

6202: The *object_name* **object must have at least one SQL template.:**
Explanation

Based on the metadata object definitions that is provided in the product documentation, the identified measure must have at least one SQL template defined for it.

User Response

Modify the identified measure so that it has at least one SQL template defined. See the *Setup and User's Guide* for more information about metadata rules.

6206: The *attribute_name* **attribute should have only one SQL template.:**
Explanation

Based on the metadata object definitions that are provided in the product documentation, the identified measure must have only one SQL template defined for it.

User Response

Modify the identified measure so that it has only one SQL template defined. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6207: The *attribute_name* **attribute is part of a join but has no column reference.:**
Explanation

The identified attribute object must refer to a database column for it to be validly referenced by a metadata join object.

User Response

Modify the identified attribute object so that it refers to a database column, or modify the related join object so that it refers to a different attribute object where the different attribute object refers to a database column. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6208: The *attribute_name* **attribute is part of a join and it must point to the same table as** *table_name***.:**
Explanation

The first identified attribute object must refer to the same database table as the other object identified.

User Response

Modify the first identified attribute so that it refers to the same database table as the other object identified, or modify the related join object so that it refers to a different attribute object where the different attribute object refers to the same database table as the other object identified. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6209: The schema of *object_name* **object exceeds the maximum length.:**
Explanation

The schema of the identified object exceeds the maximum length.

User Response

Shorten the schema for the identified object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6210: The name of *object_name* **object exceeds the maximum length.:**
Explanation

The name of the identified object exceeds the maximum length.

User Response

Shorten the name of the identified object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6211: The table name of *object_name* **object exceeds the maximum length.:**

Explanation

The table name of the identified object exceeds the maximum length.

User Response

Shorten the name of the table. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6212: The business name of *object_name* **object exceeds the maximum length.:**
Explanation

The business name of the identified object exceeds the maximum length.

User Response

Shorten the business name. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6213: The comments of *object_name* **object exceeds the maximum length.:**
Explanation

The comments of the identified object exceeds the maximum length.

User Response

Shorten the comments of the identified objects. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6214: The schema of *object_name* **object cannot start with SYS.:**
Explanation

The schema for metadata objects cannot begin with the string SYS.

User Response

Use a schema that does not begin with *SYS* for metadata objects. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6215: The schema of *object_name* **object cannot be SESSION.:**
Explanation

The schema for metadata objects cannot be the string SESSION.

User Response

Use a schema that is not the string SESSION for metadata objects. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6216: The name and schema of the *object_name* **object are not complete. Reason ID=***ID***, Message=***message***.:**
Explanation

The name or the schema or both of the identified object is missing or invalid.

User Response

Provide valid strings for both the name and the schema of the identified object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6217: The *hierarchy_name* cube hierarchy is not valid because it references levels that the *hierarchy_name* hierarchy does not refer to.:
Explanation

The cube hierarchy refers to levels that its parent hierarchy does not refer to.

User Response

Alter the cube hierarchy so that it refers only to levels that are also referenced by the parent hierarchy. Alternatively, you can alter the hierarchy so that it refers to the same levels that the cube hierarchy refers to. See the *DB2 Cube Views Guide and Reference* for more information on metadata rules.

Warning codes

6250: The API cannot create a functional dependency for the *level_name*level object because the *level_name* level key attribute does not map to one table column.:
Explanation

The API cannot create the functional dependency for the level object because the level key attribute does not map to a single table column.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies.

6251: The API will not create a functional dependency for the *level_name* level object because the level key attributes correspond to an existing, unique constraint on the dimension table.:
Explanation

The API will not create the functional dependency for the level object because the level key attributes correspond to an existing, unique constraint on the dimension table. The functional dependency is redundant.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies.

6252: The API cannot create a functional dependency for the *level_name* level object because the *level_name* level key attribute is nullable.:
Explanation

The API cannot create a functional dependency for a level object if any of the level key key attributes are nullable.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies.

6253: The API cannot create a functional dependency for the *level_name* level object because the attributes span more than one table.:
Explanation

The API cannot create a functional dependency for a level object if the level key attributes, default attributes, and related attributes span more than one table.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies.

6254: The functional dependency for the *level_name* level object excludes the *attribute_name* default or related attribute because that attribute is already included as a level key attribute.:
Explanation

The functional dependency for a level object excludes related attributes that are already included as level key attributes.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies.

6255: The functional dependency for the *level_name* level object excludes the *attribute_name* default or related attribute because that default or related attribute does not map to one table column.:
Explanation

The functional dependency for the level object excludes the default or related attribute because that attribute does not map to one table column.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies.

6256: The API cannot create a functional dependency for the *level_name* level object because all of the default and related attributes were excluded.:
Explanation

At least one default or related attribute is required to create a functional dependency that corresponds to a level object.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies.

6257: The API cannot create a functional dependency for the *level_name* level object because an error occurred while trying to create the functional dependency.:
Explanation

An error occurred while the API tried to execute an SQL statement that creates the functional dependency.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on functional dependencies. You may also check the entries in the server logs for more information.

6258: The API cannot alter or drop the *level_name* level object because an error occurred while trying to drop the associated *dependency_name* functional dependency.:
Explanation

DB2 Cube Views cannot alter or drop the level object because an error occurred while the API tried to issue an SQL statement that drops the functional dependency. Make sure that you have the authority to create or drop the functional dependency.

User Response

See the *DB2 Cube Views Guide and Reference* for more information on functional dependencies. Check the entries in the server logs for more information.

6299: At least one database view was found during validation. Constraint-related validation checks were not performed for the joins that involve columns of views. All other validation checks were performed.:
Explanation

Constraint-related validation checks were not performed for those joins that were found to involve columns of views. Constraint-related validation checks were performed on all other joins requested, and all remaining validation checks were performed on all requested objects.

User Response

Refer to the *DB2 Cube Views Guide and Reference* for more information on metadata rules, metadata validation, and query optimization.

Rule-related validation errors

6300: The *model_name* cube model does not refer to one or more facts.:
Explanation

A metadata object rule was violated by the identified cube model object. A cube model must refer to one or more facts.

User Response

Alter the identified cube model so that it refers to one or more facts. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6301: The *model_name***cube model does not refer to zero or more dimension(s).:**
Explanation

A metadata object rule was violated by the identified cube model object. A cube model must refer to zero or more dimensions.

User Response

Change the identified cube model so that it refers to zero or more dimensions. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6302: The *model_name***cube model is missing a dimension or join or both for one of its dimension-join pairs.:**
Explanation

A metadata object rule was violated by the identified cube model object. A dimension-join pair for a cube model must refer to both a dimension and a join.

User Response

Change the identified cube model so that all of its dimension-join pairs refer both a dimension and a join. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6303: The *join_name* **join referenced by the** *model_name***cube model is not valid. All the attributes on one of its sides must be referenced by the** *facts_name***facts, and all of the attributes on the other side must be referenced by one of the cube model's dimensions.:**
Explanation

A metadata object rule was violated by the identified cube model object. The joins of a cube model must each refer to the attributes of the cube model's facts on one side, and to the attributes of one of the cube model's dimensions on the other side.

User Response

Change the invalid join for the identified cube model so that all of the attributes on one side of the join come from the cube model's facts, and all of the attributes on the other side of the join come from one of the cube model's dimensions. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6304: One of the aggregations in the *measure_name***measure directly references the** *dimension_name* **dimension, which is not directly referenced by the** *model_name* **cube model.:**
Explanation

A metadata object rule was violated by the identified cube model object. The aggregations in a measure that are used by a cube model must refer only to those dimensions that are used by the same cube model.

User Response

Change the aggregation for the identified measure so that it refers only to those dimensions used by the identified cube model. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6305: The empty-dimension-list aggregation in the *measure_name* measure does not match to at least one previously unmatched dimension from the *model_name* cube model.:
Explanation

A metadata object rule was violated by the identified cube model object. Empty-dimension-list aggregations in the measures that are used by cube models must match to at least one dimension unmatched otherwise in each cube model.

User Response

Change the aggregation for the identified measure so that its empty-dimension-list matches to at least one previously unmatched dimension in the identified cube model. See the *DB2 Cube Views Guide and Reference*for more information about metadata rules.

6306: The *measure_name* measure must contain only the empty-dimension-list aggregation since the *model_name* cube model does not refer to any dimension objects.:
Explanation

A metadata object rule was violated by the identified cube model object. When a cube model does not refer to any dimensions, the cube model's measure must only contain the empty-dimension-list aggregation.

User Response

Change the identified measure so is contains only the empty-dimension-list aggregation. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6307: The *cube_name* cube does not refer to one cube facts object.:
Explanation

A metadata object rule was violated by the identified cube object. A cube must refer to one cube facts object.

User Response

Change the identified cube so that it refers to one cube facts object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6308: The *cube_name* cube does not refer to at least one cube dimension object.:
Explanation

A metadata object rule was violated by the identified cube object. A cube must refer to at least one cube dimension object.

User Response

Change the identified cube so that it refers to at least one cube dimension object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6309: The *cube_facts_name* cube facts referenced by the *cube_name* cube is not derived from the facts object referenced by the *model_name* cube model.:

Explanation

A metadata object rule was violated by the identified cube object. The cube facts that is used by the identified cube must be derived from the facts that is used by the identified cube model.

User Response

Change one or more of the identified objects so that the specified rule is no longer violated. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6310: The *cube_dimension_name* **cube dimension referenced by the** *cube_name* **cube is not derived from one of the dimension objects referenced by the** *model_name* **cube model.:**
Explanation

A metadata object rule was violated by the identified cube object. A cube dimension that is used by the identified cube must be derived from one of the dimensions that is used by the identified cube model.

User Response

Change one or more of the identified objects so that the specified rule is no longer violated. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6311: The *facts_name* **facts object does not refer to any measures.:**
Explanation

A metadata object rule was violated by the identified facts object. A facts object must refer to at least one measure.

User Response

Change the identified facts so that it refers to at least one measure. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6312: Some of the attributes and measures that are referenced by the facts object *facts_name* **cannot be joined using facts object joins.:**
Explanation

A metadata object rule was violated by the identified facts object. The attributes and measures of a facts object must all be joinable using the join objects of the facts.

User Response

Make all the attributes and measures referenced by the identified facts object joinable by referencing more join objects from the facts object. Or remove those attributes from the facts object or measures that are not joinable by the facts' current joins. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6313: The *facts_name* **facts object has multiple joins between two tables.:**

Explanation

A metadata object rule has been violated by the identified facts object. A facts object must not have multiple joins between the same two tables.

User Response

Alter the identified facts object so that it has only one join between any two given tables. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6314: The *facts_name* **facts object contains a join loop.:**
Explanation

A metadata object rule was violated by the identified facts object. The joins for the identified facts object form a path loop. This is not allowed.

User Response

Remove one of the joins that is causing the loop from the identified facts object, or change one of the joins that is causing the loop so that a loop no longer exists. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6315: The *join_name* **join does not refer to only those attributes in the** *facts_name* **facts object.:**
Explanation

A metadata object rule was violated by the identified facts object. The joins of a facts object must refer only to the attributes of the that facts object.

User Response

Change the identified join so that it refers only to the attributes of the identified facts object, or add to the facts object the missing attributes that the identified join object refers to. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6316: The *cube_facts_name* **cube facts object does not refer to a fact object or refers to more than one fact object.:**
Explanation

A metadata object rule was violated by the identified cube facts object. A cube facts object must refer to one facts object.

User Response

Change the identified cube facts object so that it refers to one facts object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6317: The *cube_facts_name* **cube facts object does not refer to any measures.:**
Explanation

A metadata object rule was violated by the identified cube facts object. A cube facts object must refer to at least one measure.

User Response

Change the identified cube facts object so that it refers at least one measure. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6318: The *measure_name* **measure that is referenced by the** *cube_facts_name* **cube facts object is not a part of the** *facts_name* **facts object.:**
Explanation

A metadata object rule was violated by the identified cube facts object. A cube facts object must refer to measures that are referred to by the facts object from which the cube facts object was derived.

User Response

Add the identified measure to the identified facts object, or remove the identified measure from the identified cube facts object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6319: The *dimension_name* **dimension does not refer any attributes. A dimension must refer to at least one attribute.:**
Explanation

A metadata object rule was violated by the identified dimension object. A dimension object must refer to at least one attribute.

User Response

Change the identified dimension object so that it refers to at least one attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6320: Some of the attributes referenced by the *dimension_name* **dimension cannot be joined using dimension joins.:**
Explanation

A metadata object rule was violated by the identified dimension object. The attributes of a dimension object must all be joinable by using the join objects of the dimension.

User Response

Make all the attributes referenced by the identified dimension object joinable by referencing more join objects from the dimension object. Or remove from the dimension object those attributes that are not joinable by the dimension's current joins. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6321: The *dimension_name* **dimension contains a join loop.:**
Explanation

A metadata object rule was violated by the identified dimension object. The joins for the identified dimension object form a path loop. This is not allowed.

User Response

Remove one of the joins that is causing the loop from the identified dimension object, or change one of the joins that is causing the loop so that a loop no longer exists. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6322: The *dimension_name* **dimension has multiple joins between two tables.: Explanation**

A metadata object rule has been violated by the identified dimension object. A dimension object must not have multiple joins between the same two tables.

User Response

Alter the identified dimension object so that it has only one join between any two given tables. Refer to the product documentation for more information about metadata rules.

6323: The *hierarchy_name* **hierarchy references levels that are not referenced by the hierarchy's parent dimension** *hierarchy_name***.: Explanation**

The hierarchies of a dimension can only refer to the levels of that dimension object. This hierarchy refers to levels that the parent dimension does not refer to.

User Response

Alter the hierarchy so that it refers to the levels of only the its parent dimension object, or add the levels that the hierarchy refers to the dimension object. See the DB2 Cube Views Guide and Reference for more information about metadata rules.

6325: The join of a dimension must refer to the attributes of only that dimension. The*join_name* **join refers to attributes that are not in the** *dimension_name***dimension.: Explanation**

The dimension and join violate the metadata object rule that a join of a dimension must only refer to attributes of that dimension.

User Response

Alter the join so that it only refers to the attributes of the parent dimension, or add the attributes that the join refers to the dimension. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6326: The *cube_dimension_name* **cube dimension does not refer to a dimension.: Explanation**

A metadata object rule was violated by the identified cube dimension object. A cube dimension object must refer to a dimension.

User Response

Change the identified cube dimension object so that it refers to a dimension. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6327: The *cube_dimension_name* **cube dimension does not refer to a cube hierarchy.:**
Explanation

A metadata object rule was violated by the identified cube dimension object. A cube dimension object must refer to a cube hierarchy.

User Response

Change the identified cube dimension object so that it refers to a cube hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6328: The *cube_hierarchy_name* **cube hierarchy that is referenced by the** *cube_dimension_name* **cube dimension is not derived from any of the hierarchies that are referenced by the** *dimension_name* **dimension.:**
Explanation

A metadata object rule was violated by the identified cube dimension object. The cube hierarchy used by the identified cube dimension must be derived from one of the hierarchies that is used by the identified dimension.

User Response

Change one or more of the identified objects so that the specified rule is no longer violated. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6329: The *hierarchy_name* **hierarchy does not refer to any levels.:**
Explanation

A hierarchy object must refer to at least one level.

User Response

Alter the identified hierarchy object so that it refers to at least one level. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6330: The *hierarchy_name* **hierarchy, which uses a recursive deployment, does not reference exactly two levels.:**
Explanation

A metadata object rule has been violated by the identified hierarchy object. A hierarchy object that uses recursive deployment must reference two levels.

User Response

Alter the identified hierarchy object so that it refers to two levels. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6332: The type of the *hierarchy_name* **hierarchy is not compatible with its deployment.:**

Explanation

A metadata object rule has been violated by the identified hierarchy object. Compatibility of hierarchy types and deployment are described in the product documentation.

User Response

Change the identified hierarchy so that its type is compatible with its deployment. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6334: The *hierarchy_name* **cube hierarchy must refer to exactly one hierarchy.:**
Explanation

The cube hierarchy violates a metadata object rule that a cube hierarchy must refer to exactly one hierarchy.

User Response

Alter the identified cube hierarchy so that it refers to one hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6335: The *hierarchy_name* **cube hierarchy must refer to at least one cube level.:**
Explanation

A metadata object rule has been violated by the identified cube hierarchy object. A cube hierarchy object must refer to at least one cube level.

User Response

Alter the identified cube hierarchy object so that it refers to at least one cube level. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6336: The *hierarchy_name1* **cube hierarchy references a cube level, but the corresponding** *hierarchy_name2* **hierarchy does not reference the corresponding** *level_name* **level. You must add the** *level_name* **level to the** *hierarchy_name2* **hierarchy, or remove the corresponding cube level from the** *hierarchy_name1* **cube hierarchy.:**
Explanation

The cube hierarchy violates the metadata object rule that a cube hierarchy must refer to cube levels that are derived from the levels referred to by the corresponding hierarchy.

User Response

Add the level to the hierarchy, or remove the level from the cube hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6337: The order of the cube levels in the *hierarchy_name* **cube hierarchy does not match the order of the corresponding levels in the** *hierarchy_name* **hierarchy.:**

Explanation

A metadata object rule has been violated by the identified cube hierarchy object. The relative ordering of the cube levels in a cube hierarchy must be the same as the relative ordering of the same levels in the hierarchy from which the cube hierarchy was derived.

User Response

Alter one of the identified objects so that the relative ordering of the attributes in both identified objects is consistent. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6340: One of the SQL expression templates for the *measure_name* measure uses a parameter that is not an attribute, measure, or column.:
Explanation

A metadata object rule was violated by the identified measure object. The SQL expression templates for measure objects must use parameters that are attributes, measures, or columns.

User Response

Change the identified measure so that its SQL expression templates use attributes, measures, or columns as parameters. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6341: A dependency loop exists among the attributes or measures used as parameters in the SQL expression template for the *measure_name* measure.:
Explanation

A metadata object rule has been violated by the identified measure object. The attributes and measures used as parameters for the SQL expression template of a measure must not form a dependency loop.

User Response

Change the identified measure so that its SQL expression templates do not contain dependency loops that involve their parameters. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6342: The *measure_name* measure has an empty string defined for one of its SQL expression templates.:
Explanation

A metadata object rule was violated by the identified measure object. The SQL expression template for a measure cannot be an empty string.

User Response

Alter the identified measure so that its SQL expression template is no longer an empty string. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6343: The SQL expression template for the *measure_name* measure contains an aggregation function.:

Explanation

A metadata object rule was violated by the identified measure object. The SQL expression template for a measure cannot contain an aggregation function.

User Response

Change the identified measure so that its SQL expression template no longer contains an aggregation function. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6344: The *measure_name* **measure is missing an aggregation or is incorrectly referencing objects other than measures.:**
Explanation

A metadata object rule was violated by the identified measure object. An aggregation is not required for a measure if that measure references at least one other measure and references only measures.

User Response

Change the identified measure by adding an aggregation or ensuring that the identified measure references at least one other measure and references only measures. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6345: The number of SQL expression templates in the *measure_name* **measure does not match the number of parameters used with the first aggregation function.:**
Explanation

A metadata object rule was violated by the identified measure object. The number of SQL templates in a measure must match the number of parameters for the first aggregation function of that measure if an aggregation is present.

User Response

Change the identified measure so that the number of parameters for its first aggregation function matches the number of SQL expression templates in the measure. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6346: The *measure_name* **measure, which has multiple SQL expression templates, does not define at least one step in the aggregation script.:**
Explanation

A metadata object rule was violated by the identified measure object. A measure with multiple SQL expression templates must define at least one step in its aggregation script.

User Response

Change the identified measure so that its aggregation script has at least one step. Or remove one of the measure's SQL expression templates provided the remaining SQL expression template refers to only other measures. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6347: The *measure_name1* **measure has an aggregation script defined. However, it should not have any aggregation scripts defined because the referenced measure,** *measure_name2*, **defines multiple templates for SQL expressions.:**
Explanation

A metadata object rule was violated by the identified measure object. If measure A refers to measure B, which defines multiple SQL templates, then measure A must not have an aggregation script. This rule applies for all levels in a measure reference tree.

User Response

Remove the aggregation script from the measure causing the problem, or remove one of the SQL expression templates from the measure referenced. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6348: The *measure_name* **measure contains a multi-parameter aggregation function that is not used as the first aggregation.:**
Explanation

A metadata object rule was violated by the identified measure object. A multi-parameter aggregation function can only be used as the first aggregation for a measure.

User Response

Make the multi-parameter aggregation function the first aggregation that is used by the identified measure, or remove the multi-parameter aggregation function from the identified measure. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6349: The *measure_name* **measure does not have exactly one empty-dimension-list aggregation.:**
Explanation

A metadata object rule was violated by the identified measure object. When a measure defines one or more aggregations, one aggregation must have an empty list of dimensions.

User Response

Change the identified measure so that it has one empty list of dimensions, or change the identified measure so that it defines no aggregations. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6350: The *dimension_name* **dimension is referenced multiple times in the** *measure_name* **measure.:**
Explanation

A metadata object rule was violated by the identified measure object. In a measure, a dimension cannot be referenced more than once either in an aggregation or across aggregations.

User Response

Alter the identified measure so that it references the identified dimension only once. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6351: The SQL expression template for the *object_name* **object is missing a token indicator with the number** *number***. Token indicators must be consecutively numbered starting with the number 1.:**
Explanation

A metadata object rule was violated by the identified measure object. In a measure's SQL expression template, token indicators must begin with 1 and must be consecutively numbered.

User Response

Change the identified measure so that the token indicators for its SQL expression templates are consecutively numbered starting with 1. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6352: The*measure_name* **measure contains an SQL expression template that does not use the provided reference,** *reference***.:**
Explanation

A metadata object rule was violated by the identified measure object. The SQL expression template for a measure must make use of every column, attribute, and measure reference that is provided. Each reference can be used more than once.

User Response

Change the SQL expression template for the identified measure so that it makes use of every column, attribute, and measure reference that was provided. Or remove the column, attribute, and measure references that are not used by the identified measure's SQL expression templates. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6353: One of the SQL expression templates for the *attribute_name* **attribute uses a parameter that is not an attribute or column.:**
Explanation

A metadata object rule was violated by the identified attribute object. The SQL expression templates for attribute objects must use parameters that are attributes or columns.

User Response

Change the identified attribute so that its SQL expression templates use attributes or columns as parameters. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6354: A dependency loop exists among the attributes used as parameters in the SQL expression template for the *attribute_name* **attribute.:**

Explanation

A metadata object rule has been violated by the identified attribute object. The attributes used as parameters for the SQL expression template of an attribute must not form a dependency loop.

User Response

Alter the identified attribute so that its SQL expression templates do not contain dependency loops involving their parameters. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6355: The *attribute_name* **attribute has an empty string defined in one of its SQL expression templates.:**
Explanation

A metadata object rule was violated by the identified attribute object. The SQL expression template for an attribute cannot be an empty string.

User Response

Change the identified attribute so that its SQL expression template is no longer an empty string. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6356: The SQL expression template for the *attribute_name* **attribute contains an aggregation function.:**
Explanation

A metadata object rule was violated by the identified attribute object. The SQL expression template for an attribute cannot contain an aggregation function.

User Response

Change the identified attribute so that its SQL expression template no longer contains an aggregation function. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6358: The *attribute_name* **attribute contains an SQL expression template that does not use the provided reference,** *reference***.:**
Explanation

A metadata object rule was violated by the identified attribute object. The SQL expression template for an attribute must make use of every column and attribute reference that is provided. Each reference can be used more than once.

User Response

Change the SQL expression template for the identified attribute so that it makes use of every column and attribute reference that is provided. Or remove the column and attribute references that are not used by the identified attribute's SQL expression templates. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6359: The*relationship_name* **attribute relationship does not refer to two distinct attributes.:**

Explanation

A metadata object rule was violated by the identified attribute relationship object. An attribute relationship object must refer to two distinct attributes.

User Response

Change the identified attribute relationship object so that it refers to two distinct attributes. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6360: The *relationship_name* **attribute relationship is incorrectly defined. The cardinality property is set to N:N, but the functional dependency property is set to YES.:**
Explanation

A metadata object rule was violated by the identified attribute relationship object. When the functional dependency property of an attribute relationship is set to YES, then the cardinality property of the attribute relationship cannot be set to N:N.

User Response

Change the identified attribute relationship so that its cardinality is not set to N:N, or its functional dependency property is set to NO. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6361: The *join_name* **join does not refer to at least one triplet. A triplet contains a left attribute, a right attribute, and an operator.:**
Explanation

A metadata object rule was violated by the identified join object. A join object must refer to at least one triplet that contains a left attribute, a right attribute, and an operator.

User Response

Change the identified join object so that it refers to at least one triplet. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6362: The left attributes in the *join_name* **join do not all resolve into a column or columns of a single table.:**
Explanation

A metadata object rule was violated by the identified join object. The left attributes of a join must all resolve into a column or the columns of a single database table.

User Response

Change the identified join object so that its left attributes all resolve into a column or the columns of a single table. Or change the left attributes of the identified join object so that they all comply with the metadata rule stated. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6363: The right attributes in the *join_name* **join do not all resolve into a column or columns of a single table.:**

Explanation

A metadata object rule was violated by the identified join object. The right attributes of a join must all resolve into a column or the columns of a single database table.

User Response

Change the identified join object so that its right attributes all resolve into a column or the columns of a single table. Or change the right attributes of the identified join object so that they all comply with the metadata rule stated. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6364: At least one of the triplets for the *join_name* join does not define a valid operation. The data types of the left and right attributes might not be compatible with each other, or they might not be compatible with the operator.:
Explanation

A metadata object rule was violated by the identified join object. Each triplet of a join object must define a valid operation. The data types for the right and left attributes must be compatible with each other taking into account the operation specified.

User Response

Change the identified join object so that each of its triplets define a valid operation. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6365: The *model_name* cube model does not refer to one and only one facts object.:
Explanation

A metadata object rule was violated by the identified cube model object. A complete cube model must refer to one facts object.

User Response

Change the identified cube model object so that it refers to one facts object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6366: The *model_name* cube model does not refer to one or more dimensions.:
Explanation

A metadata object rule was violated by the identified cube model object. A complete cube model must refer to at least one dimension object.

User Response

Change the identified cube model object so that it refers to at least one dimension object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6367: The cardinality of the*join_name* facts-to-dimension join is not set to either 1:1 or N:1.:

Explanation

The cube model will not benefit from the Optimization Advisor recommendations because the cardinality of the facts-to-dimension join is not 1:1 or N:1. Optimization will not be performed.

User Response

For the cube model to benefit from the Optimization Advisor recommendations, the cardinality for each of the joins that go from the facts to a dimension object must be set to either 1:1 or N:1. The cardinality of the join on the facts attributes must be 1 or N, and the cardinality of the dimension's attributes must be 1. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6368: The *join_name* facts-to-dimension join does not join the table for the *facts_name* facts object to a primary table for the *dimension_name* dimension.:
Explanation

An optimization rule was violated by the identified join object. Considering the join network formed by the dimension's joins, you must have at least one table (the primary table) in which all joins radiating from this table have a cardinality of N:1 or 1:1. In the cube model, the joins from the facts to the dimension objects must involve this primary table of a dimension.

User Response

In the cube model object, make sure that all of the facts-to-dimension joins are from the facts object to the primary table of each dimension. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6369: The *dimension_name* dimension does not have a primary table, as indicated by the join network formed by the joins for the dimension.:
Explanation

An optimization rule was violated by the identified dimension object. Considering the join network formed by the dimension's joins, you must have at least one table in which all joins radiating from this table have a cardinality of N:1 or 1:1. Optimization will not be performed if there is no such primary table for a dimension.

User Response

Check the cardinalities of the join objects used in the dimension. For optimization to be performed the dimension must have a primary table as described in the optimization rules. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6370: The *join_name* join involves columns on which a referential constraint is not defined.:
Explanation

An optimization rule has been violated by the identified join object. There must be a constraint defined on the columns that participate in the join. If the join is a self-join, i.e. the same set of columns is used in both sides of the equality, a primary key must be defined matching the set of columns. In all other cases, when the set of columns of one side are different from the other side of the join, a

primary key must match the columns of one side of the join, and a foreign key must match the other set of columns as well as reference the primary key. Optimization will not be performed due to the missing constraint.

User Response

Create a constraint on the columns that participate in the join. If you do not want the standard constraint because of performance implications, create informational constraints, with query optimization enabled. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6371: A primary key is not defined using the columns involved in the *join_name* **self-join.:**
Explanation

An optimization rule was violated by the identified join object. You must define a constraint on the columns that participate in the join. If the join is a self-join, that is, the same set of columns is used in both sides of the equality, a primary key must be defined that matches the set of columns. Optimization will not be performed due to the missing constraint.

User Response

If the table has a primary key defined, set the attributes of the self-join to attributes representing the primary key columns of the table. Otherwise, create a primary key on the columns that participate in the self-join. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6372: A primary key is not defined using the columns from one side of the *join_name* **join.:**
Explanation

An optimization rule was violated by the identified join object. You must define a constraint defined on the columns that participate in the join. When the set of columns of one side are different from the other side of the join, a primary key must match the columns of one side of the join, and a foreign key must match the other set of columns and reference the primary key. Optimization will not be performed due to the missing constraint.

User Response

Create a primary key on the columns of one side of the join. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6373: A foreign key is not defined using the columns from one side of the *join_name* **join.:**
Explanation

An optimization rule was violated by the identified join object. You must define a constraint defined on the columns that participate in the join. When the set of columns of one side are different from the other side of the join, a primary key must match the columns of one side of the join, and a foreign key must match the other set of columns and reference the primary key. Optimization will not be performed due to the missing constraint.

User Response

Create a foreign key constraint between the primary key columns of the join and the columns of the other side of the join. If you do not want the standard constraint because of performance implications, create informational constraints with query optimization enabled. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6374: The foreign key using the columns from one side of the *join_name* join does not reference the primary key using the columns from the other side of the join.:
Explanation

An optimization rule was violated by the identified join object. You must define a constraint on the columns that participate in the join. When the set of columns of one side is different from the other side of the join, a primary key must match the columns of one side of the join, and a foreign key must match the other set of columns and reference the primary key. Optimization will not be performed due to the missing constraint.

User Response

Create a foreign key constraint between the primary key columns of the join and the columns of the other side of the join. If you do not want the standard constraint because of performance implications, create informational constraints with query optimization enabled. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6375: The cardinality of the *join_name* join is not set to 1:1, N:1, or 1:N.:
Explanation

An optimization rule was violated by the identified join object. Optimization cannot be performed if the join cardinality is M:N.

User Response

Set the join cardinality to 1:1, 1:N or N:1, depending on the constraints on which the join is based. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6376: The cardinality of the *join_name* self-join is not set to 1:1.:
Explanation

An optimization rule was violated by the identified join object. Optimization cannot be performed if the join cardinality of a self-join is not set to 1:1.

User Response

Set the cardinality of the self-join to 1:1. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6377: The cardinality of the *join_name* join is not set to 1 for the side on which the primary key is defined.:

Explanation

An optimization rule was violated by the identified join object. The join cardinality must be 1 on the side in which a primary key is defined and N on the side in which a foreign key is defined. If the foreign key side has also a primary key defined on it, a 1 must be used as cardinality. Optimization cannot be performed if this is not the case.

User Response

The join cardinality should be set to 1 for the side on which the primary key is defined. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6378: The cardinality of the *join_name* **join is not set to N for the side on which the foreign key is defined.:**
Explanation

An optimization rule was violated by the identified join object. The join cardinality must be 1 on the side which a primary key is defined and N on the side which a foreign key is defined. If the foreign key side has also a primary key defined on it, a 1 must be used as cardinality. Optimization cannot be performed if this is not the case.

User Response

The join cardinality should be set to N for the side on which the foreign key is defined. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6379: The cardinality of the *join_name* **join is not set to 1 for the side on which both a primary key and a foreign key are defined.:**
Explanation

An optimization rule was violated by the identified join object. The join cardinality must be 1 on the side which a primary key is defined and 1 for the side on which both a primary key and a foreign key are defined. Optimization cannot be performed if this is not the case.

User Response

The join cardinality should be set to 1:1. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6380: The *attribute_name* **attribute, which is referenced by the** *join_name* **join, does not resolve to a nonnullable SQL expression.:**
Explanation

An optimization rule was violated by the identified join object. All attributes that are used in the join must resolve to nonnullable SQL expressions. Optimization cannot be performed if a join references an attribute that resolves to a nullable SQL expression.

User Response

Remove the reference to the nullable attribute from the join. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6381: The *join_name* join does not have a type of INNER JOIN.:
Explanation

An optimization rule was violated by the identified join object. The join type must set to INNER JOIN. Optimization cannot be performed.

User Response

Change the join to reference only attributes that resolve to a single column. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6382: The *attribute_name* attribute reference of *join_name* join does not resolve to a single column expression, which is a requirement for it to participate in a constraint.:
Explanation

An optimization rule was violated by the identified join object. DB2 constraints must be applied on the attributes referenced by a join. Constraints can only be applied on columns, so the attributes referenced by a join must resolve to single column in a table. Optimization cannot be performed if this is not the case.

User Response

Change the join to reference only attributes that resolve to a single column. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6385: The *model_name* cube model must refer to at least one dimension that has a hierarchy.:
Explanation

The identified cube model violates the optimization rule that requires a cube model to refer to at least one dimension that has a hierarchy.

User Response

Alter the identified cube model's dimension so that the dimension refers to at least one hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about optimization rules.

6386: Each optimization slice must have exactly one optimization level defined per cube dimension in the *cube_name* cube.:
Explanation

The cube does not satisfy the rule that an optimization slice must have exactly one optimization level per cube dimension in the cube.

User Response

Alter the optimization slice so that it refers to one optimization level per cube dimension in the cube object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6387: The optimization level must refer to one cube dimension in the *cube_name* **cube.:**
Explanation

The cube does not satisfy the metadata object rule that an optimization level must refer to exactly one cube dimension that belongs to the cube.

User Response

Alter the optimization level so that it refers to one cube dimension in the cube object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6388: The optimization level must refer to one cube hierarchy in the *cube_name* **cube.:**
Explanation

The cube does not satisfy the metadata object rule that an optimization level must refer to exactly one cube hierarchy that belongs to the cube.

User Response

Alter the optimization level so that it refers to one cube dimension and one cube hierarchy in the cube object.

6389: You must set the optimization level to allLevel, anyLevel or to a cube level reference in the *cube_name* **cube.:**
Explanation

The cube does not satisfy the metadata object rule that an optimization level must have an allLevel, an anyLevel, or a cube level reference.

User Response

Alter the identified optimization level so that it refers to an allLevel, an anyLevel, or a cube level reference in the cube object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6390: In the optimization level, the referenced cube dimension and cube hierarchy must be derived from objects in the *cube_name* **cube. The cube hierarchy must belong to the cube dimension.:**
Explanation

The cube does not satisfy the metadata object rule that the referenced cube dimension and cube hierarchy must be derived from the objects in the cube for the optimization level. The cube hierarchy must belong to the cube dimension.

User Response

Alter the optimization level so that the referenced cube dimension and cube hierarchy are derived from objects in the cube. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6391: In the optimization level, if the cube level reference is not anyLevel or allLevel, then the *level_name* **cube level must belong to the** *hierarchy_name* **cube hierarchy in the** *cube_name* **cube.:**

Explanation

The cube does not satisfy the metadata object rule in the optimization level, if the cube level reference is not anyLevel or allLevel, the cube level must belong to the cube hierarchy.

User Response

Alter the optimization level so that the referenced cube level belongs to the cube hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6392: You cannot have both a MOLAP extract optimization slice and a hybrid extract optimization slice in the *cube_name* **cube.:**
Explanation

The cube does not satisfy the metadata object rule that a cube cannot have both a MOLAP extract optimization slice and a hybrid extract optimization slice.

User Response

Alter one of optimization slices so that you do not have both a MOLAP extract optimization slice and a hybrid extract optimization slice in one cube. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6393: You cannot have more than one MOLAP extract optimization slice in the *cube_name* **cube.:**
Explanation

The cube does not satisfy the metadata object rule that a cube can have a maximum of one MOLAP extract type of optimization slices.

User Response

Alter the cube to have zero or one MOLAP extract type optimization slices. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6394: You cannot have more than one hybrid extract optimization slice in the *cube_name* **cube.:**
Explanation

The cube does not satisfy the metadata object rule that a cube can have a maximum of one hybrid extract type of optimization slices.

User Response

Alter the cube to have zero or one hybrid extract type optimization slices. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6395: The drill through optimization slice can exist only if there is a hybrid extract optimization slice in the *cube_name* **cube.:**
Explanation

The cube does not satisfy the metadata object rule that a drill through optimization slice can exist only if there is a hybrid extract optimization slice in the cube.

User Response

Alter the optimization slice type from drill through to a different type if you do not have a hybrid extract optimization slice in the cube. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6396: The *attribute_name* **attribute that is referenced by the** *level_name* **level in the** *dimension_name* **dimension must be included in the dimension attribute list.:**
Explanation

The dimension does not satisfy the metadata object rule that all attributes that are referenced by the levels in a dimension must be included in that dimension's attribute list.

User Response

Alter the level so that it does not reference the identified attribute, or add the identified attribute to the dimension. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6397: The *level_name* **level must refer to at least one level key attribute.:**
Explanation

The level does not satisfy the metadata object rule that a level must refer to at least one level key attribute.

User Response

Alter the level so that it refers to at least one level key attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6398: The *level_name* **level cannot contain the** *attribute_name* **attribute more than once in the set of level key attributes.:**
Explanation

The level does not satisfy the metadata object rule that the set of level key attributes cannot contain duplicate attributes.

User Response

Alter the level so that it does not include duplicate level key attributes. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6399: The *level_name* **level must refer to exactly one default attribute.:**
Explanation

The level does not satisfy the metadata object rule that a level must have exactly one default attribute.

User Response

Alter the level so that it includes exactly one default attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6400-6499

6401: You cannot use one attribute as both the default attribute and a related attribute in the *level_name* level.:
Explanation

The level does not satisfy the metadata object rule that one attribute cannot be used as both the default attribute and a related attribute.

User Response

Alter the level so that the default attribute is not used as a related attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6402: You cannot use one attribute as both the default attribute and a related attribute in the *level_name* level.:
Explanation

The level does not satisfy the metadata object rule that the set of related attributes cannot contain duplicate attributes.

User Response

Alter the level so that it does not include duplicate related attributes. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6403: The *level_name* level cannot contain the *attribute_name* attribute more than once in the set of related attributes.:
Explanation

The cube level does not satisfy the metadata object rule that a cube level must refer to exactly one level.

User Response

Alter the cube level so that it refers to exactly one level. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6404: The *level_name* cube level must refer to exactly one level.:
Explanation

The cube level does not satisfy the metadata object rule that all of the related attributes in a cube level must also be related attributes for the corresponding level.

User Response

Alter the cube level so it references attributes that are also referenced by the parent level. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6405: The *level_name* cube level cannot contain the *attribute_name* attribute more than once is the set of related attributes.:

Explanation

The cube level does not satisfy the metadata object rule that the set of related attributes cannot contain duplicate attributes.

User Response

Alter the cube level so that it does not contain duplicate related attributes. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

Referential constraint related errors

6500: This operation cannot be performed because the SQL template for the *attribute_name* attribute or measure still involves references to other attributes, measures, or columns. These references must be dropped prior to the execution of this operation.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The SQL expression template for the identified object involves references to other attributes, measures, or columns that must be removed from the identified object prior to the execution of this operation.

User Response

Before you drop the identified object, change the identified object so that its SQL expression template no longer references attributes, measures, or columns. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6501: The operation cannot be performed because the *attribute_name* attribute or measure is referenced by another attribute or measure.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute or measure is currently referenced by another attribute or measure, so the identified attribute or measure cannot be dropped.

User Response

Before you drop the identified attribute or measure, change the referencing objects so that they no longer reference the identified attribute or measure. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6502: The operation cannot be performed because the *dimension_name* dimension is referenced by an aggregation that is defined in a measure.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified dimension is currently referenced by an aggregation of a measure, so the identified dimension cannot be dropped.

User Response

Before you drop the identified dimension, change the referencing objects so that they no longer reference the identified dimension. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6503: The operation cannot be performed for the *object_name* **object. A cube hierarchy must reference attributes that are already referenced by the hierarchy that was used to derive the cube hierarchy.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The problem occurred because of one of the following situations:

- An attempt to remove an attribute from a hierarchy was made where the attribute being removed is still being used by a related cube hierarchy.
- An attempt to add an attribute to a cube hierarchy was made where the attribute being added is not already being used by a related hierarchy.

User Response

Perform one of the following actions:

- Remove attributes from cube hierarchies before removing the same attributes from related hierarchies.
- Add attributes to hierarchies before adding the same attributes to related cube hierarchies.

See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6504: The operation cannot be performed for the *object_name* **object. A cube hierarchy must reference attribute relationships that are already referenced by the hierarchy that was used to derive the cube hierarchy.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The problem occurred because of one of the following situations:

- An attempt to remove an attribute relationship from a hierarchy was made where the attribute relationship being removed is still being used by a related cube hierarchy.
- An attempt to add an attribute relationship to a cube hierarchy was made where the attribute relationship being added is not already being used by a related hierarchy.

User Response

Perform one of the following actions:

- Remove attribute relationships from cube hierarchies before removing the same attribute relationships from related hierarchies.
- Add attribute relationships to hierarchies before adding the same attribute relationships to related cube hierarchies.

See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6505: The operation cannot be performed because the *hierarchy_name* hierarchy is referenced by a cube hierarchy.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified hierarchy is currently referenced by a cube hierarchy, so the identified hierarchy cannot be dropped.

User Response

Before you drop the identified hierarchy, change the referencing objects so that they no longer reference the identified hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6506: The operation cannot be performed for the *object_name* object. A cube facts must reference measures that are already referenced by the facts that was used to derive the cube facts.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The problem occurred because of one of the following situations:

- An attempt to remove a measure from a facts was made, where the measure being removed is still being used by a related cube facts.
- An attempt to add a measure to a cube facts was made, where the measure being added is not already being used by a related facts.

User Response

Perform one of the following actions:

- Remove measures from cube facts before removing the same measures from related facts.
- Add measures to facts before adding the same measures to related cube facts.

See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6507: The operation cannot be performed because the *facts_name* facts object is referenced by a cube facts object.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified facts is currently referenced by a cube facts, so the identified facts cannot be dropped.

User Response

Before you drop the identified facts, change the referencing objects so that they no longer reference the identified facts. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6508: The operation cannot be performed because the *hierarchy_name* hierarchy is referenced by a dimension.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified hierarchy is currently referenced by a dimension, so the identified hierarchy cannot be dropped.

User Response

Before you drop the identified hierarchy, change the referencing objects so that they no longer reference the identified hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6509: The operation cannot be performed because the *join_name* join is referenced by a facts object.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified join is currently referenced by a facts, so the identified join cannot be dropped.

User Response

Before you drop the identified join, change the referencing objects so that they no longer reference the identified join. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6510: The operation cannot be performed because the *cube_dimension_name* cube dimension is referenced by a cube.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified cube dimension is currently referenced by a cube, so the identified cube dimension cannot be dropped.

User Response

Before you drop the identified cube dimension, change the referencing objects so that they no longer reference the identified cube dimension. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6511: The operation cannot be performed for the *object_name*object. The cube dimensions of a cube must be derived from the dimensions referenced by the cube model from which the cube was derived.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The problem occurred because of one of the following situations:

- An attempt to remove an dimension from a cube model was made where the dimension being removed is still being used by a related cube's cube dimension.

- An attempt to add a cube dimension to a cube was made, where the dimension for the cube dimension being added is not already being used by a related cube model.

User Response

Perform one of the following actions:

- Remove cube dimensions from cubes before removing related dimensions from related cube models.
- Add dimensions to cube models before adding related cube dimensions to related cubes.

See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6512: The operation cannot be performed because the dimension *dimension_name* **is referenced by a cube dimension.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified dimension is currently referenced by a cube dimension, so the identified dimension cannot be dropped.

User Response

Before you drop the identified dimension, change the referencing objects so that they no longer reference the identified dimension. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6513: The operation cannot be performed for the *object_name* **object. A cube dimension's cube hierarchy must be derived from the hierarchy referenced by the same dimension that was used to derive the cube dimension.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The problem occurred because of one of the following situations:

- An attempt to remove an hierarchy from a dimension was made, where the hierarchy being removed is still being used by a related cube dimension's cube hierarchy.
- An attempt to add a cube hierarchy to a cube dimension was made, where the hierarchy for the cube hierarchy being added is not already being used by a related dimension.

User Response

Perform one of the following actions:

- Remove cube hierarchies from cube dimensions before removing related hierarchies from related dimensions.
- Add hierarchies to dimensions before adding related cube hierarchies to related cube dimensions.

See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6514: The operation cannot be performed because the *cube_hierarchy_name* **cube hierarchy is referenced by a cube dimension.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified cube hierarchy is currently referenced by a cube dimension, so the identified cube hierarchy cannot be dropped.

User Response

Before you drop the identified cube hierarchy, change the referencing objects so that they no longer reference the identified cube hierarchy. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6515: The operation cannot be performed for the *object_name* **object. A cube dimension's cube hierarchy must be derived from the hierarchy referenced by the same dimension that was used to derive the cube dimension.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The problem occurred because of one of the following situations:

- An attempt to remove an hierarchy from a dimension was made, where the hierarchy being removed is still being used by a related cube dimension's cube hierarchy.
- An attempt to add a cube hierarchy to a cube dimension was made, where the hierarchy for the cube hierarchy being added is not already being used by a related dimension.

User Response

Perform one of the following actions:

- Remove cube hierarchies from cube dimensions before removing related hierarchies from related dimensions.
- Add hierarchies to dimensions before adding related cube hierarchies to related cube dimensions.

See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6516: The operation cannot be performed because the *join_name* **join is referenced by a dimension.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified join is currently referenced by a dimension, so the identified join cannot be dropped.

User Response

Before You drop the identified join, change the referencing objects so that they no longer reference the identified join. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6517: The operation cannot be performed because the *attribute_name* attribute is referenced by a dimension.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute is currently referenced by a dimension, so the identified attribute cannot be dropped.

User Response

Before you drop the identified attribute, change the referencing objects so that they no longer reference the identified attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6518: The operation cannot be performed because the *attribute_name* attribute is referenced by a hierarchy.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute is currently referenced by a hierarchy, so the identified attribute cannot be dropped.

User Response

Before you drop the identified attribute, change the referencing objects so that they no longer reference the identified attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6519: The operation cannot be performed because the *relationship_name* attribute relationship is referenced by a hierarchy.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute relationship is currently referenced by a hierarchy, so the identified attribute relationship cannot be dropped.

User Response

Before you drop the identified attribute relationship, change the referencing objects so that they no longer reference the identified attribute relationship. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6520: The operation cannot be performed because the *dimension_name* dimension is referenced by a cube model.:
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified dimension is currently referenced by a cube model, so the identified dimension relationship cannot be dropped.

User Response

Before you drop the identified dimension, change the referencing objects so that they no longer reference the identified dimension. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6521: The operation cannot be performed because the *join_name* **join is referenced by a cube model.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified join is currently referenced by a cube model, so the identified join cannot be dropped.

User Response

Before you drop the identified join, change the referencing objects so that they no longer reference the identified join. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6522: The operation cannot be performed because *object_name* **is referenced by a facts object.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified object is currently referenced by a facts, so the identified object cannot be dropped.

User Response

Before you drop the identified object, change the referencing objects so that they no longer reference the identified object. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6523: The operation cannot be performed because the *attribute_name* **left attribute is referenced by an attribute relationship.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute is currently referenced by an attribute relationship, so the identified attribute cannot be dropped.

User Response

Before you drop the identified attribute, change the referencing objects so that they no longer reference the identified attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6524: The operation cannot be performed because the *attribute_name* **right attribute is referenced by an attribute relationship.:**

Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute is currently referenced by an attribute relationship, so the identified attribute cannot be dropped.

User Response

Before you drop the identified attribute, change the referencing objects so that they no longer reference the identified attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6525: The operation cannot be performed because the *attribute_name* **right attribute is referenced by a join.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute is currently referenced by a join, so the identified attribute cannot be dropped.

User Response

Before you drop the identified attribute, change the referencing objects so that they no longer reference the identified attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6526: The operation cannot be performed because the*attribute_name* **left attribute is referenced by a join.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified attribute is currently referenced by a join, so the identified attribute cannot be dropped.

User Response

Before you drop the identified attribute, change the referencing objects so that they no longer reference the identified attribute. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6527: The operation cannot be performed because the *model_name* **cube model is referenced by a cube.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified cube model is currently referenced by a cube, so the identified cube model cannot be dropped.

User Response

Before you drop the identified cube model, change the referencing objects so that they no longer reference the identified cube model. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6528: The operation cannot be performed because the *cube_facts_name* **cube facts object is referenced by a cube.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified cube facts is currently referenced by a cube, so the identified cube facts cannot be dropped.

User Response

Before you drop the identified cube facts, change the referencing objects so that they no longer reference the identified cube facts. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules.

6529: The operation cannot be performed because the *facts_name* **facts object is referenced by a cube model.:**
Explanation

The requested operation cannot be performed because it will violate a referential constraint that exists between metadata objects in the metadata catalog. The identified facts is currently referenced by a cube model, so the identified facts cannot be dropped.

User Response

Before you drop the identified facts, change the referencing objects so that they no longer reference the identified facts. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules and referential constraints between metadata objects.

6530: The operation cannot be performed because the *level_name* **level is referenced by a dimension.:**
Explanation

The requested operation cannot be performed because the operation violates a referential constraint that exists between metadata objects within the metadata catalog. The level is currently referenced by a dimension, so you cannot drop the level at this time.

User Response

Before dropping the level, alter the referencing objects so that they no longer reference the level. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules and referential constraints between metadata objects.

6531: The operation cannot be performed because the *level_name* **level is referenced by a hierarchy.:**

Explanation

The requested operation cannot be performed because it violates a referential constraint that exists between metadata objects within the metadata catalog. The level is currently referenced by a hierarchy, so you cannot drop the level at this time.

User Response

Before dropping the level, alter the referencing objects so that they no longer reference the level. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules and referential constraints between metadata objects.

6532: The operation cannot be performed because the *level_name* **cube level is referenced by a cube hierarchy.:**
Explanation

The requested operation cannot be performed because it violates a referential constraint that exists between metadata objects within the metadata catalog. The cube level is currently referenced by a cube hierarchy, so you cannot drop the cube level at this time.

User Response

Before dropping the level, alter the referencing objects so that they no longer reference the cube level. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules and referential constraints between metadata objects.

6533: The operation cannot be performed for the *cube_name* **cube. The cube dimension and cube hierarchy must be derived from the cube. The cube hierarchy must belong to the cube dimension.:**
Explanation

The requested operation cannot be performed because it violates a referential constraint that exists between metadata objects within the metadata catalog. The cube dimension of the cube is currently referenced by an optimization slice's cube dimension and cube hierarchy, so you cannot drop the cube's optimization slice at this time.

User Response

Before dropping the cube's optimization slice, alter the referencing objects so that they no longer reference the cube's optimization slice. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules and referential constraints between metadata objects.

6534: The operation cannot be performed on the *level_name* **level. The related attribute cannot be removed from the level because a corresponding cube level references the related attribute.:**
Explanation

The requested operation cannot be performed because it violates a referential constraint that exists between metadata objects within the metadata catalog. You

cannot drop the related attribute from the level because the related attribute is referenced by the corresponding cube level.

User Response

To drop the related attribute, modify the referencing cube level object so that the cube level does not reference the level's related attribute that you want to drop. See the *DB2 Cube Views Guide and Reference* for more information about metadata rules and referential constraints between metadata objects.

Optimization

Optimization error codes

7001: There are no cubes defined for the *model_name* cube model.:
Explanation

There are no cubes defined for the cube model.

User Response

If you want to optimize for extract queries that read data from the cube model into a MOLAP cube, you must define cubes that represent your MOLAP cubes. You cannot optimize for extract queries without defining one or more cubes.

7002: The *model_name* cube model does not exist.:
Explanation

A cube model with the name that you specified is not defined.

User Response

Verify that the correct cube model and schema names are specified. Names and schemas are case sensitive. Use the OLAP Center to view the list of existing cube models.

7003: Table space *table_space_name* was not found.:
Explanation

A table space with this name is not defined.

User Response

Verify that the correct tablespace name is specified.

7004: The Optimization Advisor is unable to determine recommendations.:
Explanation

You specified a certain limit for how much disk space can be used for optimizing this cube model. The advisor could not produce recommendations that use less or the same specified amount of disk space.

User Response

Specify a larger disk space limit and run the Optimization Advisor wizard again.

7005: Table space *table_space_name* **cannot be used to store the summary tables.:**
Explanation

The table space does not have the correct data storage type that is required to store table data. The table space must be a REGULAR type tablespace. LONG, USER TEMPORARY, and SYSTEM TEMPORARY table spaces cannot be used to store summary tables.

User Response

Specify a REGULAR table space to store the summary tables.

7006: Table space *table_space_name* **cannot be used to store the indexes.:**
Explanation

The tablespace specified does not have the correct data storage type required to store index data. The tablespace must be a REGULAR or LONG type tablespace. USER TEMPORARY and SYSTEM TEMPORARY table spaces cannot be used to store the indexes.

User Response

Specify a REGULAR or LONG tablespace to store the indexes.

7007: Optimization validation of the *model_name* **cube model failed.:**
Explanation

The cube model and associated metadata objects violate one or more of the metadata object rules required for optimization. Optimization will not be performed.

User Response

Optimization cannot be performed unless the cube model and associated metadata objects conform to the metadata object rules for optimization. See the *Setup and User's Guide* for more information about optimization rules.

7008: The cube model does not have any dimensions that have optimizable hierarchies.:
Explanation

Optimization will not be performed because the Optimization Advisor cannot find dimensions with hierarchies that can be optimized.

User Response

Ensure that the cube model has at least one dimension that has a nonrecursive hierarchy.

7009: You cannot specify more than one MOLAP extract or hybrid extract, or both a MOLAP extract and a hybrid extract type of optimization slice for one cube. More than one extract type of optimization slice is specified for the

cube_name **cube.:**
Explanation

The specified cube is not optimized because more than one extract type of optimization slice is specified for the cube. You can specify only one MOLAP extract or Hybrid extract type of optimization slice per cube.

User Response

Make sure there is only one MOLAP extract or Hybrid extract type of optimization slice per cube.

7010: Drillthrough optimization slice must be defined at a cube level that is below the hybrid extract optimization slice in at least one cube dimension.:
Explanation

The drill through optimization slice must be defined at a cube level that is below the hybrid extract optimization slice in at least one cube dimension.

User Response

Make sure the drill through optimization slice is defined at a cube level that is below the hybrid extract optimization slice in at least one cube dimension.

7011: The Optimization Advisor was stopped and was unable to determine any recommendations within the allotted time.:
Explanation

The Optimization Advisor was stopped by the stop advise operation before determining recommendations.

User Response

Allow the Optimization Advisor to run longer so that it can determine recommendations.

7012: Invalid id value.:
Explanation

The specified id value was invalid.

7013: DB2 table sampling was attempted on a facts object that is based on a view, alias, nickname, or other database object that does not support sampling.:
Explanation

DB2 Cube Views attempted to sample data from a facts object that is based on a view, alias, or nickname. DB2 table sampling is not supported for views, aliases, and nicknames.

User Response

DB2 Cube Views can perform table sampling on facts objects that are defined only on tables, and are not defined on views, aliases, or nicknames. Turn off the sampling option for the Optimization Advisor.

7014: The Optimization Advisor cannot determine any recommendations.:

Explanation

The Optimization Advisor cannot determine any recommendations for the cube model with the parameters that you specified.

User Response

Check the informational and warning messages for more information on why the Optimization Advisor cannot recommend any summary tables.

Optimization warnings

7200: The recommended summary tables will use deferred refresh because the cube model contains one or more nondistributive measures.:
Explanation

The refresh immediate option was selected for the summary tables. However, summary tables cannot be refreshed immediately if there are nondistributive measures defined in the cube model. Distributive measures use simple aggregation functions such as SUM and COUNT that can be aggregated from any intermediate values. Nondistributive measures use more complex aggregation functions, such as STDDEV, and they must be aggregated from the base tables.

User Response

If it is not necessary to maintain the summary tables synchronously with the base tables, no action is required. If the summary tables must be maintained synchronously with the base tables, you need to change the metadata so that only distributive measures are defined.

7201: The *table_name* **recommended summary table will use deferred refresh because one or more nullable attributes were found as columns in the fullselect of this recommended summary table.:**
Explanation

The refresh immediate option was selected for the summary tables. However, the recommended summary table contains one or more attributes that are used as nullable columns in the summary table's fullselect. Using nullable columns in a summary table's fullselect can cause slow immediate refresh performance. The summary table was set to *refresh deferred* to avoid this performance problem.

User Response

To change the nullability of an attribute, you must change the attribute's SQL expression or change the nullability of the DB2 table columns used by the attribute or both. These changes are not usually recommended because they might be difficult to implement.

7202: The *table_name* **table does not have statistics.:**
Explanation

The Optimization Advisor cannot find valid table statistics values for the specified table.

User Response

Use the RUNSTATS command to create statistics for the specified table. Then run the Optimization Advisor wizard again.

7203: For the hybrid extract optimization slice *slice_name* **in the** *cube_name* **cube, you must specify a cube level for each cube dimension in the cube.:**
Explanation

The Optimization Advisor cannot optimize for the specified hybrid extract type of optimization slice because the optimization slice is not well-defined. The hybrid extract type of optimization slice must have a cube level defined for each cube dimension. You cannot select Not specified.

User Response

Make sure the optimization slice has a cube level specified for each cube dimension in the cube.

7203: For the hybrid extract optimization slice *slice_name* **in the** *cube_name* **cube, you must specify a cube level for each cube dimension in the cube:**
Explanation

The Optimization Advisor cannot optimize for the specified hybrid extract type of optimization slice because the optimization slice is not well-defined. The hybrid extract type of optimization slice must have a cube level defined for each cube dimension. You cannot select Not specified.

User Response

Make sure the optimization slice has a cube level specified for each cube dimension in the cube.

7205: The recommended summary tables must use the deferred refresh update method because one or more of the underlying base tables of the cube model is a nickname.:
Explanation

You selected the refresh immediate update option for the summary tables, but the summary tables cannot us the refresh immediate update method if any of the underlying tables are nicknames.

User Response

No action is required.

7206: The Optimization Advisor could not use data sampling to determine the recommended summary tables.:
Explanation

You specified that the Optimization Advisor should use data sampling to determine the optimal summary table recommendations. The Optimization Advisor cannot perform data sampling because of how the facts object is defined. If the facts object is defined as a nickname, the Optimization Advisor attempts to perform data sampling but might not be able to unless the cube model's dimensions have a moderate cardinality and the facts-to-dimension joins are on a

single attribute that maps to a single column. Indexes that are defined on the fact table columns that represent dimensional keys can also affect the Optimization Advisor's ability to perform data sampling for a facts object defined as a nickname. If the facts object is defined as a view, data sampling is not supported. If the facts object is defined as an alias, data sampling might not be possible depending on what the alias maps to. The impact of creating recommendations without performing data sampling is that the Optimization Advisor cannot recommend the optimal summary tables. If the fact object is a table, the Optimization Advisor can always perform data sampling.

User Response

You can use the recommendations as they are, or you can try to improve the recommendations by specifying a table instead of a view or an alias for the facts object, and running the Optimization Advisor again.

Optimization informational messages

7400: The summary tables are defined using the ROLLUP operator because the cube model contains one or more nondistributive measures.:
Explanation

Measures are either distributive or nondistributive. Distributive measures use simple aggregation functions such as SUM and COUNT that can be aggregated from any intermediate values. Nondistributive measures use more complex aggregation functions, such as STDDEV, and they must be aggregated from the base tables. To avoid the cost of aggregating nondistributive measures from the base tables, the summary tables are defined using the ROLLUP operator, which pre-aggregates the nondistributive measures.

User Response

No action is required.

7401: The *table_name* **summary table is recommended. It is estimated to have** *rows* **rows, a** *n* **MB table size, and a** *n* **MB index size.:**
Explanation

This message is a description of the recommended summary table, including the estimated row count, estimated disk space, and estimated disk space that is used for indexes.

User Response

No action is required.

7402: There are *n* **summary tables that do not fit in the specified disk space limit. They have a cumulative estimated size of** *n* **MB:**
Explanation

This message provides information about the recommended summary tables that do not fit in the disk space limit.

User Response

To view these summary tables in the recommendations, run the Optimization Advisor again with a larger specified disk space limit.

7403: The recommendations include optimizations for the *cube_name* **cube.:**
Explanation

Summary tables are recommended for the specified cube. Some queries for this cube will be optimized.

User Response

No action is required.

7404: The recommendations do not include optimizations for the *cube_name* **cube.:**
Explanation

Summary tables are not recommended specifically for this cube. Queries specific to this cube are not likely to show performance improvement.

User Response

If summary tables are not included in the recommendations because of a disk space limitation, run the Optimization Advisor again with a larger specified disk space limit. The recommendations might include one or more summary tables to optimize queries for this cube.

7405: The specified time limit ran out while the Optimization Advisor was determining the recommendations.:
Explanation

The Optimization Advisor made a recommendation. If more time is allowed, the Optimization Advisor might be able to make a better recommendation because it can perform additional analysis. Running the Optimization Advisor longer does not guarantee better results.

User Response

You can run the Optimization Advisor again with more time specified, or you can create the recommended summary tables and see if the performance is acceptable.

7406: The *dimension_name* **dimension does not have any hierarchies that can be optimized by the Optimization Advisor.:**
Explanation

The Optimization Advisor cannot optimize for recursive hierarchies. The specified dimension does not contain any hierarchies that can be optimized so the Optimization Advisor ignores this dimension. Queries that refer to attributes from this dimension are not optimized.

User Response

No action is required. Queries that use attributes from this dimension will not have any performance improvement.

7407: The recommended summary tables optimize for *n* percent of the slices in the cube model. Queries that run against the optimized slices should have improved performance.:
Explanation

SQL queries access particular slices in the cube model. One way to analyze performance improvement is to consider what portion of the slices that can be queried will be improved. If the cube model uses distributive measures, queries that access slices that are logically above the summary table slice will have improved performance.

For example, there are 30 possible slices in a cube model that has a Time dimension with the hierarchy [All-Year-Quarter-Month-Day] and a Region dimension with the hierarchy [All-Country-Region-State-City-Store]. You can calculate the number of possible slices by multiplying the number levels in the dimension hierarchies together. If the recommended summary table optimizes for the Month-City slice, then all slices at or above that slice are optimized. In this example, 20 of the 30 possible slices, or 67% (20/30) of the slices are optimized. There will never be 100% coverage because that will require duplicating the base tables in the summary tables. Typically, the lowest slices are less beneficial to optimize for because they are not very different from the base tables.

User Response

No action is required. If the percentage is low, you can run the Optimization Advisor wizard again with a larger specified disk space limit.

7408: Reading cube model metadata from the database.:
Explanation

The Optimization Advisor is reading the metadata that describes the cube model. The metadata contains information that significantly affects the optimization recommendations.

User Response

No action is required.

7409: Selecting which aggregations to include in the summary tables.:
Explanation

The Optimization Advisor is testing potential summary table configurations to determine which configuration is optimal for the specified criteria.

User Response

No action is required.

7410: Sampling data from cube model.:
Explanation

The Optimization Advisor is reading a subset of data from the fact and dimension tables so that it can estimate the size of the summary table. Sampling might occur multiple times as the Optimization Advisor considers potential summary tables.

User Response

No action is required.

7411: Defining indexes for recommended summary tables.:
Explanation

The Optimization Advisor determined the recommended summary tables and is selecting indexes to build for the summary tables.

User Response

No action is required.

7413: The *cube_name* **cube does not have any optimization slices specified so the Optimization Advisor optimizes the cube for drill-down types of queries.:**
Explanation

If there are no optimization slices defined for a cube, the Optimization Advisor optimizes the cube for drill-down types of queries.

7414: There is no cube defined for the cube model being optimized, so the Optimization Advisor optimizes the cube model as if it had a cube for report types of queries.:
Explanation

If there are no cubes defined for a cube model, the Optimization Advisor can optimize the cube model as if it had a cube that is used for drill-down types of queries. If you did not define any cubes for the cube model that you are optimizing, the Optimization Advisor makes recommendations as if you have a cube with a drill-down optimization slice defined at the All level for each cube dimension

7415: The Optimization Advisor did not recommend a summary table for optimization slice *slice_name* **in the** *cube_name* **cube.:**
Explanation

A summary table is not recommended for the specified optimization slice.

User Response

No action is required.

7416: Operation stopped. The recommendations might not be optimal.:
Explanation

The Optimization Advisor was stopped by the stop advise operation. The recommendations might not be optimal.

User Response

For optimal recommendations, run the Optimization Advisor again without limiting the time.

7417: The Optimization Advisor stopped creating recommendations because it reached the time limit that was set. The Optimization Advisor can recommend

better summary tables if you allow more time.:
Explanation

The time provided for the Optimization Advisor might be too short.
Recommended: Run the Optimization Advisor with a longer time limit.

User Response

Run the Optimization Advisor with a longer time limit. The Optimization Advisor might recommend better summary tables.

7418: The Optimization Advisor cannot recommend a summary table for the *cube_name* **cube, because the summary table has too many rows compared to the number rows in the fact table.:**
Explanation

The Optimization Advisor cannot recommend a summary table for the cube. The number of rows in the summary table is too high of a percentage of the number of rows in the fact table.

User Response

If you specified optimization slices, one or more of the optimization slices might have specified too many levels or levels that are too close to the grain of the fact table. You might specify Any level for some of the cube dimensions or specify levels that are closer to the top of the hierarchy. You can also allow the Optimization Advisor to run over a longer period of time to improve the recommendations.

OLAP Center messages

10000-10600

10000: OLAP Center is unable to retrieve any database names.
Explanation

An error occurred retrieving the list of database names from DB2.

User Response

Check that OLAP Center is installed correctly. If the problem persists, contact IBM Software Support.

10001: Type a user name.
Explanation

The **User name** field is empty.

User Response

Type a user name in the **User name** field.

10002: Type a password.

Explanation

The **Password** field is empty.

User Response

Type a password in the **Password** field.

10004: Cannot parse the attribute entered in the SQL expression.

Explanation

The SQL expression that was entered refers to an attribute that is neither valid in the given context nor present in the database.

User Response

Ensure that the SQL expression refers to only those attributes that appear in the **Data** section of the SQL Expression Builder.

10005: The metadata objects were successfully exported to the *file_name* file.

Explanation

Export was successful.

User Response

No action is required.

10006: Enter the file name to export the metadata objects to.

Explanation

The export utility writes the exported metadata objects in to the file name entered by the user.

User Response

Type a file name in the **File name** field.

10007: Select a cube or a cube model to export.

Explanation

The export window can export a cube or a cube model.

User Response

Select an object to export.

10008: Enter a unique name for the object that you are creating.

Explanation

In the SQL Expression Builder, the **Name** field of the attribute or measure being created is empty.

User Response

Type an object name in the **Name** field. The object name must be unique in the name space of the attributes and measures.

10009: Enter an SQL expression for the object.

Explanation

The SQL expression field of the object is empty.

User Response

Enter an SQL expression for the object.

10010: The *column_name* column is not qualified with a table name.

Explanation

The column name entered in the SQL expression is not qualified with a table name.

User Response

Ensure that the column name in the SQL expression is qualified with a table name using '.' as a separator.

10011: The *column_name* column is not qualified with a schema name.

Explanation

Column references in the SQL expression must be qualified with both a table name and a schema name separated by '.'.

User Response

Ensure that the column name in the SQL expression is qualified with both a table name and a schema name separated by '.'.

10012: The first element in an aggregation script cannot be a dimension.

Explanation

An aggregation script was created with a dimension as the first element.

User Response

Use an aggregation function as the first element in the aggregation script.

10013: Select an existing measure or enter an SQL expression as the second parameter for the *function_name* multiparameter function in the aggregation.

Explanation

When you use a multiparameter function in the aggregation script, the first parameter is defined as the measure that the aggregation is associated with. For the second parameter, select an existing measure or enter an SQL expression.

User Response

Enter a measure or an SQL expression as a second parameter.

10014: The *function_name* aggregation function has no matching dimensions.

Explanation

Each aggregation function in the aggregation script must be applied to at least one dimension.

User Response

Ensure that each aggregation function in the aggregation script is applied to at least one dimension.

10015: Closing parenthesis is missing for the *object_name* object.

Explanation

In OLAP Center, attributes, measures or columns specified in an expression should be enclosed by @Attribute(), @Measure() or @Column() respectively.

User Response

Enter a closing parenthesis for the object.

10016: Referring to the *object_name* object in the SQL expression creates an invalid reference loop.

Explanation

The object refers to itself in its SQL expression.

User Response

Ensure that the objects in the SQL expression do not create reference loops.

10017: No errors were found. The SQL expression is valid.

Explanation

The SQL expression is valid.

User Response

No action is required.

10018: No errors were found. The aggregation script is valid.

Explanation

The set of aggregations in the aggregation script is valid.

User Response

No action is required.

10020: Type a name.

Explanation

The **Name** field of the object is empty.

User Response

Type an object name in the **Name** field.

10021: Type a schema name.

Explanation

The **Schema** field is empty.

User Response

Type a schema name in the **Schema** field.

10022: Type a business name.

Explanation

The **Business name** field is empty.

User Response

The business name can be displayed in business intelligence applications to identify the object to an end user. Type a business name in the **Business name** field.

10023: Select at least one level to include in the cube hierarchy.

Explanation

No levels are specified in the cube hierarchy.

User Response

Select at least one level to include in the cube hierarchy.

10024: Select at least one measure for the cube facts.

Explanation

No measures are specified for the cube facts.

User Response

Select at least one measure for the cube facts.

10025: Select at least one level to include in the cube hierarchy.

Explanation

No attributes are specified in the cube hierarchy.

User Response

Select at least one attribute to include in the cube hierarchy.

10026: Select at least one dimension in the cube.

Explanation

No dimensions were specified in the cube.

User Response

Select at least one dimension and then click the [...] button to specify details for the cube dimension.

10027: One or more dimensions which currently exist in the cube have been cleared. Click Yes to delete the cube dimensions. Click No to return to the window, then click Cancel to close the window without saving the changes.

Explanation

One or more dimension selections have been cleared. The corresponding cube dimensions will be deleted from the cube.

User Response

Click **Yes** in the window to remove the cube dimensions from the cube. Click **No** to keep the cube dimensions and then click **Cancel** to close the window without saving.

10028: An attribute relationship cannot be defined with a many:many cardinality if the functional dependency check box is selected.

Explanation

An attribute relationship cannot be defined with both cardinality *many:many* and functional dependency selected.

User Response

Select a different cardinality for the attribute relationship or clear the functional dependency checkbox.

10029: Select a left attribute and a right attribute for the attribute relationship.

Explanation

An attribute relationship cannot be defined if both left and right attributes are not selected.

User Response

Select both left and right attributes.

10030: The same attribute cannot be selected for both the left and right attribute in an attribute relationship.

Explanation

An attribute relationship cannot be defined if the left and right attributes are identical.

User Response

Select different left and right attributes.

10031: An object with the name and schema specified already exists in the database. Type a different name.

Explanation

An object of the type being created or modified already exists in the database with the same name and schema specified.

User Response

Enter a unique name for the object.

10032: Specify at least one attribute pair.

Explanation

A join must have at least one attribute pair.

User Response

Specify at least one attribute pair.

10033: Duplicate attribute pairs cannot be created.

Explanation

An attribute pair already exists that matches the new selections.

User Response

Select different left and right attributes.

10034: Select at least one table.

Explanation

No tables have been selected.

User Response

Select at least one table in order to proceed.

10035: Select or create new joins to join all of your selected tables.

Explanation

No joins were selected.

User Response

Select or create new joins that will join all of your selected tables.

10036: Select at least one attribute.

Explanation

No attributes were selected.

User Response

Select at least one attribute.

10037: Select a join to join the dimension with the facts object.

Explanation

No joins were selected.

User Response

Select one join which will join your dimension with the facts object.

10038: Specify only one join between two given tables. The *join_name1* join and the *join_name2* join both join the same two tables.

Explanation

More than one join was selected for the same pair of tables.

User Response

Select only one join for each pair of tables.

10039: All selected tables must be joined. Select a join for the *table_name* table.

Explanation

All selected tables must be joined.

User Response

Select a join for the specified table.

10040: The number of selected tables does not correspond to the number of selected joins. Verify that there are no join loops and that all of the tables are joined.

Explanation

All selected tables must be joined.

User Response

Ensure that there are no join loops and that all of the tables are joined.

10042: Select at least one measure.

Explanation

No measures were specified.

User Response

Select at least one measure.

10043: Select a table column.

Explanation

A table column was not specified.

User Response

Select a column.

10044: Select an SQL expression.

Explanation

An SQL expression was not specified.

User Response

Click the **Build Expression** button to build your expression.

10045: An aggregation script was not specified.

Explanation

An aggregation script was not specified.

User Response

Click the **Build Script** button to build your aggregation script.

10046: Select a measure before opening the expression builder.

Explanation

A measure was not selected.

User Response

Select a measure from the table.

10047: Select a measure before opening the Aggregation Script Builder.

Explanation

A measure was not selected.

User Response

Select a measure.

10048: The Aggregation Script Builder cannot be launched for the *measure_name* measure because the *model_name* cube model does not have at least one dimension.

Explanation

An aggregation script cannot be specified if the cube model does not have at least one dimension.

User Response

Add dimensions to the cube model before specifying an aggregation script.

10049: To edit the expression, specify an attribute.

Explanation

An attribute is not selected.

User Response

Select an attribute.

10050: The metadata will be refreshed from the database. Any changes that were being made when the error occurred will be lost.

Explanation

An error occurred calling the DB2 stored procedure.

User Response

Click **OK** to refresh the metadata displayed byOLAP Center. Any changes that were being made when the error occurred will be lost. The objects displayed in OLAP Center will be refreshed with the corresponding objects in the database allowing the user to continue working.

10051: The *model_name* cube model cannot be validated for optimization. DB2 returned the following message:*message*.

Explanation

OLAP Center cannot start the Optimization Advisor wizard for the selected cube model because the selected cube model did not pass the validation that was performed by the stored procedure API.

User Response

Check the stored procedure API documentation for the cube model validation rule. Follow the instructions from the return message from DB2.

10052: Some of the loaded attributes or measures map to *column_names* columns that no longer exist in the database. Resolve the problem either by restoring the tables to which the columns belong or by dropping the invalid attributes or measures or both.

Explanation

This message appears when you start OLAP Center or after you click **View —> Refresh**. It appears because a table that the loaded attributes or measures map to was dropped or renamed.

User Response

Correct the problem in one of the following ways:

- Restore the table that was deleted or renamed.
- Map the attributes/measures to a table that does exist in the database.

- Drop the attribute/measures that map to the columns that do not exist.

10053: The *model_name* cube model optimization validation returned a warning. DB2 returned the following message:*message*

Explanation

OLAP Center attempted to validate the cube model before starting the Optimization Advisor and DB2 returned a warning. The warning might indicate that you have a cube model that cannot be optimized. For example, your cube model might contain views that reference tables that do not have constraints defined between them.

User Response

Check the message returned by DB2 and decide if you want to continue running the Optimization Advisor wizard.

10060: The cube model is not complete. Before a cube can be created, the cube model must contain a facts object, at least one dimension, and at least one hierarchy for each dimension.

Explanation

The cube model is not in a valid state for a cube to be created.

User Response

Modify the cube model so that it has a facts object and at least one dimension. Ensure that each dimension has at least one hierarchy.

10061: When a cube model is dropped, its dimensions are removed and its facts are dropped. The removed dimensions will continue to be available from the All Dimensions folder. Are you sure you want to drop the *model_name* cube model?

Explanation

Drop confirmation message.

User Response

Ensure the selected object is the one you want to drop and click **Yes**. If you do not want to drop the selected object, click **No**.

10062: When a dimension is dropped, its hierarchies and corresponding cube dimensions are also dropped. Are you sure you want to drop the *dimension_name* dimension?

Explanation

Drop confirmation message.

User Response

Ensure that the selected object is the one you want to drop and click **Yes**. If you do not want to drop the selected object, click **No**.

10063: When a cube is dropped, its cube dimensions, cube hierarchies, and cube facts are also dropped. Are you sure you want to drop the *cube_name* cube?

Explanation

Drop confirmation message.

User Response

Ensure the selected object is the one you want to drop and click **Yes**. If you do not want to drop the selected object, click **No**.

10064: When a cube dimension is dropped, its cube hierarchies are also dropped. Are you sure you want to drop the *cube_dimension_name* cube dimension?

Explanation

Drop confirmation message.

User Response

Ensure the selected object is the one you want to drop and click **Yes**. If you do not want to drop the selected object, click **No**.

10065: Are you sure you want to drop *object_name*?

Explanation

Drop confirmation message.

User Response

Ensure the selected object is the one you want to drop and click **Yes**. If you do not want to drop the selected object, click **No**.

10066: When a dimension is removed, all of the corresponding cube dimensions are removed from their cubes. Are you sure you want to remove the *dimension_name* dimension from *object_name*?

Explanation

Remove dimension confirmation message.

User Response

Ensure that the selected object is the one that you want to remove and click **Yes**. If you do not want to remove the selected object, click **No**.

10067: The file with the name *file_name* already exists. Do you want to overwrite its contents?

Explanation

Overwrite file confirmation message.

User Response

Ensure that you want to overwrite the contents of the file name that you entered.

10068: Unable to determine the data type for the object with the *object_name* name and *schema_name* schema. The database returned the following information: *message*.

Explanation

For the specified object, OLAP Center cannot determine the source data type or aggregated data type.

User Response

Ensure that the SQL expression for the specified object is correct. If you cannot resolve the problem, contact IBM Software Support.

10069: Unable to determine the source data type for the measure with the *measure_name* name and *schema_name* schema.

Explanation

For the specified measure, OLAP Center cannot determine the source data type because the specified measure has an invalid source expression. A measure can have an invalid source expression when the None aggregation setting is applied to it because the measure is validated with aggregations of referenced measures rather than as a self-contained expression.

User Response

You can perform one of the following actions:

- Alter the source expression of the specified measure so that it validates correctly with the None aggregation setting.
- Do not use the specified measure in your expression.

10070: When a facts object is dropped, its measures are also dropped. Are you sure you want to drop the *facts_name* facts?

Explanation

Drop confirmation message.

User Response

Ensure the selected object is the one you want to drop and click **Yes**. If you do not want to drop the selected object, click **No**.

10071: All the selected objects will be dropped from the database. Do you want to drop these objects?

Explanation

More than one object was selected and the drop option was selected.

User Response

Ensure that the selected objects are the ones that you want to drop and click **Yes**. If you do not want to drop the selected objects, click **No**.

10072: Some of the selected objects cannot be dropped. These objects remain in the database.

Explanation

OLAP Center cannot drop all of the selected objects. This is probably because some of the selected objects are referenced by other objects in the database and dropping the selected object makes the referencing object invalid.

User Response

No action is required.

10073: None of the selected objects can be dropped.

Explanation

OLAP Center cannot drop any of the selected objects. This is probably because the selected objects are referenced by other objects in the database and dropping the selected objects makes the referencing objects invalid.

User Response

No action is required.

10074: Are you sure you want to remove *object_name* from *directory_name*?

Explanation

Remove confirmation message.

User Response

Ensure the selected object is the one you want to remove and click **Yes**. If you do not want to remove the selected object, click **No**.

10075: All the selected objects will be removed from *directory_name*. Do you want to remove these objects?

Explanation

More than one object was selected and the remove option was selected.

User Response

Ensure that the selected objects are the ones you want to remove and click **Yes**. If you do not want to remove the selected objects, click **No**.

10076: Some of the selected objects cannot be removed. These objects remain in the database.

Explanation

OLAP Center cannot remove all of the selected objects. This is probably because some of the selected objects are referenced by other objects in the database and removing the selected object makes the referencing object invalid. It could also be because the parent object needs at least one child object, for example, in the case of cube hierarchy

User Response

No action is required.

10077: None of the selected objects can be removed.

Explanation

OLAP Center cannot remove any of the selected objects. This is probably because the selected objects are referenced by other objects in the database and removing the selected objects makes the referencing object invalid.

User Response

No action is required.

10078: The operation succeeded. The following information messages were returned by the database: *database_name*.

Explanation

The database operation succeeded but some information messages were returned.

User Response

Examine the information messages and decide whether any further action needs to be taken.

10080: Object of type *type* not found during the second pass of XML.

Explanation

An object referenced in the XML being read could not be located.

User Response

Ensure that the XML file being imported is correctly formed. If this error occurs while starting OLAP Center, contact IBM Software Support.

10081: The system failed to parse XML from the file *file_name*. The error occurred at line *line_number* character *character_number*. The parser returned the following information: information.

Explanation

A parsing error occurred while attempting to import an XML file

User Response

Ensure that the XML file being imported is correctly formed and is a valid Cube Views XML metadata file. Check the line and character number to find the error.

10082: An unexpected parser exception was encountered in the file *file_name*. The following information was returned: information.

Explanation

An unexpected parsing error occurred while attempting to import an XML file.

User Response

Ensure that the XML file being imported is correctly formed and is a valid Cube Views XML metadata file. Check the line and character number to find the error.

10084: An object with the name *object_name* in schema *schema_name* already exists. The object cannot be created. Type a unique name, schema, or both for the new object.

Explanation

OLAP Center attempted to create a new object, but an object of this type with the same name and schema already exists.

User Response

Enter a different name, schema, or both for the object being created.

10085: An object of name *object_name* in schema *schema_name* already exists. The object cannot be renamed. Type a unique name, schema, or both for the object being renamed.

Explanation

OLAP Center attempted to rename an object, but an object of this type with the same name and schema already exists.

User Response

Enter a different name, schema, or both for the object being renamed.

10086: A database connection could not be made. DB2 returned: *message*.

Explanation

OLAP Center could not connect to the database. Some error information provided by DB2 is included in the message.

User Response

Read the text returned by DB2 and correct the problem.

10087: The *object_name1* metadata object cannot be dropped because it is referred to by the *object_name2* object of type *type*.

Explanation

The selected metadata object cannot be dropped because it is in use by at least one other metadata object.

User Response

Remove the object from any other metadata objects it is a part of, then try dropping the object again.

10088: An error occurred registering the DB2 driver with the JDBC Driver Manager. A database connection could not be established. The following information was returned: *message*.

Explanation

Before connecting to a DB2 database, OLAP Center has to register the JDBC driver that it will use with the Driver Manager. An error occurred during the JDBC driver registration.

User Response

Check the DB2 installation to make sure that the `db2java.zip` and `db2jcc.jar` files are installed. Ensure that Java and any JDBC components are installed correctly. Read the information returned in the message to help resolve the problem.

10089: An error occurred while accessing the database. The database returned the following information: \n SQL State: *message*\n SQL Error Code: *code*\n SQL Message: *SQL_message*.

Explanation

OLAP Center application called DB2 using the API stored procedure. The execute command threw an SQLException that could not be handled by OLAP Center.

User Response

Use the additional error information provided in the message to resolve the problem. If you cannot resolve the problem, contact IBM Software Support.

10090: Executing the DB2 stored procedure caused a false return code. No error information was found in the returned XML document. Contact IBM Software Support.

Explanation

The OLAP Center application called DB2 using the API stored procedure. The execute command returned *false*, but there was no error information in the XML document returned by the stored procedure.

User Response

It is possible that the operation completed successfully, but you should report this problem to IBM Software Support.

10091: An error occurred while processing a database API call. The following information was returned: \n SQL State:*message*\n SQL Error Code: *code*\n Operation:*operation*\n Status ID: *ID*\n Status Text: *text*.

Explanation

The OLAP Center stored procedure API call had an error while executing some OLAP Center changes.

User Response

See the information contained in the message. If the problem cannot be resolved, contact IBM Software Support.

10092: An error occurred while parsing the XML returned by the database API call. The following information was returned: *message*.

Explanation

The OLAP Center stored procedure API call returned XML that was incomplete or badly formed. OLAP Center could not read the returned XML.

User Response

Use the information that is contained in the message to resolve the problem. If the problem cannot be resolved, contact IBM Software Support.

10093: The *file_name* file does not exist.

Explanation

The specified file does not exist.

User Response

Specify a file that exists.

10094: An I/O error occurred reading the *file_name* file. The following system information was returned: *message*.

Explanation

An I/O error occurred while reading from a file.

User Response

Check the system information to try to resolve the problem or specify a different file.

10095: An I/O error occurred writing to the *file_name* file. The following system information was returned: *message*.

Explanation

An I/O error occurred while writing to a file.

User Response

Check the system information to try to resolve the problem or specify a different file.

10096: A query to retrieve the database schema failed. The database returned the following information: *message*.

Explanation

A query to retrieve the database schema failed.

User Response

Check the database information to resolve the problem.

10097: A query to retrieve a schema's tables failed. The database returned the following information: *message*.

Explanation

A query to retrieve a schema's tables failed.

User Response

Check the database information to resolve the problem.

10098: A query to retrieve a table's columns failed. The database returned the following information: *message*.

Explanation

A query to retrieve a table's columns failed.

User Response

Check the database information to resolve the problem.

10099: A commit of a DB2 connection failed. The database returned the following information: *message*.

Explanation

A commit of a DB2 connection failed.

User Response

Check the database information to resolve the problem.

10100: A rollback of a DB2 connection failed. The database returned the following information: *message*.

Explanation

A rollback of a DB2 connection failed.

User Response

Check the database information to resolve the problem.

10101: *Object_name* cannot be dropped because it is the last cube dimension in the *cube_name* cube. A cube must have at least one cube dimension to be valid.

Explanation

OLAP Center attempted to drop the last cube dimension in a cube.

User Response

A cube must have at least one cube dimension to be valid. Do not attempt to drop the last cube dimension from a cube.

10102: Object *object_name1* of type *type1* refers to object *object_name2* or type *type2* which could not be found.

Explanation

An object within the XML file being read refers to an object which cannot be found. If the error occurs during import, the object being referred to might not exist within the file being imported.

User Response

If an import is being performed, ensure the file contains all of the objects needed for the import to succeed. If the error occurs while starting OLAP Center, contact IBM Software Support.

10103: The measure cannot be dropped because the facts object must contain at least one measure.

Explanation

The measure cannot be dropped because the facts object must contain at least one measure.

User Response

No action is required.

10104: The measure cannot be removed because the cube facts must contain at least one measure.

Explanation

The measure cannot be removed because the cube facts must contain at least one measure.

User Response

No action is required.

10105: The cube level cannot be removed because the cube hierarchy must contain at least one cube level.

Explanation

The cube level cannot be removed because the cube hierarchy object must contain at least one cube level.

User Response

No action is required.

10106: The level cannot be removed because it is referenced by a cube level in the *hierarchy_name* cube hierarchy which is associated with the *hierarchy_name* hierarchy.

Explanation

The level cannot be removed because it is referenced by a cube level within a cube hierarchy which is associated with the hierarchy from which the attribute is being removed.

User Response

No action is required.

10107: The attribute cannot be removed because it is referenced by the *level_name* cube level which is associated with the *level_name* level.

Explanation

The attribute cannot be removed because it is referenced by a cube level which is associated with the level from which the attribute is being removed.

User Response

No action is required.

10108: The attribute cannot be removed because it is referenced by the *hierarchy_name* hierarchy which is associated with the *dimension_name* dimension.

Explanation

The attribute cannot be removed because it is referenced by a hierarchy which is associated with the dimension from which the attribute is being removed.

User Response

No action is required.

10109: The attribute cannot be removed because it is referenced by the *join_name* join which is associated with the *dimension_name* dimension.

Explanation

The attribute cannot be removed because it is referenced by a join which is associated with the dimension from which the attribute is being removed.

User Response

No action is required.

10110: The level cannot be moved because it is referenced by a cube level in the *hierarchy_name* cube hierarchy associated with the *hierarchy_name* hierarchy.

Explanation

The level cannot be moved up or down because it is referenced by a cube level within a cube hierarchy which is associated with the hierarchy being edited.

User Response

No action is required.

10111: The attribute cannot be removed because it is the only level key attribute of the *level_name* level.

Explanation

The attribute cannot be removed because it is the only level key attributes in the level from which it is being removed and the level must have at least one level key attribute.

User Response

No action is required.

10112: The attribute cannot be removed because it is the default attribute of the *level_name* level.

Explanation

The attribute cannot be removed because it is the default attribute of the level from which the attribute is being removed.

User Response

No action is required.

10113: The attribute cannot be removed because it is referenced by the *level_name* level which is associated with the *dimension_name* dimension.

Explanation

The attribute cannot be removed because it is referenced by a level which is associated with the dimension from which the attribute is being removed.

User Response

No action is required.

10114: The level cannot be removed because it is the only level in the *hierarchy_name* hierarchy.

Explanation

The level cannot be removed because it is the only level in the hierarchy and the hierarchy must contain at least one level.

User Response

No action is required.

10200: The file being imported does not have a UTF-8 encoding. Select a file with UTF-8 encoding.

Explanation

OLAP Center can import files only in the UTF-8 encoding.

User Response

Import a file with the supported encoding.

10201: Enter a file name for the SQL script used to refresh summary tables.

Explanation

The Optimization Advisor wizard creates an SQL script to refresh summary tables when the Deferred update option is selected. This script should be saved in a file and run to refresh the summary tables.

User Response

Enter a file name to save the SQL script to.

10202: Enter a file name for the SQL script used to create summary tables.

Explanation

The Optimization Advisor wizard generates an SQL script to create summary tables. This script should be saved in a file and run to create the summary tables.

User Response

Enter a file name to save the SQL script to.

10203: The selected measure cannot have None as its aggregation setting. Only calculated measures which refer exclusively to other measures in their expressions can specify the None aggregation setting.

Explanation

The None aggregation setting can only be selected for measures that only use expressions which refer exclusively to other measures.

User Response

Select a different aggregation.

10204: No dimensions exist. Create a new dimension to add to the cube model.

Explanation

No dimensions exist. Create a new dimension to add to the cube model.

User Response

Create a new dimension, instead of adding a dimension.

10205: There are no dimensions to add because all of the existing dimensions are already included in the cube model.

Explanation

All existing dimensions have been added to the cube model.

User Response

No action is required.

10206: You have changed your selected options. To see new recommendations from the Optimization Advisor wizard, you must run the Optimization Advisor wizard process again. If you do not run the Optimization Advisor wizard process again, you will see the recommendations created for the earlier options. Do you want to run the Optimization Advisor wizard process again?

Explanation

You have changed the selected options after running the Optimization Advisor wizard process. To view updated recommendations for the summary tables, run the Optimization Advisor wizard process again. If you do not run the Optimization Advisor wizard process again, you will see the recommendations created for the earlier options.

User Response

Click **Yes** to run the Optimization Advisor wizard process. Click **No** if you do not want to run the Optimization Advisor wizard process again.

10207: No dimension table has been detected.

Explanation

No dimension table was detected.

User Response

Ensure that referential integrity constraints are correctly set.

10208: *Object_names* objects that OLAP Center cannot directly display exist in the database. These objects might cause future problems with OLAP Center. Click Yes to drop the objects or click No to keep the objects in the database.

Explanation

OLAP Center detected a number of objects (such as hierarchies or facts) in the database that it cannot display directly. These objects might be preexisting or might be created after importing metadata. These objects might cause name clash and reference problems in OLAP Center in the future. Unless you have a good reason to keep these objects, it is recommended that you choose to drop them.

User Response

Click **Yes** to drop the objects or click **No** to keep the objects in the database.

10209: Unexpected error occurred during the import operation. Check the input XML file for errors.

Explanation

During import, the stored procedure API returned a warning with nothing in the output XML.

User Response

Ensure that the input XML metadata complies with the format defined in the OLAP metadata schema and the XML file defines all the metadata objects referred in it.

10210: Import operation failed. The stored procedure API returned the following message: *message*.

Explanation

During import process, the stored procedure API returned an error message.

User Response

Resolve the problem using the information provided in the message. If the problem cannot be resolved, contact IBM Software Support.

10211: The nonnumeric measure *measure_name* cannot use the *function_name* aggregation function because that function expects a numeric argument.

Explanation

Measures with nonnumeric data types cannot have numeric aggregation functions. You can only only select **MIN**, **MAX**, or **COUNT** as aggregation functions for nonnumeric data.

User Response

Choose a different aggregation function.

10212: Unable to read the objects from the input XML file. Check the input XML file for errors.

Explanation

OLAP Center failed to read the objects from the input XML file.

User Response

Ensure that the input XML metadata complies with the format defined in the OLAP metadata schema and the XML file defines all of the metadata objects referred in it.

10213: The *file_name* input XML file does not exist in the specified directory.

Explanation

The input XML file does not exist in the specified directory.

User Response

Ensure that the input XML file exists in the specified directory.

10214: The *object_name* object contained in the import file refers to the *column_name* column that does not exist in the database. Ensure that the tables and columns referred to by the metadata objects in the import file exist before importing the file.

Explanation

The import XML file contains objects that refer to tables and columns that do not exist in the database.

User Response

Ensure that the tables referred to by the objects in the import XML file exist in the database before importing the file.

10215: OLAP Center cannot run the SQL script recommended by the Optimization Advisor wizard. The database returned the following information: *message*.

Explanation

OLAP Center cannot execute the SQL script recommended by the Optimization Advisor wizard. You might not have sufficient privileges to execute the SQL script.

User Response

Ensure that you have the correct authorities to run the Optimization Advisor recommendations. The required authorities are described in the topic on "Authorities and privileges" in the OLAP Center online help. See the *DB2 Cube Views Setup and User's Guid*e for information on optimizing a cube model.

10216: The recommendations from the Optimization Advisor were successfully saved in the specified file(s).

Explanation

The recommended create summary tables SQL script and if applicable, the refresh summary tables SQL script, were saved into the specified files.

User Response

No action is required.

10217: The summary tables and their indexes were created successfully.

Explanation

The summary tables and indexes recommended by the Optimization Advisor were successfully created in the database.

User Response

No action is required.

10218: You have selected a view. The Optimization Advisor cannot verify that referential constraints exist for tables referenced by your view.

Explanation

Optimization might not be effective when you create summary tables for cube models using views that reference tables without constraints. The Optimization Advisor cannot detect if constraints exist on the tables referenced by the view.

User Response

If the tables referenced by your view do not have constraints and you want to run the Optimization Advisor you can either: 1. Not use the view in your cube model. 2. Create constraints for the tables before running the Optimization Advisor.

10219: Cancelling the Optimization Advisor wizard discards the recommended SQL scripts without saving them. Click Yes to close the Optimization Advisor wizard without saving the recommended SQL scripts. Click No to return to the Optimization Advisor wizard so that you can save the recommended SQL scripts.

Explanation

Clicking the **Cancel** button on the Optimization Advisor wizard discards the recommended SQL scripts without saving them.

User Response

Click **Yes** to close the Optimization Advisor wizard without saving the recommended SQL scripts or click **No** to return to the Optimization Advisor wizard so that you can continue using the Optimization Advisor wizard and save the SQL scripts.

10220: The specified file is in a format that is for an old version of DB2 Cube Views. The Import wizard can convert the metadata objects described by the specified file to the current version of DB2 Cube Views. Click Yes to convert the metadata objects described by the specified file and continue importing. Click No to stop the import so that you can specify a different file or close the Import wizard.

Explanation

The specified metadata source file contains XML in a format that is for an older version of DB2 Cube Views. The Import wizard can read the file and then convert its contents to a format that is understood by the current version of DB2 Cube Views.

User Response

Click **Yes** to convert the metadata objects described by the specified source file to the current DB2 Cube Views format. The Import Options page shows the objects from the specified file converted to the new version of DB2 Cube Views. Click **No** to stop the Import wizard from converting the metadata objects described by the specified file. You can specify a different meatadata source file or close the Import wizard.

10221: The recommended summary tables are expected to use *disk_space_size* MB of disk space.

Explanation

The summary tables recommended by the Optimization Advisor are expected to use the specified disk space.

User Response

Ensure you have at least the specified disk space available before running the recommended scripts.

10222: Do you want to stop the Optimization Advisor? Click Yes to stop the Optimization Advisor from creating recommendations and return the recommendations that it obtained so far. Click No to allow the Optimization Advisor to continue creating recommendations.

Explanation

You clicked **Stop** while the Optimization Advisor was creating optimization recommendations.

User Response

Click **Yes** to stop the Optimization Advisor from creating recommendations and return the recommendations that it has obtained so far. Click **No** to allow the Optimization Advisor to continue creating recommendations. If you click **Yes** the advisor wizard will display the recommendations it has obtained so far.

10300: Failed to parse the *measure_name* measure entered in the SQL expression.

Explanation

The SQL expression specified refers to a measure that is either invalid in the given context or is not present in the database.

User Response

Ensure that the SQL expression refers to only those measures that appear in the Data list of the SQL Expression Builder.

10301: Failed to parse the *column_name* column entered in the SQL expression.

Explanation

The SQL expression specified refers to a column that is either invalid in the given context or is not present in the database.

User Response

Ensure that the SQL expression refers to only those columns that appear in the Data list of the SQL Expression Builder.

10302: The *attribute_name* attribute is not qualified with a schema name.

Explanation

References to attributes in the SQL expression must be qualified with a schema name separated by a '.'.

User Response

Ensure that all of the references to attributes in the SQL expression are qualified with a schema name separated by '.'.

10303: The *measure_name* measure is not qualified with a schema name.

Explanation

References to measures in the SQL expression must be qualified with a schema name separated by a '.'.

User Response

Ensure that all the references to measures in the SQL expression are qualified with a schema name separated by '.'.

10304: Missing object name inside the *object_name* object tag.

Explanation

The SQL expression specified has an empty column tag @Column or an empty attribute tag @Attribute or an empty measure tag @Measure.

User Response

Ensure that the object type tags @Column, @Measure and @Attribute have an enclosing object name.

10305: The expression specified is invalid. The database returned the following information: *message*.

Explanation

There is a syntax error in the SQL expression. This error is also displayed when the SQL expression references columns, attributes, or measures without enclosing tags. A reference to a column, attribute or measure must be enclosed inside @Column(), @Attribute() or @Measure() tags respectively.

User Response

Correct the syntax error. Ensure that each column, attribute and measure is enclosed in the appropriate tag.

10306: The data type of the expression that you have entered is nonnumeric. Enter a numeric expression as a second parameter.

Explanation

The data type of the second parameter must be numeric.

User Response

Ensure that the data type of the expression entered results in to a numeric data type.

10307: The expression of the measure *measure_name* results to a nonnumeric data type. Select a measure whose expression results to a numeric data type.

Explanation

The data type of the second parameter must be numeric.

User Response

Ensure that the data type of the selected measure's expression is a numeric.

10308: OLAP Center cannot communicate with the specified database. This might be because the database is not correctly configured for DB2 Cube Views. Configuring the database might take some time. Click Yes to configure the specified database. Click No if you do not want to configure the specified database now.

Explanation

OLAP center can connect to the database using the username and password supplied, but it cannot communicate with the stored procedure API.

This might be because:

- The DB2 Cube Views stored procedure API is not registered for the specified database.
- The DB2 Cube Views catalog tables do not exist for the specified database.

User Response

Click **Yes** to configure the database for DB2 Cube Views otherwise click **No**.

10309: OLAP Center cannot connect to the specified database because the database is configured for an older version of DB2 Cube Views. The database must be migrated to the current version of DB2 Cube Views. Click Yes to have OLAP Center migrate the specified database. Click No if you do not want OLAP Center to migrate the specified database.

Explanation

OLAP center can connect to the database using the username and password supplied, but it cannot retrieve metadata from catalog. This might be because the DB2 Cube Views catalog tables are configured for an older version of DB2 Cube Views.

User Response

Click **Yes** to migrate the DB2 Cube Views catalog to proper version otherwise click **No**.

10310: The *database_name* database was successfully configured.

Explanation

OLAP Center successfully created the DB2 Cube Views catalog tables and registered the stored procedure API for the specified database.

User Response

No action is required.

10311: OLAP Center cannot configure the database for DB2 Cube Views. The database returned the following information: *information*.

Explanation

OLAP Center cannot configure the specified database for DB2 Cube Views.

This might be because:

- OLAP Center cannot register the DB2 Cube Views stored procedure API.
- OLAP Center cannot create one or more DB2 Cube Views catalog tables.

User Response

Ensure that you have the correct setup and install authorities that are described in the topic on ″Authorities and privileges″ in the OLAP Center online help. See the DB2 Cube Views Setup and User's Guide for information on configuring a database.

10312: The aggregation validation failed. One or more specified aggregation functions are not compatible with the source SQL Expression.

Explanation

One or more specified aggregation functions are not compatible with the source SQL Expression. This might be because the specified aggregation function expects a parameter with a data type that is different from the source SQL expression data type.

User Response

Ensure that the aggregation function is valid for the specified measure's source data type.

10313: The measure's source expression is syntactically correct only with the None aggregation setting. The measure must use the None aggregation setting.

Explanation

The measure expects the None aggregation setting when:

- The SQL Expression is syntactically incorrect when the aggregation functions are not applied to its referred measures but it is syntactically correct when those aggregation functions are applied. For example, char + int is syntactically incorrect but COUNT(char) + SUM(int) is syntactically correct.
- The SQL Expression uses OLAP functions like RANK(), DENSE_RANK() and ROW_NUMBER().

User Response

Ensure that the measure has None aggregation setting applied to it.

10401: The expression cannot include a column function, a scalar fullselect, or a subquery.

Explanation

The SQL expression cannot include a column function, scalar fullselect or a subquery.

User Response

Correct the use of the column function to eliminate the invalid expression.

10501: The schema name cannot start with *prefix*.

Explanation

The schema name cannot start with 'SYS' and 'SESSION'.

User Response

Type different schema name.

10502: The join properties are not valid for cube model performance optimization. Resolve this problem then run the Optimization Advisor wizard again. The database returned the following information: *message*.

Explanation

The join properties are invalid for cube model performance optimization.

User Response

Specify correct settings for your join by applying the optimization validation rules.

10503: The hierarchy cannot be modified because it has an associated cube hierarchy.

Explanation

If a cube hierarchy exists for the hierarchy, the hierarchy cannot be modified.

User Response

Ensure that no cube hierarchy references the hierarchy being modified before making changes to the hierarchy. You can also create a different hierarchy with the required modifications.

10504: This measure must use the None aggregation setting because it refers to the measure that uses a multiparameter aggregation function.

Explanation

Only measures that use the None aggregation setting can refer to measures that use a multiparameter function. You cannot change the aggregation setting from None to another function.

User Response

You can perform one of the following actions:

- Do not alter the measure's aggregation setting.
- Alter the specified measure so that it does not use a multiparameter function.

10505: This measure cannot use a multiparameter function because the *measure_name* measure that uses an aggregation setting other than None, refers to this measures.

Explanation

Only measures that use the None aggregation setting can refer to measures that use a multiparameter function. You cannot change the aggregation script of the measure being edited to include a multiparameter function because the measure being edited is referred to by a measure that does not use the None aggregation setting.

User Response

You can perform one of the following actions:

- Do not alter the measure's aggregation script.
- Alter the specified measure so that it does not refer to the measure being edited.

10506: The existing aggregation setting is invalid with the specified SQL expression. OLAP Center will reset the aggregation setting to *setting*.

Explanation

The existing aggregation setting is invalid with the new SQL expression and was reset to the default aggregation setting. This might be because:

- The data type of the source SQL expression changed.
- The current aggregation setting is expected to be None. It must be None when:
  - The SQL expression is syntactically incorrect when the aggregation functions are not applied to its referred measures, but it is syntactically correct when those aggregation functions are applied. For example, char + int is syntactically incorrect, but COUNT(char) + SUM(int) is syntactically correct.
  - The SQL Expression uses OLAP functions like RANK(), DENSE_RANK() and ROW_NUMBER().

User Response

No action is required.

10507: One or more dimensions in the cube model do not have a hierarchy. They will not be available for inclusion in the cube.

Explanation

For a cube dimension to be created it must be based on a dimension that has at least one hierarchy. You are attempting to create or modify a cube that has one or more dimensions that do not have a hierarchy. These dimensions will be omitted from the selection list used for defining cube dimensions.

User Response

Either create or modify your cube without references to the omitted dimensions or ensure that each dimension in the cube model has a hierarchy.

10508: The recursive deployment option is valid only if two levels are selected for the hierarchy.

Explanation

Exactly two level must exist for a recursive deployment.

User Response

Choose either exactly two level for recursive deployment or select another deployment option.

10509: You did not specify all of the required properties for the optimization slices. For each optimization slice, specify a query type and an option for each cube dimension.

Explanation

You did not specify all of the required properties for the optimization slices.

User Response

For each optimization slice, specify a query type and an option for each cube dimension.

10510: Select at least one level key attribute for the level.

Explanation

No level key attributes are specified for the level.

User Response

Select at least one level key attribute for the level.

10511: You cannot specify both a MOLAP extract type of slice and a hybrid extract type of slice for the same cube. You must remove one of the slices from the cube.

Explanation

One cube cannot have both a MOLAP extract and a hybrid extract type of optimization slice. You can have only one type of extract optimization slice specified per cube.

User Response

Remove either the MOLAP extract slice or the hybrid extract slice from the cube.

10512: A maximum of one MOLAP extract type of optimization slice is allowed per cube.

Explanation

A maximum of one MOLAP extract type of optimization slice is allowed per cube.

User Response

Make sure that you do not have more than one MOLAP extract type of optimization slice.

10513: A maximum of one hybrid extract optimization slice is allowed per cube.

Explanation

A maximum of one hybrid extract optimization slice is allowed per cube.

User Response

Make sure that you do not have more than one hybrid extract type of optimization slice.

10514: You can create a drill through type of optimization slice only if a hybrid extract type of optimization slice is already defined for the cube.

Explanation

A hybrid extract optimization slice is required to specify a drill through type of optimization slice.

User Response

Create a hybrid extract type of optimization slice. Then you can create a drill through type of optimization slice.

10515: DB2 Cube Views specified ANY as the optimization levels for the new cube dimensions for each of the existing optimization slices.

Explanation

If you add cube dimensions to a cube after you define optimization slices for the cube, DB2 Cube Views extends the existing optimization slices by specifying Any as the optimization levels for the new cube dimensions.

User Response

You can modify the optimization levels for the slices by clicking Specify on the Query Types page of the Cube Properties window.

10516: You must specify at least one optimization slice in the Optimization Slices window because you specified Advanced settings for the query type of the cube.

Explanation

You must specify at least one optimization slice in the Optimization Slices window because you selected Advanced settings for the query type of the cube.

User Response

Click **Specify** and create at least one optimization slice in the Optimization Slices window.

10517: Your current changes to the cube and cube dimensions will be saved when you leave the Dimensions page. Click Yes to save your changes in the database. Click No to remain on the Dimensions page to make more changes or to cancel.

Explanation

After you modify the cube dimensions in the Cube Properties window and leave the Dimensions page, all of your changes to the cube dimensions are saved in the database and cannot be undone by clicking **Cancel**.

User Response

Click **Yes** to save the current changes. Click **No** to cancel the transaction and remain on the Dimensions page.

10518: You cannot modify the cube hierarchy because you already defined advanced optimization slices for the cube_name cube. You must delete all of the advanced optimization slices, then you can modify the cube hierarchy.

Explanation

You cannot modify the cube hierarchy if you already defined advanced optimization slices for the cube.

User Response

You must delete all advanced optimization slices, then you can modify the cube hierarchy.

10519: You cannot modify the set of cube dimensions because you already defined advanced optimization slices for the *cube_name* cube. You must delete all of the advanced optimization slices, then you can add or delete cube dimensions to the cube.

Explanation

You cannot modify the set of cube dimensions if you already defined advanced optimization slices for the cube.

User Response

You must delete all advanced optimization slices, then you can add or delete cube dimensions to the cube.

10520: Your changes to the properties will also modify the existing optimization slices for the cube. You can review the optimization slices to view and modify the changes, or you can click Cancel on the properties window to close the window and cancel your changes.

Explanation

If you modify the set of cube dimensions or the cube hierarchy, the existing advanced optimization slices for that cube will be modified in one of the following ways:

- If a new cube dimension is added, all existing optimization slices will be extended to include the new cube dimension at the Any level.

- If a cube dimension is removed, the corresponding level will be removed from all existing optimization slices.
- If a cube hierarchy is altered so that the level that is defined in the optimization slices does not exist, the missing hierarchy level is replaced with the Any level in all applicable optimization slices.
- If an existing cube hierarchy is replaced with a new cube hierarchy, the levels in the optimization slices for the corresponding cube dimension will be set to Any.

User Response

Open the Optimization Slices window and verify the OLAP Center's modifications. Make any necessary changes.

Status messages from DB2 and DB2 Cube Views

When the DB2 Cube Views stored procedure is called, regardless of whether the stored procedure was executed, DB2 returns an SQLCODE and an SQLSTATE to the calling application. If the DB2 Cube Views stored procedure can execute, the stored procedure returns a status message as part of the XML data that is sent to the calling application.

The following table shows the relationship between the status messages that are returned by metadata operations and the SQLSTATE that is returned by DB2 for the call to the stored procedure.

Table 63. Metadata operation IDs versus SQLSTATE codes

SQL CODE	SQL STATE	Metadata operation status message IDs	Metadata operation status message types	Metadata operation status messages returned
0	0	0 2	Informational	No
0	0	1	Informational	Yes
0	0	599 6006 6299 7200 7201 7202	Warning	No
462	01HQ1	0 – 7999 (excluding IDs listed in other rows)	Error	No
443	38Q00	Not applicable	Not applicable	Not applicable
443	38Q01	Not applicable	Not applicable	Not applicable
443	38Q02	Not applicable	Not applicable	Not applicable
443	38Q03	Not applicable	Not applicable	Not applicable

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 give 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, 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.

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 information 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.

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.

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.

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 measurement 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

AIX
DB2
DB2 Connect
DB2 Universal Database
IBM
Office Connect

The following terms are trademarks or registered trademarks of other companies:

Microsoft, Windows, Windows NT, Windows 2000, Windows XP, and Microsoft Excel are trademarks or registered trademarks of Microsoft Corporation.

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

UNIX is a registered trademark in the United States, other countries, or both and is licensed exclusively through X/Open Company Limited.

Linux is a registered trademark of Linus Torvalds. Red Hat and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

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

Glossary

This glossary defines terms that are used in this book.

aggregation function. One of the DB2 SQL aggregation functions such as SUM, AVG, MIN, and MAX. The aggregation function is used to control how rollups are performed on measures.

attribute. A DB2 object that maps to either a single column in a table or an expression that is a combination of a set of columns or other attributes or bother. An attribute can perform a number of roles. For example, it can be a reference to data that is in the cube, or it can be a reference to a column that is used by a join or other attribute relationship.

attribute relationship. Describes relationships of attribute objects in general. The relationships are described by a left and a right attribute, a type, a cardinality, and whether they determine a functional dependency. The type describes what the role of the right attribute is with respect to the left attribute. There are two possible types: Descriptive and Associated. The Descriptive type specifies that the right attribute is a descriptor of the left attribute.

balanced hierarchy. A hierarchy with meaningful levels and branches that have a consistent depth. The logical parent of each attribute is in the level directly above it. See *network hierarchy*, *ragged hierarchy*, and *unbalanced hierarchy*.

calculated measure. Contains built-in calculations that you create by using the Expression Builder in the OLAP Center or with SQL. A calculated measure has an SQL expression that performs calculations and does not map to a single column or attribute.

constraint. A rule that the database manager enforces. There are four types of constraints: unique, referential, table check, and informational.

cube. A DB2 object that is derived from a cube model. The cube facts and cube dimensions are subsets of those that are referenced in the cube model. model. Cubes are appropriate for tools and applications that do not use multiple hierarchies because cube dimensions allow only one cube hierarchy per cube dimension.

cube dimension. A DB2 object that is part of a cube and is derived from a dimension in the cube model that corresponds to the cube. A cube dimension references a subset of the attributes of the dimension from which it is derived. It also references a single cube hierarchy.

cube facts. A DB2 object that is part of a cube and is derived from a dimension in the cube model that corresponds to the cube. A cube facts references a subset of the measures from the facts object from which it is derived.

cube hierarchy. A DB2 object that is part of a cube dimension and is derived from a hierarchy in the dimension that corresponds to the cube dimension. A cube hierarchy references a subset of the attributes of the hierarchy from which it is derived where the order of the attributes must be in the same order as their order in the hierarchy.

cube level. A DB2 object that is a subset of a level and is used in a cube. A cube level references the level from which it is derived (parent level) and inherits the level key attributes and default attribute that are defined for the parent level.

cube model. A DB2 object that describes all data related to a collection of measures. Typically, the cube model relates to a star schema or snowflake schema in the database. The cube model references a single facts object and one or more dimensions. Cube models can be optimized to improve the performance of SQL queries issued to the star schema or snowflake schema of the cube model.

dimension. A DB2 object that references a collection of related attributes that describe some aspect of a set of measures. A dimension can reference attributes from one or more dimension tables. However, if attributes from multiple dimension tables are used, the tables must have joins between them and those joins must be referenced by the dimension. A dimension also references one or more hierarchies and can reference relationships between its attributes.

dimension table. A table in a data warehouse whose entries describe data in a fact table. Dimension tables contain the data from which dimensions are created.

facts object. A DB2 object that groups related measures that are interesting to a specific application. The facts object stores information about the attributes that are used in fact to dimension joins, and the attributes and joins that are used to map the additional measures across multiple database tables. Therefore, in addition to a set of measures, a facts object stores a set of attributes and a set of joins. A facts object is used in a cube model as the center of a star schema.

fact table. A central table in a data warehouse schema that contains numerical measures and keys relating facts to dimension tables. Fact tables contain data that

describes specific events within a business, such as bank transactions or product sales.

functional dependency. A DB2 object that indicates that a level object's default attribute and related attributes are functionally determined by the level's key attributes. Functional dependencies allow you to specify that one or more columns are functionally dependent upon one or more other columns, provided all the columns exist within the same table.

hierarchy. A DB2 object that defines relationships among a set of one or more attributes within a specific dimension of a cube model. DB2 Cube Views supports four types of hierarchies: balanced, unbalanced, ragged, and network. Hierarchies can be deployed as either standard or recursive.

hybrid cube. Contains multidimensional data and references relational data so that you can query lower level data in your base tables.

join. Joins two relational tables. A join references attributes that then reference columns in the tables that are being joined. The simplest form of a join references two attributes: one that maps to a column in the first table and one that maps to a column in the second table. The join also includes an operator to indicate how the columns are compared. A join object can also be used to model composite joins where two or more columns from the first table are joined to the same number of columns in the second table. A composite join uses pairs of attributes to map corresponding columns. Each pair of attributes has an operator that indicates how that pair of columns are compared. A join also has a type and cardinality. Joins can be used in dimensions to join dimension tables together or in a cube model to join the dimensions of the cube model to its facts object or within a facts object to join multiple fact tables.

level. A DB2 object that consists of one or more attributes that are related and work together as one logical step in a hierarchy's ordering.

materialized query table. A table whose definition is based on the result of a query and whose data is in the form of precomputed results that are taken from one or more tables on which the materialized query table definition is based.

measure. A DB2 object that defines a measurement entity and is used in facts objects. Measures become meaningful within the context of a dimension. Common examples of measure objects are Revenue, Cost, and Profit.

metadata. Information about the properties of data, such as the type of data in a column (numeric, text, and so on) or the length of a column. It can also be information about the structure of data or information that specifies the design of objects such as cubes or dimensions.

MQT. See *materialized query table*.

network hierarchy. A hierarchy in which the order of levels is not specified, but in which levels do have semantic meaning. Because the attribute levels do not have an inherent parent-child relationship, the order of the levels is not important. See *balanced hierarchy, ragged hierarchy*, and *unbalanced hierarchy*.

outrigger table. Any dimension table in a snowflake schema that is not the primary dimension table in the dimension.

primary dimension table. In a snowflake schema, the dimension table that joins to the fact table.

ragged hierarchy. A hierarchy in which each level has a consistent meaning but the branches have inconsistent depths because at least one member attribute in a branch level is unpopulated. See *balanced hierarchy, network hierarchy*, and *unbalanced hierarchy*.

recursive deployment. Uses the inherent parent-child relationships between the attributes of the hierarchy. An unbalanced hierarchy that uses a recursive deployment is represented as parent-child attribute pairs.

schema. In the SQL-92 standard, a collection of database objects that are owned by a single user and form a single namespace. A namespace is a set of objects that cannot have duplicate names. For example, two tables can have the same name only if they are in separate schemas, no two tables in the same schema can have the same name.

snowflake schema. An extension of a star schema such that one or more dimensions are defined by multiple tables. In a snowflake schema, only primary dimension tables are joined to the fact table. Additional dimension tables are joined to primary dimension tables.

standard deployment. Uses the level definitions of the hierarchy where each attribute in the hierarchy defines one level. For example, a balanced hierarchy for a Time dimension is usually organized by each defined level including Year, Quarter, and Month. Standard deployment can be used with all four hierarchy types.

star join. A join between a fact table (typically a large fact table) and at least two dimension tables. The fact table is joined with each dimension table on a dimension key.

star schema. A relational database structure in which data is maintained in a single fact table at the center of the schema with additional dimension data stored in dimension tables. Each dimension table is directly related to and usually joined to the fact table by a key column. Star schemas are used in data warehouses.

summary table. Contains aggregated data of the base tables that are used by your cube model. DB2 Cube

Views uses DB2 summary tables to improve the performance of queries that are issued to cube models. A summary table is a special type of a materialized query table (MQT) that specifically includes summary data. Because DB2 Cube Views always recommends MQTs that have summarized data, the term summary table is used in the DB2 Cube Views documentation to describe the recommended MQTs. See *materialized query table*.

slice. A region of multidimensional database or cube.

unbalanced hierarchy. A hierarchy with levels that have a consistent parent-child relationship but have an inconsistent semantic meaning for all members in a particular level. Also, the hierarchy branches have inconsistent depths. See *balanced hierarchy, network hierarchy*, and *ragged hierarchy*.

Index

H

I

J

L

M

N

O

Contacting IBM

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

For information or to order any of the DB2 Cube Views products, contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

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

- 1-800-237-5511 for customer support
- 1-888-426-4343 to learn about available service options

Product Information

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

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

http://www.ibm.com/software/data/db2/db2md/
Provides links to information about DB2 Cube Views.

http://www.ibm.com/software/data/db2/udb
The DB2 Universal Database Web pages provide current information about news, product descriptions, education schedules, and more.

http://www.elink.ibmlink.ibm.com/
Click Publications to open the International Publications ordering Web site that provides information about how to order books.

http://www.ibm.com/education/certify/
The Professional Certification Program from the IBM Web site provides certification test information for a variety of IBM products.

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

Comments on the documentation

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

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

Program Number: 5724-E15

Printed in USA

SC18-7298-01